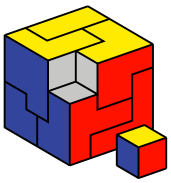
Prologin 2023



Concours national d'informatique

Correction de la phase de sélection

Table des matières

Questionnaire

Valérian Fayt, Quentin Rataud

Question 1: Pokémon

> Parmi les termes informatiques suivant, lequel est un Pokémon ?
> — Lodash
> — Dplyr
> — Onyx
> — Hadoop
> — **Ditto**
> — Flask

Ditto est le nom anglais du Pokémon Métamorph. *Lodash* est une bibliothèque JavaScript, *Dplyr* est une bibliothèque R, *OnyX* est un logiciel pour macOS, contrairement à son homonyme *Onix*, qui lui est un Pokémon. *Hadoop* et *Flask* sont deux frameworks, contrairement au Pokémon homonyme *Phlask*.

Question 2: I'm vengence

> Quel est le résultat de `Array(16).join("wat" - 1) + " Batman!"` en JavaScript ?
> — `Batman!`
> — Liste vide
> — Erreur de type
> — **`WATMAN`**
> — `NaN`
> — `15W`

Exécuter le code tel quel retourne la chaîne de caractère `'NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaN Batman!'`, qui ne correspond à aucune des réponses. Cependant, en recherchant ce résultat, le code de la question où certaines des réponses proposées, on peut trouver la conférence "Wat". [1] Dans celle-ci, le conférencier conclue par exécuter cette ligne de code en Javascript, et intitule le résultat de ce code "WATMAN".

Question 3: Short but intense

> Parmi les propositions suivantes, laquelle n'est pas un langage de programmation ?
> — `4`
> — **`#`**
> — `!`

1. `https://www.destroyallsoftware.com/talks/wat`

— `$`
— `_`
— `)`

Le `4` [2] est un langage de programmation nommé selon la dernière décimale de pi, où le programme prend la forme d'un nombre commençant par `3.` et se terminant par `4`. Le `!` [3] est aussi un langage de programmation, prononcé "not". Le `$` [4] est un langage proche du `‘`, utilisant uniquement deux instructions. Le `_` [5] est un langage où les différentes instructions rajoutent des caractères au code source lui-même. Enfin, le `)` [6] est un langage de programmation utilisant des caractères spéciaux comme mots-clés.

Question 4: Y'a de l'echo là non ?

Laquelle de ces propositions est le titre d'une conférence donnée en 2002 par un ingénieur logiciel chez Google ?

— Prologin : Travail Entraînement Fun Fun
— Red Green Blue : Black and White
— Zero Zero Zero Zero : One
— Echo : Alpha Beta Charlie Delta
— Three Two One Zero : Two Thousand
— **Chicken Chicken Chicken : Chicken Chicken**

Il s'agit d'une conférence présentée par Douglas E. Zongker. Aucune vidéo de la conférence de 2002 n'existe, mais des vidéos d'une seconde présentation en 2007 peuvent facilement être trouvées en ligne. Le pdf à l'origine de cette conférence, daté de 2002, peut être également trouvé en ligne. [7]

Question 5: Bitoduc

Qu'est-ce qu'un «code encoquillé» ?

— Un molusque qui se trouve sur les côtes bretonnes
— Un code comportant des erreurs de syntaxe
— Un code protégé
— **Shell code en français**
— C'est un langage de programmation fait pour la programmation spaghetti

Selon `bitoduc.fr`, «code encoquillé» est la traduction française de «shell code».

Question 6: Galactique

Quel problème a trouvé pour la première fois, en 2020, un algorithme de complexité quasi-linéaire utilisant une transformée de Fourier à plus de 1,000 dimensions ?

— Pourquoi Valérian a échangé son nom avec un panda
— Trier une liste contenant deux entiers
— Le problème du voyageur de commerce
— Le problème du flot de coût minimum
— **Multiplier deux entiers**
— L'exponentiation modulaire

C'est un exemple d'«Algorithme galactique», c'est à dire un algorithme ayant une complexité temporelle remarquable, mais inutilisable en pratique. On peut retrouver en ligne les papiers de la recherche. [8]

2. `https://esolangs.org/wiki/4`
3. `https://esolangs.org/wiki/!`
4. `https://esolangs.org/wiki/%60`
5. `https://esolangs.org/wiki/%EF%BC%BF`
6. `https://esolangs.org/wiki/)`
7. `https://isotropic.org/papers/chicken.pdf`
8. `https://hal.science/hal-02070778/document`

Question 7: Digging the archives

> Quelle question de ce QCM a déjà été posée lors des qualifications de l'édition 2004 ? Indiquer le titre de la question.
>
> — **X-pré-sion**

On peut retrouver la question dans les archives de Prologin. [9]

Question 8: Going back in time

> Que proposait la news du 06 octobre 2001, 21 :10 sur le site Prologin ?
>
> — Obtenir une affiche de l'édition 1998
>
> — **Obtenir un CD de l'édition 2001**
>
> — Récupérer un emplacement sur le serveur de Prologin
>
> — Recevoir une photo du président

On peut aisément retrouver la news grâce à Wayback Machine [10] « Demandez gratuitement votre CD PROLOGIN 2001, avec les photos, les vidéos, animation, sources du serveur, etc ... »

Question 9: Multilingue

> Dans quels langages peut être exécuté le programme ci-dessous ?
>
> ```
> *BUFFER : A.A ; .( Hello, world !) @ To Including?
> Macro SkipThis; OUTPUT = Char(10) "Hello, World !"
> ;OneKeyInput  Input('Char', 1, '[-f2-q1]')  ; Char
> End; SNOBOL4 + PureBASIC + Win32Forth + REBOL = <3
> EndMacro: OpenConsole() : PrintN("Hello, world !")
> Repeat : Until Inkey() :  Macro SomeDummyMacroHere
> REBOL  [ Title:  "'Hello, World !' in 4 languages"
> CopyLeft:  "Developed in 2010 by Society" ]  Print
> "Hello, world !"  EndMacro:  func  [][]  set-modes
> system/ports/input [binary: true] Input  set-modes
> system/ports/input  [binary: false] NOP:: EndMacro
> ; Wishing to refine it with new language ? Go on !
> ```
>
> — Pascal, REBOL, ASM68k, C
>
> — SNOBOL4, REBOL, Perl, JavaScript
>
> — PureBasicv4.x, C, PHP, JavaScript
>
> — Perl, REBOL, Win32Forth, ASM68k
>
> — **SNOBOL4, Win32Forth, PureBasicv4.x, REBOL**

C'est un exemple de programme *multilangage*. On retrouve ce code en particulier dans la page Wikipédia anglaise. [11]

Question 10: I'll take everything

> Comment appelle-t-on un paquet réseau dont toutes les options sont définies ?
>
> — Cisco Packet
>
> — Paquet TTC
>
> — Easter egg packet
>
> — Deep packet
>
> — **Christmas tree packet**

9. https://prologin.org/contest/2004/qualification/quiz/
10. http://web.archive.org/web/20020212053558/http://www.prologin.org/news.php
11. https://en.wikipedia.org/wiki/Polyglot_(computing)

Un *Christmas tree packet* est défini par Eric S. Raymond comme étant un paquet ayant toutes les options définies, à la manière d'un sapin de Noël avec toutes les guirlandes éléctriques allumées. [12]

Question 11: Quine

> Lequel de ces codes affiche son propre code source ?
> — `!function pikalul() alert("!"+pikalul+"()") ()`
> — `(function pikalul()return'('+pikalul+')()') ()`
> —
> — `(pika=lul=>alert("(pika = $pika)()"))()`

Si tous les codes sont proche d'être des quines, il existe pour chacun d'entre eux au moins une différence subtile entre le code et le code affiché. Les deux premiers codes possèdent un espace en trop avant les dernières parenthèses. La réponse est donc la réponse vide. En fait, il s'agit même ici d'un code titré « Meilleur abus des règles » aux IOCCC de 1994 [13]. D'à partir de 1995, principalement grâce à cette soumission, les règles des IOCCC précisent que les soumissions doivent contenir au moins un caractère.

Question 12: X-pré-sion

> Laquelle de ces expressions régulières reconnaît son propre motif en entier ?
> — `a*`
> — `[.*]`
> — `[A-Za-z0-9]*`
> — **`(a|[^a])*`**

- `a*` reconnaît toutes les chaînes composées uniquement d'un certain nombre de `a`. Cette expression ne reconnaît donc pas son astérisque.
- `[.*]` ne reconnaît que les chaînes composées d'un unique caractère compris dans les crochets : un point ou un astérisque. Cette expression ne peut pas reconnaître ses crochets.
- `[A-Za-z0-9]*` reconnaît toutes les chaînes composées d'un certain nombre de majuscules, minuscules ou chiffres. Elle ne reconnaît donc pas ses tirets, ses crochets ni son astérisque.
- Enfin, `(a|[^a])*` reconnaît toutes les chaînes de caractères composées d'un certain nombre de « a, ou n'importe quel caractère n'étant pas un a ». En clair, cette expression peut reconnaître toutes les chaînes de caractères, et donc elle-même.

On peut d'ailleurs simplement analyser et tester ces expressions régulières sur un site comme `regex101.com`.

12. `http://www.catb.org/jargon/html/C/Christmas-tree-packet.html`
13. `https://github.com/c00kiemon5ter/ioccc-obfuscated-c-contest/blob/master/1994/smr.hint`

Capture The Flags

Julie Durandeau, Julie Fiadino, Romain Le Miere, Quentin Rataud, Odric Roux-Paris

1.1 Crackme

Ce challenge est un crackme basique. Il demande à l'utilisateur un `serial` et le valide.

1.1.1 Les anti-debuggers

Getenv

`Gdb` ajoute deux variables d'environnement au programme débuggé : `LINES` et `COLUMNS`. Pour cela, la fonction `getenv` est utilisée. Si la variable existe, un pointeur sur la valeur est retourné sinon `NULL` est renvoyé. Or `NULL` est égal à `((void *) 0)`, qui peut être casté en 0.

La protection consiste à multipler le résultat de `getenv` par une valeur arbitraire (`0xcafebabe` et `0xbabecafe`). Ensuite, on se sert de cette valeur comme indice pour accéder à cette chaîne de caractères :
`"The serial is : **********"`.

Si les variables `LINES` et `COLUMNS` sont présentes, alors le produit de la valeur arbitraire est trop grande et produit un `segfault`. Sinon, le produit est égal à 0 et l'accès au string se fait sans soucis.

Il existe plusieurs manières de patcher la protection. Par exemple, les variables d'environnement peuvent être supprimées directement dans `gdb` avec la commande `unset environment <VAR>`.

Ptrace

On peut remarquer avec `strace` que `ptrace` est appelé avec comme paramètre `PTRACE_TRACEME` et que le crackme plante juste après.

```
$ strace ./crackme
...
read(0, f"f\n", 1024)                    = 2
lseek(0, -1, SEEK_CUR)                   = -1 ESPIPE (Illegal seek)
ptrace(PTRACE_TRACEME)                   = -1 EPERM (Operation not permitted)
 --- SIGSEGV {si_signo=SIGSEGV, si_code=SI_KERNEL, si_addr=NULL} ---
 +++ killed by SIGSEGV (core dumped) +++
 [2]    47837 segmentation fault (core dumped)  strace ./crackme
```

C'est un anti-debugger basique, `PTRACE_TRACEME` permet de savoir si un programme est tracé. En s'arrêtant un peu avant le `syscall`, le code assembleur ressemble à :

```
push   $0xca9b
mov    (%rsp),%rdi
xor    $0xcafe,%rdi
push   %rdi
pop    %rax
xor    %rdi,%rdi
syscall
mov    %rax,%rdx
```

```
pop    %rax
movabs $0xabababababababa,%rax
imul   %rax,%rdx
movzbl 0x402030(%rdx),%eax
```

Le `syscall` de `ptrace` est `0x65`. Or l'opération `xor` de `0xca9b` et `0xcafe` est égal à `0x65` et il est mis dans `rax`. Le premier paramètre du `syscall` est mis à 0, ce qui correspond à `PTRACE_TRACEME`.

La méthode pour faire planter le crackme est la même que la précédente. Il existe plusieurs manières de supprimer cet anti-debugger. L'instruction `mov %rax,%rdx` peut être remplacée par `xor %rdx, %rdx` après le `syscall` (les deux instructions font 3 bytes chacune).

1.1.2 Junk code / Evil ASM

En fouillant dans le binaire, on peut remarquer de l'assembleur qui ne semble pas correct. Les décompilateurs comme celui de `ghidra` ont beaucoup de mal à les analyser et produisent des résultats farfelus. Ces instructions ont été enfaite pensées pour tromper les désassembleurs et les décompilateurs. Elles peuvent être remplacées par des `nop` afin que les décompilateurs puissent mieux comprendre le code.

Voici un exemple de `drunk code` :

```
xor    $0x2d,%rdx
add    $0x4,%rdx
lea    (%rsp),%rdx
je     4012bf <exit@plt+0x23f>
cmp    %rax,%rdx
jne    4012bb <exit@plt+0x23b>
rdtsc
add    $0x3d4,%rax
add    $0x38,%rdx
pop    %rax
xor    $0x345ad,%rax
xor    $0x345ad,%rax
push   %rax
```

Que fait ce code ? Bonne question !

1.1.3 L'algorithme vérifiant le serial.

Dans cette correction, nous allons utiliser `Ghidra` et principalement son décompilateur. Après l'importation du binaire, la fonction `main` décompilée ressemble à ça :

```
int main(void)
{
  int iVar1;
  int extraout_EAX;
  size_t len_serial;
  char *pcVar2;
  long lVar3;
  long unaff_RBP;
  undefined8 *puVar4;
  long in_FS_OFFSET;
  char cVar5;
  bool bVar6;
  byte bVar7;
  undefined8 serial;
  undefined8 local_410;
  undefined8 local_408 [127];
  long local_10;

  bVar7 = 0;
  local_10 = *(long *)(in_FS_OFFSET + 0x28);
  /* Premier anti-debugger avec getenv
   * Ghidra ne comprend pas la suite d'instruction et l'ignore.
```

```
  */
  getenv("LINES");
  getenv("COLUMNS");
  puts("Enter your Serial: ");
  serial = 0;
  local_410 = 0;
  puVar4 = local_408;
  for (lVar3 = 0x7e; lVar3 != 0; lVar3 = lVar3 + -1) {
    *puVar4 = 0;
    puVar4 = puVar4 + (ulong)bVar7 * -2 + 1;
  }
  iVar1 = __isoc99_scanf("%1024s",&serial);
  bVar7 = (byte)lVar3;
  fflush(stdin);
  if (iVar1 == -1) {
                    /* WARNING: Subroutine does not return */
    exit(1);
  }
                    /* Deuxième anti-debugger avec ptrace. */
  syscall();
  len_serial = strlen((char *)&serial);
  check_serial((char *)&serial,(int)len_serial);
  if (extraout_EAX == 0) {
    fail();
  }
  else {
    success();
  }
  /*
   * Le drunk/evil asm est ici.
   */
  cVar5 = '\0';
  bRam0000000000000001 = bRam0000000000000001 & bVar7;
  bVar6 = bRam0000000000000001 == 0;
  pcVar2 = (char *)(**(code **)(unaff_RBP + -0x3f00b762))();
  if (bVar6) {
    *pcVar2 = *pcVar2 + bVar7 + cVar5;
  }
  if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
                    /* WARNING: Subroutine does not return */
    __stack_chk_fail();
  }
  return 0;
}
```

Après un nettoyage, on se rend compte où se situe :

- les deux anti-debuggers
- la fonction affichant le succès et l'échec
- la fonction vérifiant le sérial
- le `drunk/evil asm`.

Regardons de plus près la fonction `check_serial`. De même ici, du `drunk/evil asm` a été introduit. On peut le remarquer grâce aux erreurs de `Ghidra` à la décompilation (`/* WARNING: Bad instruction - Truncating control flow here */`). Cette suite d'instruction représente le `drunk/evil asm` :

```
push   %rax
push   %rbx
call   145d <exit@plt+0x34d>
jmp    5890de5e <stdin@GLIBC_2.2.5+0x58909e4e>
pop    %rax
pop    %rbx
xchg   %rax,%rbx
```

On remplace ces instructions par des nop. La fonction décompilée ressemble maintenant à ça :

```
int check_serial(char *serial,int len_serial)
{
  int k;
  long j;
  long *intput_serial;
  undefined4 in_register_00000034;
  long i;
  long in_FS_OFFSET;
  long *mat [25];
  long local_10;

  intput_serial = mat;
  local_10 = *(long *)(in_FS_OFFSET + 0x28);
  res = 0;
  // Le serial doit faire 25 caractères.
  if (CONCAT44(in_register_00000034,len_serial) == 0x19) {
    i = 0;
    do {
      j = 0;
      do {
        intput_serial[j] = serial[j];
        j = j + 1;
      } while (j != 5);
      i = i + 5;
      serial = serial + 5;
      intput_serial = intput_serial + 5;
    } while (i != 0x19);
    res = FUN_00101370(mat,&DAT_00102060);
  }
  if (local_10 == *(long *)(in_FS_OFFSET + 0x28)) {
    return res;
  }
}
```

Il y a deux boucles qui vont de 0 à 5 exclus. Leur but est de transformer le `serial` en une matrice de 5 par 5. Au niveau de la mémoire, la matrice est représentée par un tableau de 25 cases à la suite.

On remarque que la matrice consruite à partir du `serial` et l'emplacement mémoire `DAT_00102060` sont passés en paramètre à la fonction `FUN_00101370`. On déduit qu'elle va vérifier le sérial.

Après analyse de la fonction `FUN_00101370`, on obtient le pseudocode suivant :

```
int FUN_00101370(long *mat1,long *mat2)
{
  undefined *in_RAX;
  long *mat1';
  long *mat2'';
  long in_RDX;
  long sum;
  long *mat1'';
  long j;
  long *mat2';
  long i;

  if ((in_RDX != -0x2f) && (&stack0xfffffffffffffff8 == in_RAX)) {
    rdtsc();
  }
  mat1'' = mat1 + 5;
  i = 0;
  do {
    j = 0;
    mat2' = mat2;
```

```
    do {
      sum = 0;
      mat1' = mat1;
      mat2'' = mat2';
      do {
        sum = sum + *mat1' * *mat2'';
        mat1' = mat1' + 1;
        mat2'' = mat2'' + 5;
      } while (mat1' != mat1'');
      if ((i != j) && (sum != 0)) {
        return 0;
      }
      if ((i == j) && (sum != 0x1cb372d6)) {
        return 0;
      }
      j = j + 1;
      mat2' = mat2' + 1;
    } while (j != 5);
    i = i + 1;
    mat1 = mat1 + 5;
    mat1'' = mat1'' + 5;
    if (i == 5) {
      return 1;
    }
  } while( true );
}
```

L'instruction `rdtsc` n'est là que pour faire jolie, elle ne sert à rien. Elle fait donc partie d'un `drunk/evil asm`.

Cette fois-ci nous avons 3 boucles. Pour la première matrice `mat1`, dans la 3^e boucle son pointer est incrementé de 1 en 1. Cependant dans la 1^{re} boucle, on s'aperçoit que le pointer est incrémenté de 5 en 5. On déduit que c'est un parcours ligne par ligne.

Pour la deuxième matrice, c'est l'inverse. On l'incrémente de 5 par 5 dans la 3^e boucle, puis de 1 par 1 dans la 2^e boucle. Elle est parcourue colonne par colonne.

On a donc une matrice parcourue ligne par ligne et l'autre parcourue colonne par colonne. Les valeurs des matrices sont multipliées entre elles puis ajoutées à une somme. Ce qui est un produit de matrice. Les conditions `((i != j) && (sum != 0))` et `((i == j)) && (sum != 0x1cb372d6)` vérifient si la diagonale du produit des deux matrices est égale à `0x1cb372d6`.

Pour résumé, le `serial` est validé si cette condition est remplie :

$$M_{\text{serial}} \cdot M_{\text{présente}} = I_5 \cdot 0x1cb372d6$$

Où I_5 est la matrice identité et $M_{\text{présente}}$ la matrice à l'emplacement mémoire `DAT_00102060`.

1.1.4 Construire le sérial.

La matrice présente ($M_{\text{présente}}$) dans le binaire est égale à :

$$\begin{pmatrix} -18414610 & 5968619 & 6761505 & -207835 & 6482846 \\ 22936990 & -7371334 & 4344000 & -17038250 & 1542684 \\ -5258160 & -132300 & -5469710 & 21093310 & -11688630 \\ -5943210 & 8154961 & 199625 & -2958365 & 1816744 \\ 11910590 & -6105892 & -6078390 & -351550 & 1774702 \end{pmatrix}$$

On remarque que les nombres font 8 bytes chacun soit la taille d'un `size_t`.

On va l'inverser mais avant il faut la diviser par `0x1cb372d6`. En effet, car

$$M_{\text{serial}} \cdot M_{\text{présente}} = I_5 \cdot 0x1cb372d6 \Leftrightarrow \frac{1}{0x1cb372d6} \cdot M_{\text{serial}} \cdot M_{\text{présente}} = I_5$$

L'inversion donne :

$$\begin{pmatrix} 80 & 82 & 79 & 76 & 79 \\ 71 & 73 & 78 & 123 & 65 \\ 114 & 101 & 89 & 40 & 41 \\ 117 & 82 & 101 & 65 & 100 \\ 121 & 63 & 63 & 63 & 125 \end{pmatrix}$$

En la convertissant en chaine de caractère on obtient le flag `PROLOGIN{AreY()uReAdy???}`.
Voici un script qui permet de calculer le flag :

```
import numpy as np
from numpy.linalg import inv

MULTIPLICATOR = 0x1cb372d6
SIZE_OF_SSIZE_T = 8
NB_COLUMN = 5

mat_bytes = b"""
\xee\x03\xe7\xfe\xff\xff\xff\xff\xeb\x12\x5b\x00\x00\x00
\x00\x00\x21\x2c\x67\x00\x00\x00\x00\x00\x25\xd4\xfc\xff
\xff\xff\xff\xff\x9e\xeb\x62\x00\x00\x00\x00\x00\x9e\xfd
\x5d\x01\x00\x00\x00\x00\xba\x85\x8f\xff\xff\xff\xff\xff
\xc0\x48\x42\x00\x00\x00\x00\x00\x56\x04\xfc\xfe\xff\xff
\xff\xff\x1c\x8a\x17\x00\x00\x00\x00\x00\x50\xc4\xaf\xff
\xff\xff\xff\xff\x34\xfb\xfd\xff\xff\xff\xff\xff\xf2\x89
\xac\xff\xff\xff\xff\xff\xbe\xdb\x41\x01\x00\x00\x00\x00
\x4a\xa5\x4d\xff\xff\xff\xff\xff\x56\x50\xa5\xff\xff\xff
\xff\xff\x51\x6f\x7c\x00\x00\x00\x00\x00\xc9\x0b\x03\x00
\x00\x00\x00\x00\xe3\xdb\xd2\xff\xff\xff\xff\xff\xa8\xb8
\x1b\x00\x00\x00\x00\x00\xbe\xbd\xb5\x00\x00\x00\x00\x00
\xdc\xd4\xa2\xff\xff\xff\xff\xff\x4a\x40\xa3\xff\xff\xff
\xff\xff\xc2\xa2\xfa\xff\xff\xff\xff\xff\x6e\x14\x1b\x00
\x00\x00\x00\x00
""".replace(b"\n", b"")

mat = []
k = 0
for _ in range(NB_COLUMN):
    line = []
    for _ in range(NB_COLUMN):
        line.append(
            int.from_bytes(mat_bytes[k:k + SIZE_OF_SSIZE_T],
                           byteorder="little",
                           signed=True) / MULTIPLICATOR)
        k += SIZE_OF_SSIZE_T
    mat.append(line)

mat = np.array(mat)
mat = inv(mat)
flag = ""
for line in mat:
    flag += "".join([chr(int(i)) for i in line])

print(f"The flag is \"{flag}\"")
```

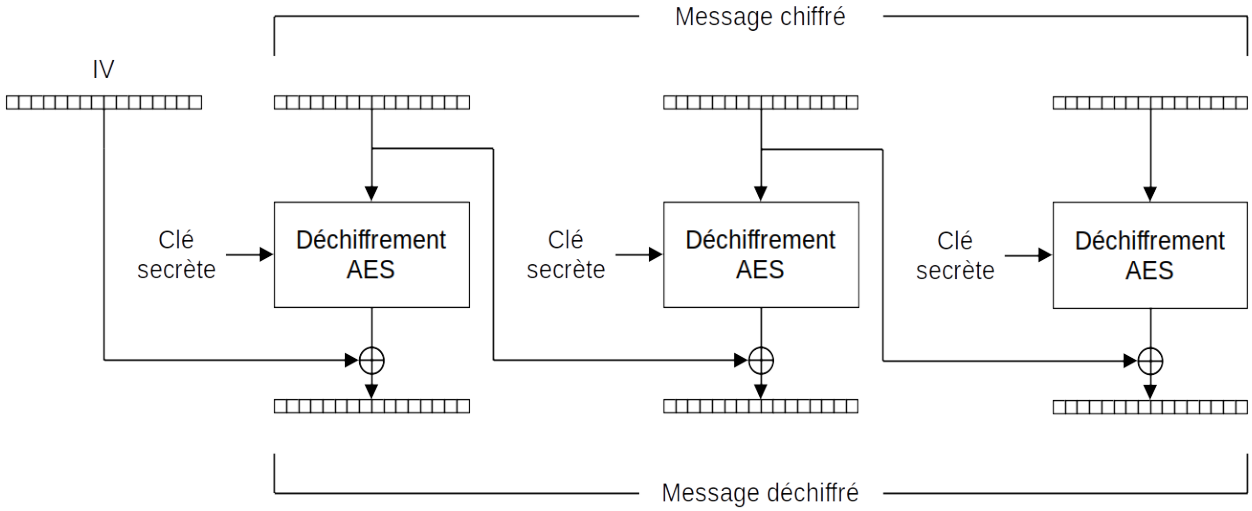


FIGURE 1.1 – Le déchiffrement AES - CBC

1.2 Oracle

1.2.1 Situation

Oh non ! Alors que toute l'équipe travaillait sur la machine spatio-temporelle, vous vous faîtes avoir par un méchant rançongiciel ! Vos données sont dès à présent chiffrées par un algorithme d'encryption infaillible, et le seul moyen de les déchiffrer est de leur payer une somme colossale... ... ou pas ? Heureusement, vous connaissez un grand oracle du rembourrage dans les environs qui est capable de déchiffrer de l'AES de tête. Essayez de retrouver vos précieuses données !

1.2.2 Fonctionnement de l'API

Dans le fichier `server.py`, on peut constater qu'il existe deux types de requêtes possible afin d'obtenir plus d'informations :

- Un `GET`, qui donne accès à la variable `PASSWORD`, contenant le drapeau chiffré : c'est ainsi que le jeu génère la clé chiffrée ;
- Un `POST`, qui appelle la fonction `decrypt`, prenant en paramètre un message chiffré : c'est la fonction qui est appelée par l'oracle.

Nous pouvons donc obtenir le drapeau chiffré en une simple requête.
En regardant plus en detail la fonction `decrypt`, on peut comprendre son fonctionnement :

1. Elle essaye de déchiffrer le message chiffré ;
2. Si le déchiffrement a échoué, la fonction retourne l'erreur rencontrée ;
3. Sinon, si la constante `ADMIN` vaut `True`, la fonction retourne le message déchiffré ;
4. Sinon, la fonction retourne une erreur indiquant un problème de permissions.

On peut constater que le chiffrement utilisé est le chiffrement AES, suivant le mode d'opération CBC, dit *Enchaînement des blocs*. La figure 1.1 schématise le fonctionnement du déchiffrement :

1. Le drapeau chiffré est découpé en blocs de 16 caractères
2. Chaque bloc de 16 caractère est déchiffré individuellement par l'AES
3. Chaque résultat se voit ensuite appliqué un *OU Exclusif* avec le bloc chiffré précédent (ou le Vecteur d'Initialisation pour le premier bloc)

Padding Après avoir déchiffré le drapeau, il reste encore les caractères de *padding* à la fin du message à enlever.
La méthode de chiffrement utilisée découpe le message en blocs de 16 caractères. Le padding est utilisé afin de remplir entièrement le dernier bloc, même si la longueur du message n'est pas un multiplie de 16 caractères. S'il manque 1 caractère à remplir, alors on rajoute une fois le caractère 0x01. S'il manque 5 caractères à remplir, alors on rajoute cinq fois le caractère 0x05, etc.

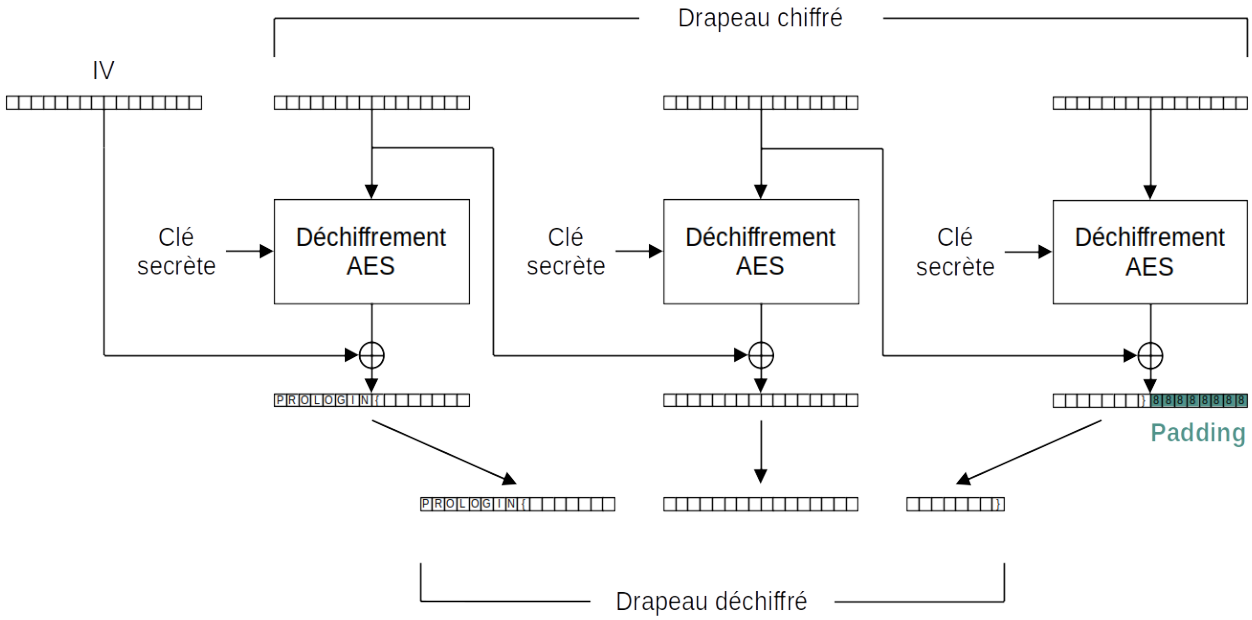


FIGURE 1.2 – Le padding dans le déchiffrement AES-CBC

Dans la figure 1.2, le caractère 0x08 est représenté par un 8. Le message se termine par huit fois le caractère 0x08, et possède donc un padding valide.

Lors du déchiffrement, la fonction `unpad` s'attend à ce que le dernier bloc contienne un padding valide. Dans le cas contraire, la fonction `decrypt` renverra une erreur de padding.

Par exemple, le bloc `31 41 59 26 5A 3F 92 09 09 09 09 09 09 09 09 09` possède un padding valide, car il se termine par neuf caractères 0x09.

1.2.3 La faille

Malheureusement, la constante `ADMIN` valant `False`, la fonction `decrypt` ne nous permettra pas de directement déchiffrer le drapeau. De plus, sans la clé secrète, il nous est impossible d'effectuer le processus nous même.

Cependant, nous pouvons tout de même obtenir des informations cruciales : la fonction tente tout de même de déchiffrer le message même si nous n'en avons pas les permissions, et nous indiquera quand même si le message chiffré renseigné est erroné. La seule erreur que la fonction peut envoyer est l'erreur de *padding* : si le message déchiffré possède un *padding* invalide, la fonction `unpad` retournera une erreur, auquelle nous aurons accès, indiquant que le padding est incorrect.

Le simple fait de pouvoir savoir si le *padding* engendré par un quelconque message est valide ou non nous est suffisant pour effectuer une *Padding Oracle Attack*, afin de déchiffrer caractère par caractère le drapeau dans son entièreté.

Détecter le padding On peut voir dans la figure 1.3 que modifier un bit précis de l'avant dernier bloc du message chiffré modifiera uniquement le bit correspondant dans le dernier bloc du message déchiffré. Ainsi, en modifiant l'avant dernier caractère de l'avant dernier bloc, nous sommes sûrs de modifier uniquement l'avant dernier caractère du dernier bloc du message déchiffré. Ce faisant, le padding ne sera valide que si le padding était composé d'un seul caractère. Sinon, cela indique que le padding est composé de plus d'un caractère. Dans la figure 1.4, on peut constater que, étant donné que le padding était composé de plus d'un caractère, modifier l'avant dernier caractère de l'avant dernier bloc avant d'utiliser la fonction `decrypt` engendre une erreur. Au contraire, dans la figure 1.3, on constate que modifier le 5e caractère du drapeau chiffré n'invalide pas le *padding*, car celui-ci fait moins de 12 caractères.

En altérant tour à tour chaque caractère de l'avant-dernier bloc avant de l'envoyer dans la fonction de déchiffrement, on obtient le résultat suivant :

```python
import requests

# L'URL de base.
ENDPOINT = "https://ctf.prologin.org/password"

# On recupere le mot de passe depuis le serveur.
def get_password():
```

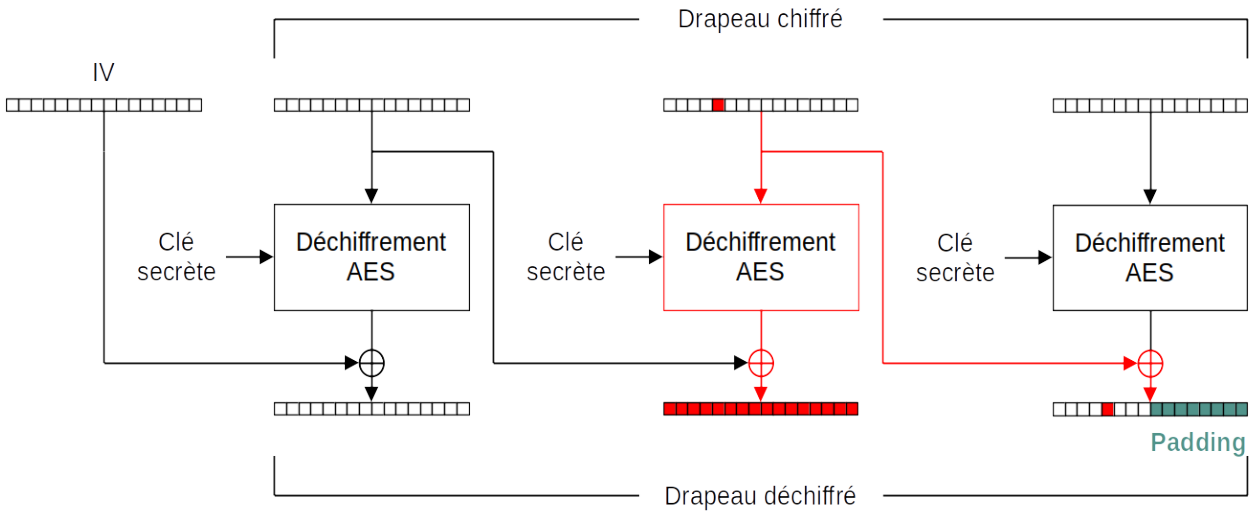


FIGURE 1.3 – Effets de la modification d'un caractère dans le déchiffrement AES-CBC

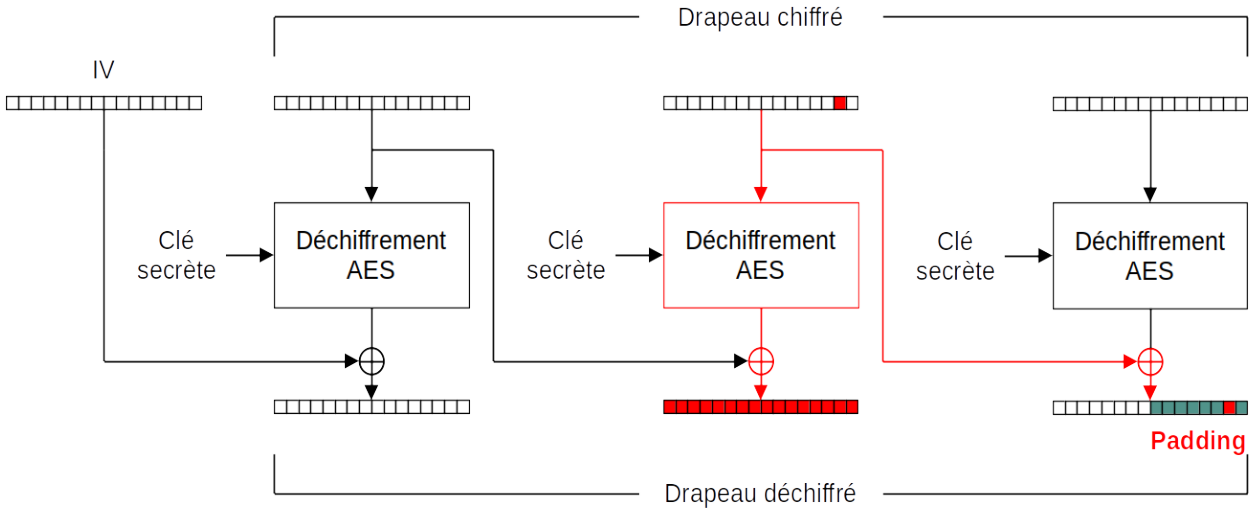


FIGURE 1.4 – Invalidation du *padding* par modification d'un caractère

```
    res = requests.get(ENDPOINT)
    return res.json()

# On demande au serveur de decrypter la cle.
def decrypt(iv, ciphertext):
    res = requests.post(ENDPOINT, json={"iv": iv, "ciphertext": ciphertext})
    return res.json()['error']

PASSWORD = get_password()


iv = PASSWORD['iv']
ciphertext = list(bytes.fromhex(PASSWORD['ciphertext']))

for length in range(16):
    ciphertext[-17-length] ^= 1
    print(length, decrypt(iv, bytes(ciphertext).hex()))
    ciphertext[-17-length] ^= 1

0 {'error': 'Cette clé est incorrecte. Tu t'es fait avoir, ça se voit à l'œil nu!'}
```

```
1 {'error': 'Cette clé est incorrecte. Tu t'es fait avoir, ça se voit à l'œil nu!'}
2 {'error': 'Cette clé est incorrecte. Tu t'es fait avoir, ça se voit à l'œil nu!'}
3 {'error': 'Cette clé est incorrecte. Tu t'es fait avoir, ça se voit à l'œil nu!'}
4 {'error': 'Cette clé est incorrecte. Tu t'es fait avoir, ça se voit à l'œil nu!'}
5 {'error': 'Cette clé est incorrecte. Tu t'es fait avoir, ça se voit à l'œil nu!'}
6 {'error': 'Cette clé est incorrecte. Tu t'es fait avoir, ça se voit à l'œil nu!'}
7 {'error': 'Cette clé est incorrecte. Tu t'es fait avoir, ça se voit à l'œil nu!'}
8 {'error': 'J'ai pu la déchiffrer sans aucun problème. Cependant, je te la donnerais
            seulement contre 999,999.58$.'}
9 {'error': 'J'ai pu la déchiffrer sans aucun problème. Cependant, je te la donnerais
            seulement contre 999,999.58$.'}
10 {'error': 'J'ai pu la déchiffrer sans aucun problème. Cependant, je te la donnerais
            seulement contre 999,999.58$.'}
11 {'error': 'J'ai pu la déchiffrer sans aucun problème. Cependant, je te la donnerais
            seulement contre 999,999.58$.'}
12 {'error': 'J'ai pu la déchiffrer sans aucun problème. Cependant, je te la donnerais
            seulement contre 999,999.58$.'}
13 {'error': 'J'ai pu la déchiffrer sans aucun problème. Cependant, je te la donnerais
            seulement contre 999,999.58$.'}
14 {'error': 'J'ai pu la déchiffrer sans aucun problème. Cependant, je te la donnerais
            seulement contre 999,999.58$.'}
15 {'error': 'J'ai pu la déchiffrer sans aucun problème. Cependant, je te la donnerais
            seulement contre 999,999.58$.'}
```

On peut en conclure que modifier n'importe lequel des 8 derniers caractères du drapeau entraine une erreur de padding, tandis que modifier n'importe lequel des 8 premiers caractères du dernier bloc n'entraine pas d'erreur de padding.

Cela signifie que le drapeau déchiffré contient originalement 8 caractères de padding.

Étant donné que le drapeau chiffré est composé de trois blocs, le drapeau doit avoir une longueur de 40 caractères.

Déchiffrer le drapeau Maintenant, nous pouvons utiliser un principe similaire pour déchiffrer caractère par caractère le drapeau.

Sachant que le dernier bloc se termine obligatoirement par 8 fois le caractère 0x08, nous pouvons effectuer un *OU Exclusif* sur chacun des 8 derniers caractères de l'avant dernier bloc avec `0x01` afin de changer les 8 derniers caractères du message déchiffré en huit fois le caractère 0x09.

Ce faisant, nous savons que le message possède un padding incorrect, à moins que le 8e caractère du dernier bloc du drapeau déchiffré soit déjà 0x09, auquel cas le padding serait correct.

Nous pouvons utiliser ce fait pour vérifier si ce caractère possède une valeur précise. Par exemple, nous pouvons vérifier que le dernier caractère du drapeau déchiffré est bien }, 0x7D : Pour ce faire, nous effectuons un *OU Exclusif* sur le 8e caractère de l'avant dernier bloc du drapeau chiffré avec la valeur 0x74 (*OU Exclusif* de 0x7D et 0x09), et nous pouvons vérifier que le padding est bien correct.

```python
import requests

# L'URL de base.
ENDPOINT = "https://ctf.prologin.org/password"

# On recupere le mot de passe depuis le serveur.
def get_password():
    res = requests.get(ENDPOINT)
    return res.json()

# On demande au serveur de decrypter la cle.
def decrypt(iv, ciphertext):
    res = requests.post(ENDPOINT, json={"iv": iv, "ciphertext": ciphertext})
    return res.json()

PASSWORD = get_password()


iv = PASSWORD['iv']
```

```python
ciphertext = list(bytes.fromhex(PASSWORD['ciphertext']))

pad = 8
# On change les 8 caracteres 0x08 du padding en des 0x09
for i in range(pad):
    ciphertext[-17-i] ^= pad ^ (pad + 1)

value = 0x7D # `}`
ciphertext[-17-pad] ^= value ^ (pad + 1)

print(decrypt(iv, bytes(ciphertext).hex()))
```

```
{'error': 'J'ai pu la déchiffrer sans aucun problème. Cependant, je te la donnerais
           seulement contre 999,999.58$.'}
```

De la même maniere, nous pouvons désormais trouver le caractère précédent par force brute. Nous pouvons encore une fois augmenter la taille du padding, car nous connaissons le dernier caractère du drapeau. Apres avoir mis les 9 derniers caractères à 0x09, nous pouvons appliquer un *OU Exclusif* sur ces 9 caractères avec la valeur 0x03 afin d'obtenir 9 fois le caractère 0x10, et donc un padding invalide (à moins que l'avant dernier caractère du drapeau soit également un 0x10.)

En vérifiant pour chaque caractère possible si le padding est correct, on obtient ce resultat :

```python
import requests

# L'URL de base.
ENDPOINT = "https://ctf.prologin.org/password"

# On recupere le mot de passe depuis le serveur.
def get_password():
    res = requests.get(ENDPOINT)
    return res.json()

# On demande au serveur de decrypter la cle.
def decrypt(iv, ciphertext):
    res = requests.post(ENDPOINT, json={"iv": iv, "ciphertext": ciphertext})
    return res.json()

PASSWORD = get_password()


iv = PASSWORD['iv']
ciphertext = list(bytes.fromhex(PASSWORD['ciphertext']))

pad = 8
# On change les 8 caracteres 0x08 du padding en des 0x09
for i in range(pad):
    ciphertext[-17-i] ^= pad ^ (pad + 1)

# On change le dernier caractere en un 0x09
last_value = 0x7D
ciphertext[-17-pad] ^= last_value ^ (pad + 1)

pad += 1
# On change les desormais 9 caracteres 0x09 de padding en des 0x10
for i in range(pad):
    ciphertext[-17-i] ^= pad ^ (pad + 1)

# On teste toute les valeur possible pour l'avant-dernier caractere
for value in range(256):
    ciphertext[-17-pad] ^= value ^ (pad + 1)
    print(hex(value), decrypt(iv, bytes(ciphertext).hex()))
    ciphertext[-17-pad] ^= value ^ (pad + 1)
```

```
...
0x1d {'error': 'Cette clé est incorrecte. Tu t'es fait avoir, ça se voit à l'œil nu!'}
0x1e {'error': 'Cette clé est incorrecte. Tu t'es fait avoir, ça se voit à l'œil nu!'}
0x1f {'error': 'Cette clé est incorrecte. Tu t'es fait avoir, ça se voit à l'œil nu!'}
0x20 {'error': 'Cette clé est incorrecte. Tu t'es fait avoir, ça se voit à l'œil nu!'}
0x21 {'error': 'J'ai pu la déchiffrer sans aucun problème. Cependant, je te la donnerais
                seulement contre 999,999.58$.'}
0x22 {'error': 'Cette clé est incorrecte. Tu t'es fait avoir, ça se voit à l'œil nu!'}
0x23 {'error': 'Cette clé est incorrecte. Tu t'es fait avoir, ça se voit à l'œil nu!'}
0x24 {'error': 'Cette clé est incorrecte. Tu t'es fait avoir, ça se voit à l'œil nu!'}
0x25 {'error': 'Cette clé est incorrecte. Tu t'es fait avoir, ça se voit à l'œil nu!'}
...
```

L'avant dernier caractère est donc 0x21, donc un point d'exclamation !
On continue en suivant le même principe afin de déchiffrer l'entièreté du drapeau.

1.2.4 Implémentation

```python
import requests

# L'URL de base.
ENDPOINT = "https://ctf.prologin.org/password"

# On recupere le mot de passe depuis le serveur.
def get_password():
    res = requests.get(ENDPOINT)
    return res.json()

# On demande au serveur de decrypter la cle.
def decrypt(iv, ciphertext):
    res = requests.post(ENDPOINT, json={"iv": iv, "ciphertext": ciphertext})
    return res.json()

PASSWORD = get_password()


def divide_blocks(text):
    blocs = []
    for i in range(0, len(text), 16):
        blocs.append(list(text[i:i+16]))
    return blocs


def valid_padding(cipher_blocs):
    iv = bytes(cipher_blocs[0]).hex()
    ciphertext = ''.join(map(lambda bloc: bytes(bloc).hex(), cipher_blocs[1:]))
    result = decrypt(iv, ciphertext)['error']
    return 'œil' not in result


def detect_padding(cipher_blocs):
    for length in range(1, 16):
        cipher_blocs[-2][-length] ^= 1
        valid = valid_padding(cipher_blocs)
        cipher_blocs[-2][-length] ^= 1
        if valid:
            break
    return length - 1


def break_bloc(cipher_blocs, pad_length=0):
```

```
    result = []
    save = cipher_blocs[-2].copy()
    for length in range(pad_length, 16):
        for i in range(15, 15 - length, -1):
            cipher_blocs[-2][i] ^= length ^ (length + 1)
        for k in range(256):
            cipher_blocs[-2][-length-1] ^= k
            valid = valid_padding(cipher_blocs)
            if valid:
                result.append(k ^ (length+1))
                break
            cipher_blocs[-2][-length-1] ^= k
    cipher_blocs[-2] = save
    return result


iv = list(bytes.fromhex(PASSWORD['iv']))
ciphertext = bytes.fromhex(PASSWORD['ciphertext'])

cipher_blocs = [iv] + divide_blocks(ciphertext)
pad_length = detect_padding(cipher_blocs)

plain = break_bloc(cipher_blocs, pad_length)
cipher_blocs.pop()

while len(cipher_blocs) > 1:
    plain.extend(break_bloc(cipher_blocs))
    cipher_blocs.pop()

print(bytes(reversed(plain)))
```

Exercices

Baptiste Arnold, Oscar Chevalier, Augustin Glorian, Célian Raimbault, Quentin Rataud

2.1 Choix complexe

2.1.1 Enoncé

Augustin, Cyril, Julie, Oscar, Raphaël, et Valérian souhaitent organiser une soirée cinéma chez l'un d'entre eux. Mais pour ce faire, ils doivent commencer par choisir quel film ou série ils vont regarder.

Après quelques discussions, ils n'arrivèrent toujours pas à se mettre d'accord. C'est alors que Valérian propose que chaque personne écrive le titre du film ou de la série qu'il voudrait voir et un autre titre qu'il ne voudrait absolument pas regarder. Après quelques minutes de réflexion et après avoir récolté les deux titres des six amis, les films et les séries ont été regroupés dans une liste 'adore' et une liste 'déteste'.

C'est ici que Valérian vous demande votre aide afin de les aider dans leur choix cornélien.

Vous devez, à partir de deux listes **adore** et **déteste**, afficher le nombre de films et de séries que les six jeunes peuvent regarder.

On considère que les jeunes peuvent regarder un film si et seulement si au moins une personne adore le film, et que personne ne le deteste.

Ils comptent sur vous !

2.1.2 Stratégie

La difficulté principale dans ce problème réside dans la possibilité de l'apparitions de doublons dans les films adorés ou détestés. En effet, nos amis pouvant avoir des goûts très similaires, il n'est pas rare qu'une série ou un film apparaisse plusieurs fois dans la liste **adore** ou la liste **deteste**. Étant donné que notre objectif est de déterminer le nombre de films apparaissant au moins une fois dans **adore** mais pas dans **deteste**, la première étape peut être de retirer les doublons des deux listes.

Une structure de données utile pour résoudre ce problème est celui des *ensembles*[14], qui ne conserve qu'une unique copie de chaque élément. Nous pouvons ensuite facilement comparer l'ensemble des films adorés à l'ensemble des films détestés.

2.1.3 Implémentation

Python3

```python
from typing import List


def nombre_films(adore: List[str], deteste: List[str]) -> None:
    """
    :param adore: liste des noms du film adoré de chaque personne
    :param deteste: liste des noms du film détesté de chaque personne
    """
    # TODO Afficher, sur une ligne, le nombre de films qui sont uniquement
```

14. https://fr.wikipedia.org/wiki/Ensemble_(informatique)

```
    # adorés.

    # On construit les ensembles des films adorés ou détestés
    adore = set(adore)
    deteste = set(deteste)

    # On considère qu'un film est visionnable s'il appartient aux films
    # adorés, mais pas aux films détestés
    visionnable = adore - deteste

    print(len(visionnable))


if __name__ == "__main__":
    adore = [input() for _ in range(6)]
    deteste = [input() for _ in range(6)]
    nombre_films(adore, deteste)
```

C++

```
#include <iostream>
#include <istream>
#include <string>
#include <vector>
#include <bits/stdc++.h>

/// \param adore liste des noms du film adoré de chaque personne
/// \param deteste liste des noms du film détesté de chaque personne
void nombre_films(const std::vector<std::string>& adore, const std::vector<std::string>& deteste) {
    using namespace std;

    // On construit les ensembles des films adorés ou détestés
    set<string> ensemble_adore(adore.begin(), adore.end());
    set<string> ensemble_deteste(deteste.begin(), deteste.end());

    int compte = 0;
    for (string film : ensemble_adore) {
        if (ensemble_deteste.count(film) == 0) {
            compte++;
        }
    }

    cout << compte << endl;
}

int main() {
    std::vector<std::string> adore(6); ///< liste des noms du film adoré de chaque personne
    for (std::string& i : adore) {
        std::getline(std::cin >> std::ws, i);
    }
    std::vector<std::string> deteste(6); ///< liste des noms du film détesté de chaque personne
    for (std::string& i : deteste) {
        std::getline(std::cin >> std::ws, i);
    }
    nombre_films(adore, deteste);
}
```

2.2 Mise en boîte

2.2.1 Énonce

Nos 6 amis sont toujours en train de choisir leurs film quand Cyril se rend compte que les restes du repas qu'ils ont dégusté juste avant attendent encore sur la table basse. Pour conserver leurs aliments, il faut les mettre dans des boîtes et les ranger au réfrigérateur. Qui sait, ils pourraient en avoir besoin si d'aventure ils décidaient de se lancer dans un grand voyage ! Le hasard faisant bien les choses, ils ont autant de boîtes que de restes. Tous les récipients ne sont toutefois pas de la bonne taille.

Aidez nos 6 amis à remplir le plus de boîtes sachant qu'un aliment d'un certain volume ne peut entrer que dans une boîte d'un volume supérieur ou égal. Afin de concerver au mieux le goût de la nourriture, on ne mettra qu'un aliment dans chaque boîte.

2.2.2 Stratégie

Si un reste peut loger dans une boîte, alors ce même reste peut loger dans toutes les boîtes de volume supérieur. Afin de ne pas occuper une trop grande boîte inutilement, on préfère donc toujours utiliser la boîte la plus petite possible pouvant contenir chaque aliment. Pour ce faire, il peut être intéressant de *trier* la liste des volumes des boîtes et des restes !

En ayant la liste des boîtes triées, nous pouvons simplement tenter de loger tour à tour chacun de nos aliments dans la première boîte possible. Grâce au tri au préalable des deux listes, cela peut se faire en un unique parcours.

Cette stratégie peut s'apparenter à la stratégie des "deux pointeurs". L'idéal est de simplement utiliser deux variables pour itérer sur nos deux listes sans les modifier, qui peut s'avérer coûteux. On rappelle que la complexité temporelle de la suppression d'un élément d'une liste est en général linéaire !

2.2.3 Implémentation

```python
from typing import List


def mise_en_boite(n: int, restes: List[int], boites: List[int]) -> None:
    """
    :param n: le nombre de boites et de restes
    :param restes: Liste des volumes des restes
    :param boites: Liste des volumes des boites
    """
    restes.sort()
    boites.sort()

    r = 0
    for boite in boites:
        # Si on peut loger le reste dans la plus petite boîte, on le fait
        if restes[r] <= boite:
            r += 1

        # Si l'on n'a pas pu loger le plus petit reste dans cette boîte,
        # alors aucun reste ne pourra loger dans celle-ci.
        # On peut donc ignorer cette boîte et simplement passer à la boîte
        # suivante

    print(r)


if __name__ == "__main__":
    n = int(input())
    restes = list(map(int, input().split()))
    boites = list(map(int, input().split()))
    mise_en_boite(n, restes, boites)
```

2.3 Pas malin-dromes

2.3.1 Enoncé

Les amis, ayant enfin décidé quel film regarder, décident de se connecter sur Netflux pour le visionner. Malheureusement, Raphaël a oublié le mot de passe de son compte Netflux ! Il a besoin de vous pour tenter de le retrouver.

Étant tête en l'air Raphaël oublie souvent ses mots de passe. Il a donc un fichier comportant une multitude de mots de passe différents, où se trouve d'ailleurs aussi le mot de passe de son compte Netflux. Il y a une chose dont il se rappelle concernant celui-ci, c'était que son mot de passe était un pas malin-drome. Aidez Raphaël à déterminer le nombre de pas malin-dromes présents dans le fichier.

Un pas malin-drome est une variante du palindrome. C'est un mot qui, si on en extrait uniquement les chiffres, uniquement les minuscules **ou** uniquement les majuscules, forme, à partir des caractères extraits, un palindrome. Un pas malin-drome est donc un mot où les lettres minuscules forment un palindrome, les lettres majuscules forment un palindrome et les chiffres forment un palindrome. Les autres caractères peuvent être présents dans un mot de passe mais seront ignorés dans la recherche de pas malin-drome.

Ils comptent sur vous !

2.3.2 Stratégie

Différentes stratégies sont envisageables pour cet exercice. La stratégie la plus courante a été d'extraire les chiffres, les minuscules, ainsi que les chiffres dans trois chaînes de caractères pour ensuite vérifier que ces trois chaînes de caractères soient des palindromes. Cela peut se faire à l'aide d'expressions régulières, où en parcourant chaque mot de passe caractère par caractère pour décider de l'ajouter ou non dans chacune des trois chaînes de caractères

La seconde stratégie, plus efficace en mémoire, se base sur l'utilisation de pointeurs. Pour vérifier si les chiffres d'un mot de passe forment un palindrome par exemple, il est en effet possible de placer un pointeur à chaque extrémité du mot de passe, et de les faire se rapprocher en ignorant tous les caractères n'étant pas des chiffres. Lorsque les deux pointeurs indiquent un chiffre, on vérifie que les deux chiffres soient identique.

L'implémentation dans la section 2.3.3 utilise la première stratégie.

2.3.3 Implémentation

Python3

```python
from typing import List

def est_palindrome(mot: str) -> bool:
    """
    Retourne vrai si le mot est un palindrome
    """
    return mot == mot[::-1]

def est_pas_malin_drome(mot: str) -> bool:
    """
    :param mot: le mot de passe
    Retourne vrai si le mot de passe est un pas malin-drome.
    """

    minuscules = ""
    majuscules = ""
    chiffres = ""

    for char in mot:
        if char.islower():
            minuscules += char
        elif char.isupper():
            majuscules += char
        elif char.isdigit():
            chiffres += char

    return (
        est_palindrome(minuscules) and
        est_palindrome(majuscules) and
```

```
        est_palindrome(chiffres)
    )

def nb_pas_malin_drome(n: int, mots: List[str]) -> None:
    """
    :param n: le nombre de mots de passe contenus dans le fichier de mots de passe de Raphael
    :param mots: la liste des mots de passe à decoder
    """
    # TODO Afficher le nombre de pas malin-dromes situés dans le fichier de mots
    # de passe Raphael

    compteur = 0

    for mot in mots:
        if est_pas_malin_drome(mot):
            compteur += 1

    print(compteur)


if __name__ == "__main__":
    n = int(input())
    mots = [input() for _ in range(n)]
    nb_pas_malin_drome(n, mots)
```

2.4 Grève Générale

2.4.1 Énoncé

Malheureusment, Raphaël bloque son compte Netflux après trois tentatives échouées de connexion. Les 6 jeunes tentent alors d'aller directement voir le film au cinéma.

Cependant, ils ne savent pas qu'aujourd'hui, c'est la grève nationale de la ***S****ociété* ***N****ationale des* ***C****inémas* ***F****rançais ! Tous les cinémas sont donc fermés, mais chaque cinéma indique la position d'une autre salle de cinéma où aller en attendant la réouverture, qui est fermée à son tour...*

Julie dirige la marche et doit prévoir les potentiels trajets.

N cinémas sont numérotés par un nombre entre 1 et N. Tous les cinémas sont fermés, et la valeur **redirection[i]** donne le numéro du cinéma indiqué par le cinéma i.

Les jeunes commencent par aller voir le cinéma S. Voyant le cinéma fermé, ils vont donc vérifier le cinéma **redirection[S]**, avant de constater que celui-ci est fermé aussi, et ainsi de suite.

Au bout d'un certain nombre de redirections, il est démontrable que les jeunes vont finir par retomber une 2^e fois sur une salle de cinéma déjà visitée auparavant. À ce moment-là, ils s'arrêtent de parcourir les salles de cinéma, car ils savent qu'ils ne vont jamais en finir.

Aidez Julie à prédir, pour chaque valeur de S (entre 1 et N) combien de salles de cinéma le groupe aurait visité ?

2.4.2 Stratégie

Simuler le trajet des jeunes pour chaque valeur de S peut prendre au pire des cas un temps linéaire, pour chaque point de départ. Cela implique une complexité temporelle pire cas quadratique pour trouver toutes les valeurs, ce qui ne permet pas de passer les tests de performances. En effet, si nous imaginons le cas d'un grand cycle de taille N, alors nous parcourerions chacun des N cinémas pour chaque cinéma de départ. En revanche, au vu des contraintes de validation, cela est suffisant pour passer les tests de validations. En effet, N valant au plus 1,000, il est raisonnable de penser que nous pouvons simuler un million de redirections en moins d'une seconde sur un ordinateur moderne. Le premier code de la section 2.4.3 implémente cette simulation.

Pour passer les tests de performances, il va falloir éviter beaucoup de calculs redondants. En effet, on se rend compte que dans le cas d'un grand cycle, on peut se retrouver à reparcourir en entier plusieurs milliers de fois le même cycle ! Une observation clé pour résoudre ce problème est que nous pouvons représenter les cinémas grâce à un *graphe fonctionnel*[15], où les cinémas sont les sommets, et les redirections sont nos arêtes. En faisant ainsi, on se rend compte que le graphe est formé de cycles, et de chemins menant à un cycle. On remarque ici que la réponse pour tous les cinémas présents dans un cycle est toujours la taille du cycle, et que la réponse pour tous les cinémas présents dans un chemin menant à un cycle est de 1 supérieur à la réponse de son successeur.

Alors, nous pouvons résoudre ce problème efficacement en détectant les cycles, en calculant leur taille, et en parcourant les chemins menant à un cycle à l'envers. La seconde implémentation de la section 2.4.3 emploie cette stratégie.

Pour plus de challenge, sachez qu'il est possible de passer les tests de performance de ce problème en utilisant une quantité constante de mémoire additionnelle ! Cela peut être réalisé en faisant usage de l'*Algorithme du lièvre et de la tortue.*[16]

2.4.3 Implémentation

Ce premier code permet de passer les tests de validations. Sa complexité temporelle pire cas est de $O(N^2)$.

```python
from typing import List


def trajets_retour(n: int, redirection: List[int]) -> None:
    """
    :param n: le nombre de cinémas
    :param redirection: le lieu de redirection de chaque cinéma
    """

    reponse = []

    for cinema in range(n):
        redirections = 0
        visite = set()
```

15. En théorie des graphes, un graphe fonctionnel est un graphe où tous les demi-degrés extérieurs des sommets sont égaux à 1
16. https://fr.wikipedia.org/wiki/Algorithme_du_li%C3%A8vre_et_de_la_tortue

```
        while cinema not in visite:
            visite.add(cinema)
            cinema = redirection[cinema] - 1
            redirections += 1

        reponse.append(redirections)

    print(*reponse)


if __name__ == "__main__":
    n = int(input())
    redirection = list(map(int, input().split()))
    trajets_retour(n, redirection)
```

Ce second code permet de passer les tests de performance. Sa complexité temporelle pire cas est de $O(N)$.

```
from typing import List


def trajets_retour(n: int, redirection: List[int]) -> None:
    """
    :param n: le nombre de cinémas
    :param redirection: le lieu de redirection de chaque cinéma
    """

    reponse = [None] * n

    for cinema in range(n):

        # On connait déjà la réponse
        if reponse[cinema] is not None:
            continue

        actuel = cinema
        chemin = []
        visite = set()
        while actuel not in visite and reponse[actuel] is None:
            visite.add(actuel)
            chemin.append(actuel)
            actuel = redirection[actuel] - 1

        if reponse[actuel] is None:
            # Nous venons de découvrir un cycle pour la première fois,
            # formé par les taille_cycle derniers éléments de chemin.
            taille_cycle = len(chemin) - chemin.index(actuel)
            for actuel in chemin[-taille_cycle:]:
                reponse[actuel] = taille_cycle

            chemin = chemin[:-taille_cycle]

        # Les cinémas menant au cycle entraînent toujours une redirection de
        # plus que leur successeur.
        for precedent in reversed(chemin):
            reponse[precedent] = reponse[actuel] + 1
            actuel = precedent

    print(*reponse)



if __name__ == "__main__":
```

```
n = int(input())
redirection = list(map(int, input().split()))
trajets_retour(n, redirection)
```

2.5 Stabilisateurs

2.5.1 Énoncé

Impossible de se connecter à Netflux et tous les cinémas sont fermés... Mais finalement, rien ne remplace l'expérience de l'univers du film. Vivre aux époques passées ou futures de ces productions donne envie à nos jeunes ingénieurs en herbe. Ils décident donc, tout naturellement, de construire une machine à voyager à travers les univers.

Oscar se charge de s'assurer de la stabilité inter-temporelle de la machine. Qui sait dans quel univers nos amis pourraient atterir si la machine partait en biberine !

Afin de stabiliser la machine, il dispose de K stabilisateurs spatio-temporels, ainsi que de N accroches. Pour mettre en place un stabilisateur, il faut l'attacher à précisément 4 accroches. De plus, une accroche ne peut servir que pour un seul stabilisateur.

Si les 4 accroches sont toutes situées exactement à la même hauteur, le stabilisateur possède un indice de stabilité égal à P. Cependant, si les 4 accroches ne sont pas toutes situées exactement à la même hauteur, cela peut entraîner un déséquilibre.

On appelle le "déséquilibre" d'un stabilisateur la différence entre la plus haute et la plus basse de ses accroches, au carré. Par exemple, si un stabilisateur se repose sur 4 accroches de hauteur 6, 5, 6, et 7, le stabilisateur possède un déséquilibre de $(7-5)^2 = 4$.

Si un stabilisateur possède un déséquilibre, alors son indice de stabilité vaut P moins son déséquilibre.

L'indice de stabilité de la machine après avoir positionné un certain nombre de stabilisateurs est la somme des indices de stabilité de tous les stabilisateurs positionnés. **Vous n'êtes pas obligés de positionner tous les stabilisateurs**.

Aidez Oscar à trouver l'indice de stabilité maximal qu'il peut obtenir pour la machine !

2.5.2 Stratégie

Lemme Il est toujours préférable de choisir, pour chaque stabilisateur positionné, quatre accroches adjacentes dans la liste des hauteurs triées.

Preuve Supposons la liste des accroches triée. Supposons que nous ayons une solution telle que les stabilisateurs positionnés n'utilisent pas chacune quatre accroches adjacentes dans la liste. C'est-à-dire qu'il existera au moins un stabilisateur A qui utilisera quatre accroches non adjacentes.

Prenons une accroche c non sélectionnée par ce stabilisateur, située dans la liste entre l'accroche la plus basse et la plus haute du stabilisateur A. Si cette accroche n'est pas utilisée par un autre stabilisateur, alors remplacer la plus basse ou la plus haute accroche du stabilisateur par celle-ci augmentera l'indice de stabilité total.

Si cette accroche est utilisée par un autre stabilisateur B, alors :

- Si toutes les autres accroches du stabilisateur B sont situées plus haut que les accroches du stabilisateur A, alors échanger l'accroche la plus haute de A et c augmentera les indices de stabilité de A et B.
- Si toutes les autres accroches du stabilisateur B sont situées plus bas que les accroches du stabilisateur A, alors échanger l'accroche la plus basse de A et c augmentera les indices de stabilité de A et B.
- Sinon, échanger la plus haute ou la plus basse accroche de A avec c augmentera l'indice de stabilité de A sans modifier l'indice de stabilité de B.

Ainsi, toute solution ayant au moins un stabilisateur n'utilisant pas quatre accroches adjacentes dans la liste des accroches n'est pas optimale.

La première étape va donc être de trier notre liste d'accroches. Notre objectif va donc être de choisir au plus K groupes distincts de 4 accroches adjacentes dans cette liste d'accroches triées, afin de maximiser l'indice total de stabilité.

Itérer toutes les possibilités de groupe est possible en une complexité temporelle de $O(N^K)$. Cela est suffisant pour passer les tests de validation, mais très loin de pouvoir passer les tests de performance.

Afin de réduire drastiquement cette complexité, il est possible de faire un rapprochement avec le *problème du sac à dos*.[17] En effet, si nous voulons trouver une solution optimale en plaçant au plus K stabilisateurs en ne se servant que des N dernières accroches, nous n'avons que deux possibilités :

- Utiliser les 4 premières accroches possibles, et trouver une solution optimale en plaçant au plus $K-1$ stabilisateurs avec les $N-4$ dernières accroches.
- Ne pas utiliser la première accroche, et trouver une solution optimale en plaçant au plus K stabilisateurs avec les $N-1$ dernières accroches.

En utilisant le principe de la *programmation dynamique*, nous voyons que si nous enregistrons ou mettons en cache toutes les valeurs de retour possibles de la fonction en fonction de ses paramètres, il n'y a que $N \times K$ paramètres possibles. La réponse à notre problème est donc la valeur de retour avec N et K en paramètres, et la complexité temporelle totale est donc de $O(NK)$, suffisant pour passer les tests de performance.

17. https://fr.wikipedia.org/wiki/Probl%C3%A8me_du_sac_%C3%A0_dos

L'implémentation ci-contre possède une complexité de mémoire de $O(NK)$ également, mais en codant itérativement, nous pouvons réduire la complexité de mémoire à $O(N)$.

2.5.3 Implémentation

```python
from typing import List
from functools import lru_cache
from sys import setrecursionlimit
setrecursionlimit(10 ** 9)


def stabilite_maximale(n: int, k: int, p: int, accroches: List[int]) -> None:
    """
    :param n: nombre d'accroches
    :param k: nombre de stabilisateurs
    :param p: indice de stabilité parfaite
    :param accroches: hauteur de chaque accroche
    """
    # TODO Afficher l'indice de stabilité maximal obtenable.
    accroches.sort()
    gains = []
    for i in range(n-3):
        gains.append(p - (accroches[i+3] - accroches[i]) ** 2)

    @lru_cache(None)
    def knapsack(i: int, k: int) -> int:
        """
        Algorithme du sac à dos.
        Détermine le gain maximal, en positionnant au maximum k stabilisateurs,
        et sans utiliser les i premières accroches (triées par hauteur).
        :param i: position actuelle dans la liste des accroches
        :param k: nombre de stabilisateurs restants
        """
        if k <= 0:
            return 0
        if i >= n-3:
            return 0

        prend = gains[i] + knapsack(i+4, k-1)
        passe = knapsack(i+1, k)
        return max(prend, passe)

    print(knapsack(0, k))

if __name__ == "__main__":
    n = int(input())
    k = int(input())
    p = int(input())
    accroches = list(map(int, input().split()))
    stabilite_maximale(n, k, p, accroches)
```

2.6 Refroidissement

2.6.1 Énoncé

La machine à voyager de nos chers amis peut présenter des pannes en cas de surchauffe. Pour éviter ce genre de pannes, il faudrait mettre en place un système de refroidissement.

La machine contient N points pouvant être reliés par des tuyaux. Chaque tuyau refroidit le liquide y passant d'un certain nombre de degrés. Les tuyaux ne sont pas bidirectionnels.

Votre but est de savoir combien de tuyaux au minimum composent un chemin refroidissant d'au moins K degrés le liquide. Ce chemin commence par le point A et se termine au point B. Pour cela, vous disposerez du plan de la machine comprenant les points et les emplacements possibles des tuyaux.

Dans ce plan, s'il existe un emplacement de tuyau entre les points U et V, alors vous pouvez construire autant de tuyaux entre ces points (possiblement aucun tuyau). Pour des raisons physiques, votre chemin ne peut pas emprunter plusieurs fois le même tuyau. Cependant, vous pouvez contourner cette restriction en construisant plusieurs tuyaux sur un même emplacement, cela compte alors comme plusieurs poses de tuyaux. De même, le plan peut contenir des emplacements reliant le même point ($U = V$).

2.6.2 Stratégie

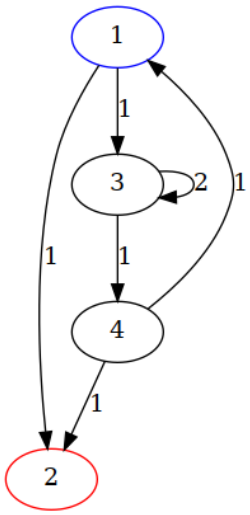


FIGURE 2.5 – Exemple de graphe

Voici comment résoudre ce problème en temps $O(N^3 \log^2(NK))$.
Prenons pour exemple le graphe en figure 2.5.
La point A (le départ) est en bleu et le point B (la cible) est en rouge.

Reformulation du problème Le but est de trouver le chemin le plus court (nombre d'arêtes minimal) ayant un poids total (somme des poids des arêtes) $\geq K$.

Intuition Faisons un tableau tel que $\text{dp}_{t,i,j}$ soit égal au poids maximal d'un chemin de longueur t du nœud i au nœud j. Avec ceci, nous pouvons trouver à partir de quel t une valeur est plus grande que K.

Programmation Dynamique $\text{dp}_{1,i,j} = \text{adj}_{i,j}$ avec `adj` la matrice d'adjacence du graphe. Ensuite on a $\text{dp}_{t+1,i,j} = \max_k(\text{dp}_{t,i,k} + \text{dp}_{1,k,j})$, nous pouvons voir cela comme la valeur maximale des poids de chaque chemin allant de i jusqu'à k de longueur t avec une arête de k à j.

Nous remarquons que cette méthode est similaire à une multiplication de matrice. Il est donc possible d'optimiser en utilisant l'*exponentiation rapide* pour avoir une complexité de $O(N^3 \log P)$ avec P l'exposant (au lieu de $O(N^3 \times P)$).

Recherche dichotomique Pour résoudre ce problème de manière optimisée, nous pouvons combiner exponentiation rapide et recherche dichotomique.

Si nous créons un nœud supplémentaire comme le nœud jaune dans la figure 2.6, relié à lui-même ainsi qu'au nœud A de départ avec un poids de 0, il nous est utile pour accumuler les poids maximaux. C'est-à-dire, au lieu de calculer le poids maximal d'un chemin de t nœuds, nous pouvons ainsi calculer le poids maximal d'un chemin de t nœud **ou moins**.

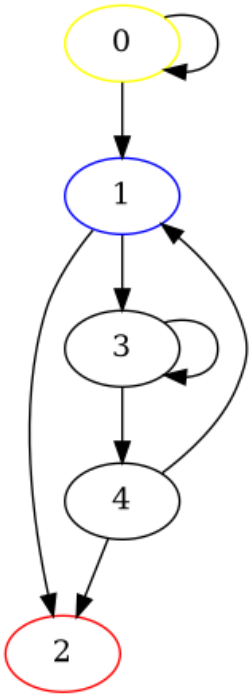


FIGURE 2.6 – Ajout du nœud supplémentaire (en jaune)

Si un tel chemin existe, alors il possède une taille comprise entre 0 et $(N+1) \times K$ nœuds.

Soit $p(t)$ le prédicat «Il est possible de trouver un chemin d'au plus t arêtes tel que le poids total est $\geq K$.» Le prédicat p permet de décider comment réduire l'intervalle de la recherche dichotomique. Ce prédicat peut être vérifié en $O(N^3 \log t)$, grâce à la méthode exprimée dans le paragraphe **Programmation Dynamique**.

Au total, la complexité temporelle est de $O(N^3 \log^2(NK))$.

2.6.3 Implémentation

```cpp
#include <bits/stdc++.h>

using namespace std;

using ll = long long;
using mat = vector<vector<ll>>;

constexpr ll NEUTRAL = INT64_MIN / 3;

// res[i, j] = Max_k(a[i, k] + b[k, j])
// Similaire à une multiplication de matrice
// a et b sont de forme [n, n]
mat cumul(const mat &a, const mat &b) {
    auto n = a.size();
    mat res(n, vector<ll>(n, NEUTRAL));
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            ll item = NEUTRAL;

            for (int k = 0; k < n; ++k) {
                item = max<ll>(item, a[i][k] + b[k][j]);
            }

            // Le laisser neutre
            if (item < 0) {
                item = NEUTRAL;
            }
```

```
            res[i][j] = item;
        }
    }

    return res;
}

// Exponentiation rapide avec la fonction cumul
mat cumul_exp(const mat &graph, int len) {
    if (len > 1) {
        auto m2 = cumul(graph, graph);
        auto res = cumul_exp(m2, len / 2);
        if (len % 2 == 1) {
            res = cumul(res, graph);
        }

        return res;
    }

    return graph;
}

// Chemin de poids max entre A et B avec une taille maximum len
ll path_weight(const mat &graph, int a, int b, int len) {
    if (len == 1) {
        return graph[a][b];
    }

    return cumul_exp(graph, len)[a][b];
}

int shortest_path(const mat &graph_default, int a, int b, int min_weight) {
    // Ajout noeud supplémentaire
    int n = graph_default.size() + 1;
    mat graph = graph_default;
    for (auto &row : graph) {
        row.push_back(NEUTRAL);
    }

    graph.push_back(vector<ll>(n, NEUTRAL));

    graph[n - 1][a] = 0;
    graph[n - 1][n - 1] = 0;

    // Cas impossible (pas de cycles entre A et B et pas assez de poids)
    auto weight_n = path_weight(graph, n - 1, b, n + 1);
    if (weight_n < min_weight &&
            weight_n == path_weight(graph, n - 1, b, 2 * (n + 1))) {
        return -1;
    }

    // Binary search
    int l = 1;
    int r = (n + 1) * min_weight + 1;
    while (l < r) {
        int mid = l + (r - l) / 2;
        ll p = path_weight(graph, n - 1, b, mid);
        if (p < min_weight) {
            l = mid + 1;
        } else {
```

```
            r = mid;
        }
    }

    // Le noeud supplémentaire (départ) a une arête jusqu'à A, il faut l'enlever
    return l - 1;
}

int main() {
    long long N, M, K, A, B;
    cin >> N >> M >> K >> A >> B;
    A--; B--;
    mat graph(N, vector<ll>(N, NEUTRAL));
    for(int i = 0; i < M; i++) {
        long long U, V, W;
        cin >> U >> V >> W;
        U--; V--;
        graph[U][V] = max(graph[U][V], W);
    }

    auto res = shortest_path(graph, A, B, K);
    cout << res << endl;

    return 0;
}
```

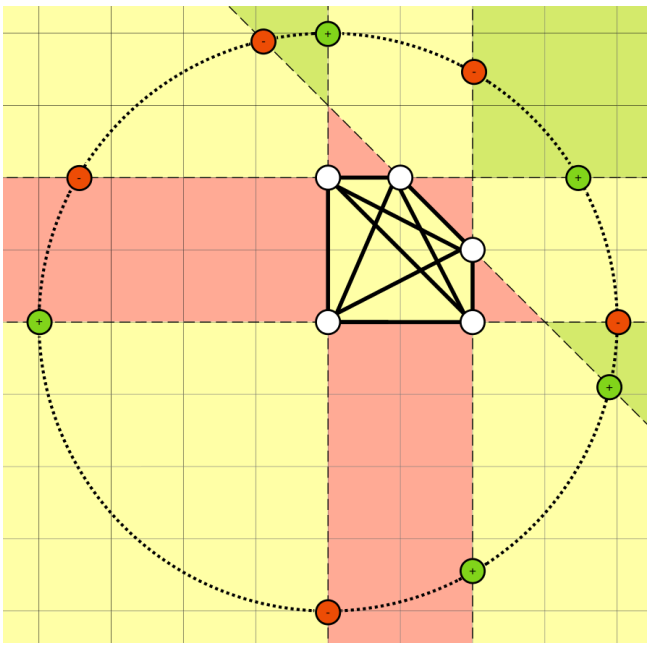


FIGURE 2.7 – Ordre des enveloppes convexes induits par ajout d'un point, pour chaque point du plan

2.7 Extension Stratégique

2.7.1 Énoncé

La machine est dorénavant prête au décollage ! Mais avant cela, il faut planifier le voyage, dans la plus grande des précautions.

Pour pouvoir se déplacer efficacement, il existe des points de contrôle permettant un mouvement d'une vitesse proche de la vitesse de la lumière ! Mais ces déplacements pouvant être dangereux, une zone de sécurité a été construite, contenant tous les tunnels entre les points de contrôle.

Pour contribuer à la recherche spatio-temporelle, Augustin planifie la construction d'un nouveau point de contrôle...

Il existe aujourd'hui $N+1$ points de contrôle, dont leur position est définie par deux coordonnées dans le plan spatio-temporel. Votre machine se situe au premier point de contrôle, de coordonnées $(0, 0)$. Vous connaissez la position des N autres points de contrôle.

Chaque paire de points de contrôle est reliée par un tunnel en ligne droite dans le plan spatio-temporel. Afin de s'assurer de la sécurité des déplacements entre les points de contrôle, il existe une zone de sécurité, représentée dans le plan spatio-temporel par un polygone d'aire minimale, contenant la totalité des tunnels.

Augustin planifie la construction d'un nouveau point de contrôle dans le plan spatio-temporel. La zone de sécurité sera adaptée pour y intégrer le nouveau point de contrôle, ainsi que tous les tunnels en direction de votre point de contrôle, tout en restant un polygone et en gardant une aire minimale.

Attention cependant, vous ne disposez d'aucune information concernant ce qui se passe au delà d'une distance **D** de votre machine temporelle. Au delà de cette distance, c'est un grand flou que personne n'a encore jamais exploré ! Autrement dit, l'espace-temps connu forme un cercle de rayon **D** centré sur votre machine dans le plan spatio-temporel. Il est hors de question de construire un point de contrôle en dehors de cette zone ! Tous les points de contrôle actuels sont d'ailleurs situés dans l'espace-temps connu.

Aidez Augustin à déterminer quel serait le nombre minimal d'arêtes de la zone de sécurité après la construction de votre point de contrôle.

2.7.2 Observations

La zone de sécurité peut être vue comme l'intérieur de l'*enveloppe convexe* formée par les points de contrôle. L'objectif du problème est donc de minimiser le nombre d'arêtes de l'enveloppe convexe, en ajoutant un unique point dans le cercle de l'espace-temps connu.

Si nous prenons le 2e exemple de l'énoncé, la Figure 2.7 représente le nombre d'arêtes de l'enveloppe convexe après ajout d'un point, pour tous les points du plan.

- Si nous ajoutons un point dans une zone en vert, le nombre d'arêtes de l'enveloppe convexe est 4.
- Si nous ajoutons un point dans une zone en jaune, le nombre d'arêtes de l'enveloppe convexe est 5.
- Si nous ajoutons un point dans une zone en rouge, le nombre d'arêtes de l'enveloppe convexe est 6.

On se rend compte ici que le résultat est toujours intimement lié aux droites correspondant aux prolongations des segments de l'enveloppe convexe.

Lemme Une solution optimale existera toujours en plaçant un point directement sur le cercle.

Preuve En effet, nous pouvons remarquer que, si nous partons de n'importe quel sommet de l'enveloppe convexe originale et que nous nous dirigeons vers l'extérieur du polygone, la solution ne peut que s'améliorer. Ainsi, si nous prenons n'importe quel point dans l'espace-temps connu, et que nous nous éloignons du polygone (donc nous rapprochons du cercle) en suivant une demi-droite partant d'un sommet du polygone et n'intersectant pas le polygone, la solution ne peut que s'améliorer.

Enfin, si nous parcourons dans le sens trigonométrique les segments de l'enveloppe convexe originale, et que nous étendons le segment des deux côtés, par une demi-droite suivant le sens trigonométrique (-), et par une demi-droite dans le sens inverse (+), nous obtenons deux intersections avec le cercle. Appelons les intersections avec les demi-droites trigonométriques les intersections (-) et les autres intersections les intersections (+).

Nous pouvons remarquer que, en parcourant le cercle de l'espace-temps connu dans le sens trigonométrique, dès que nous rencontrons une intersection (-), la solution empire d'une arête, et quand nous rencontrons une intersection (+), la solution s'améliore d'une arête.

2.7.3 Stratégie

Nous allons d'abord trouver l'enveloppe convexe de tous les points de contrôle. Cela peut se faire en un temps de $O(n \log n)$ en utilisant un algorithme comme le parcours de Graham. Ensuite, nous pouvons parcourir l'enveloppe convexe et calculer les différents points d'intersection des droites et du cercle, en $O(h)$, h étant l'ordre de l'enveloppe convexe.

Enfin, nous sélectionnons un point du cercle, ajoutons un point de contrôle à cette position, et calculons le nombre d'arêtes de la nouvelle enveloppe convexe. Cela peut se faire en $O(h)$ en conservant l'ordre des sommets, avec un second parcours de Graham. Il suffit finalement de parcourir les $O(h)$ points d'intersection dans l'ordre du cercle pour à chaque fois mettre à jour l'ordre de l'enveloppe convexe.

La réponse est la valeur minimale obtenue pendant le parcours. La complexité temporelle finale est ainsi de $O(n \log n)$.

2.7.4 Implémentation

```cpp
#include <bits/stdc++.h>
using namespace std;

// geometry
typedef long double ftype;
typedef complex<ftype> point;
#define x real()
#define y imag()
#define EPSILON 1e-4

ftype dot(point a, point b) {
    return (conj(a) * b).x;
}

ftype cross(point a, point b) {
    return (conj(a) * b).y;
}

// Trie les points pour un parcours de Graham
// Complexité temporelle: O(n log n)
vector<point> graham_sort(vector<point>& checkpoints) {
    // Trouve le point avec la coordonnées y la plus basse
    point p0 = {0, 0};
    for (point p: checkpoints) {
        if (p.y < p0.y || (p.y == p0.y && p.x < p0.x))
            p0 = p;
    }

    // Trie tous les points selon leur angle avec p0
```

```
    sort(checkpoints.begin(), checkpoints.end(),
        [&p0] (const point& p1, const point& p2)
    {
        if (p1 == p0) return true;
        if (p2 == p0) return false;
        point v1 = {p1.x - p0.x, p1.y - p0.y};
        point v2 = {p2.x - p0.x, p2.y - p0.y};
        return cross(v1, v2) > 0;
    });

    // Supprime les doublons
    vector<point> result;
    result.push_back(p0);
    for (point p: checkpoints) {
        if (p == p0)
            continue;
        if (result.size() == 1) {
            result.push_back(p);
            continue;
        }
        point p1 = result.back();
        point v = {p.x - p0.x, p.y - p0.y};
        point v1 = {p1.x - p0.x, p1.y - p0.y};
        if (abs(cross(v, v1)) <= EPSILON) {
            if (abs(v.x) + abs(v.y) > abs(v1.x) + abs(v1.y))
                result.pop_back();
            else
                continue;
        }
        result.push_back(p);
    }

    result.push_back(p0);
    return result;
}

// Calcule l'enveloppe convexe
// Complexité temporelle: O(n)
vector<point> graham_scan(const vector<point> &checkpoints) {
    vector<point> hull;
    for (point p: checkpoints) {
        while (true) {
            if (hull.size() < 2) {
                hull.push_back(p);
                break;
            }
            point p0 = hull[hull.size()-2];
            point p1 = hull.back();

            point v0 = {p1.x - p0.x, p1.y - p0.y};
            point v1 = {p.x - p1.x, p.y - p1.y};

            ftype c = cross(v0, v1);
            if (c < -EPSILON || (abs(c) <= EPSILON && dot(v0, v1) > 0)) {
                hull.pop_back();
            }
            else {
                hull.push_back(p);
                break;
            }
        }
```

```
    }

    return hull;
}

// Intersection Cercle-Ligne
pair<point, point> intersection(point p0, point p1, ftype D) {

    // Equation de la ligne : ax + by + c = 0
    ftype a = p1.y - p0.y;
    ftype b = p0.x - p1.x;
    ftype c = - p0.x * a - p0.y * b;

    // Point le plus proche du centre du cercle
    ftype inv = 1/(a*a+b*b);
    point mid = {-a*c*inv, -b*c*inv};

    // Calcul des deux intersections
    ftype d = D*D - c*c*inv;
    ftype mult = sqrt(d*inv);
    point shift = (point){b, -a} * mult;
    point r0 = mid + shift;
    point r1 = mid - shift;

    return {r0, r1};
}

int main() {
    // Lecture de l'entrée
    ftype D; cin >> D; D += EPSILON;
    int N; cin >> N;
    vector<point> checkpoints(N+1);
    for (int i = 0; i < N; i++)
    {
        ftype px, py;
        cin >> px >> py;
        checkpoints[i] = {px, py};
    }
    checkpoints[N] = {0,0};

    // Premier calcul de l'enveloppe convexe
    vector<point> sorted = graham_sort(checkpoints);
    vector<point> hull = graham_scan(sorted);
    vector<point> left;
    vector<point> right;

    int h = hull.size()-1;
    int best = h;

    // Calcul des intersections entre les prolongations des arêtes de
    // l'enveloppe convexe et le cercle
    for (int i = 0; i < h; i++)
    {
        point p0 = hull[i];
        point p1 = hull[i+1];
        pair<point, point> intersections = intersection(p0, p1, D);
        left.push_back(intersections.first);
        right.push_back(intersections.second);
    }

    // Égalisation des deux pointeurs dans les deux listes
```

```
    vector<point>::iterator l = left.begin();
    vector<point>::iterator r = right.begin();

    int best_i = 0;
    ftype max_dot = LLONG_MIN;

    for (int i = 0; i < h; i++)
    {
        if (cross(*l, right[i]) < 0)
            continue;
        ftype val = dot(*l, right[i]);
        if (val > max_dot)
        {
            max_dot = val;
            best_i = i;
        }
    }

    r += best_i;

    // Ajout d'un point quelconque aux sommets et recalcul de l'enveloppe
    // convexe
    int actual = 0;
    point initial = *l;
    l++;
    if (l == left.end())
        l = left.begin();
    actual--;

    sorted.push_back(initial);
    sorted = graham_sort(sorted);
    hull = graham_scan(sorted);
    actual += hull.size();
    best = min(best, actual);

    // Parcours du cercle avec mise à jour instantanée de la taille de
    // l'enveloppe convexe
    for (int i = 0; i < 2*h-1; i++)
    {
        if (cross(*l, *r) > 0) {
            initial = *l;
            l++;
            actual--;
            if (l == left.end())
                l = left.begin();
            best = min(best, actual);
        }
        else {
            initial = *r;
            r++;
            actual++;
            if (r == right.end())
                r = right.begin();
        }
    }

    cout << best << endl;
}
```