

**RAPPORT DE PROJET C/C++**

-

**FIN DE PARCOURS**

# **Projet : Advent of Code 2016**

*Projet réalisé par :*

Clément SAGETTE  
Dan SIMON

*Projet encadré par :*

M. ZANNI  
M. MOREAU



# **SOMMAIRE**

## **I/ Introduction**

## **II/ Résolution Advent of Code 2016**

- i.** Jour 1
- ii.** Jour 2
- iii.** Jour 3
- iv.** Jour 4
- v.** Jour 5
- vi.** Jour 6
- vii.** Jour 7
- viii.** Jour 8
- ix.** Jour 9
- x.** Jour 10
- xi.** Jour 11
- xii.** Jour 12
- xiii.** Jour 13
- xiv.** Jour 14
- xv.** Jour 15
- xvi.** Jour 16
- xvii.** Jour 17
- xviii.** Jour 18

## **III/ Conclusion**

# I/ Introduction

Ayant commencé l'étude du langage C/C++ début octobre 2021 et souhaitant tous deux travailler dans le monde de la finance à la sortie de notre formation aux mines, nous avons à coeur de développer nos compétences sur ce langage de programmation. Ainsi, au lieu de réaliser un projet composé d'un seul objectif majeur, nous avons fait le choix de réaliser l'Advent of Code 2016 afin de se challenger sur plusieurs défis plus ou moins difficiles afin d'apprendre plus rapidement de nouvelles choses et en plus grande quantité.

Pour réaliser ce challenge, nous avons fait le choix de suivre la même démarche que le cours électif C/C++, à savoir commencer les défis en C lors de notre phase d'apprentissage du C, puis continuer en C++ lors de la phase d'apprentissage du C++.

La description de chaque journée se fera de la manière suivante :

1. Brève description du problème et de ses attentes
2. Explication globale du fonctionnement de l'algorithme et possibles astuces
3. Validation du résultat sur un exemple concret

L'intégralité des codes peut être retrouvée sur le GitHub suivant dans la branche main :

[https://github.com/csagette/advent\\_code\\_2016](https://github.com/csagette/advent_code_2016)

Remarque 1 : Chaque dossier de journée sur Github se composera du code principal intitulé **main**, d'un fichier secondaire **main2** si nécessaire pour la deuxième partie d'un défi, d'un fichier brouillon **test\_try** ainsi que des fichiers .h et .c/cpp supplémentaires utiles.

Remarque 2 : Il est possible que certains codes ne fonctionnent plus pour l'exemple de la première partie du défi d'une journée. En effet, certains codes ont été modifiés pour pouvoir résoudre la seconde partie du problème. Toutefois, il est bon à savoir qu'il est impossible d'obtenir la seconde partie du problème sans vérifier la première partie du problème donc notre code était bien fonctionnel avant modification.

Remarque 3 : Dans chacun des algorithmes, l'utilisateur souhaitant utiliser ses propres données d'entrée devra modifier les éléments suivant si existant:

- Le fichier input.txt. Attention à bien vérifier que ce dernier est dans un répertoire connu et ne pas oublier de modifier en début de chaque fonction **main** le lien vers ce fichier dans la fonction **fopen** en C et **file** en C++.
- Les données en début de chaque fonction **main** qui correspondront aux données d'entrée non sous forme de fichier input.txt

Remarque 4 : Pour ce challenge assez long pour des débutants en C/C++, nous avons découpé le travail en deux. Ainsi, les jours pairs ont été réalisés par Clément Sagette et les jours impairs par Dan Simon, ce qui peut également expliquer la différence de style dans certains problèmes assez similaires.

## II/ Résolution Advent of Code 2016

### i. Jour 1

Le problème du premier jour consiste dans un premier temps à trouver la distance de Manhattan entre notre position actuelle et la position du **Easter Bunny Headquarters**, position initialement inconnue et que l'on peut atteindre à partir d'une suite d'instructions de déplacement Est-Ouest et Nord-Sud en commençant à la position initiale nulle. Pour réaliser cet objectif, nous avons alors juste implémenté la distance de Manhattan qui permet d'obtenir la distance quasiment immédiatement en ayant seulement besoin du point de départ et d'arrivée.

La seconde partie du problème consistait à trouver cette fois-ci la première position où nous passons par celle-ci à deux reprises, qui en réalité est la véritable localisation du Headquarters. Pour ce faire, et étant encore débutant sur C lors de ce premier challenge, nous avons décidé d'utiliser une liste chaînée afin de garder en mémoire les positions déjà évaluées. Il suffit alors de vérifier pour chaque nouvelle position si cette dernière n'a pas déjà été visitée.

Remarque: Dans le code utilisé, et également souvent par la suite, nous avons utilisé le code ASCII des caractères pour identifier ce qui nous intéressait dans une chaîne de caractère. Nous avons par ailleurs appris dès ce premier jour à utiliser le type FILE et comment gérer l'ouverture, la lecture et l'écriture de fichier.

```

- R5, L5, R5, R3 leaves you 12 blocks away.

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
(x = 5, y = 2)
(x = 5, y = 3)
(x = 5, y = 4)
(x = 5, y = 5)
(x = 6, y = 5)
(x = 7, y = 5)
(x = 8, y = 5)
(x = 9, y = 5)
(x = 10, y = 5)
(x = 10, y = 4)
(x = 10, y = 3)
(x = 10, y = 2)
La distance est de 12.000000 blocs
(base) clement@MacBook-Pro-de-Clement Clément %

```

```

For example, if your instructions are R8, R4, R4, R8, the first location
you visit twice is 4 blocks away, due East.

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
cd
(base) clement@MacBook-Pro-de-Clement Day 1 % cd Clément
(base) clement@MacBook-Pro-de-Clement Clément % gcc -Wall -o main main.c list.c
(base) clement@MacBook-Pro-de-Clement Clément % ./main
Le point (x,y) = (4.000000,0.000000) a déjà été visité
La distance est de 4.000000 blocs
(base) clement@MacBook-Pro-de-Clement Clément %

```

## ii. Jour 2

Le problème du second jour consiste, une fois arrivé au **Easter Bunny Headquarters**, à trouver le code de la salle de bain de la chambre car ces dernières sont sécurisées. Nous avons alors accès à un petit boîtier de taille 3x3 composé des entiers de 1 à 9 ainsi que d'une liste de lignes d'instructions de déplacement. La position initiale d'où il faut partir est le centre du boîtier. Par exemple, l'instruction ULL signifie qu'il faut réaliser un mouvement "Up" puis deux mouvements "Left", ce qui impose d'arriver sur le chiffre 1 du boîtier comme le second mouvement à gauche à partir de la case 1 n'a aucun effet. Le code est alors la suite de nombre obtenue à partir des instructions de chaque ligne. Pour réaliser cette tâche, il suffit alors de lire les lignes du fichier une à une puis de vérifier où est ce qu'on l'atterrît après lecture des caractères, ou instructions dans notre contexte, de la ligne. On enregistre dans notre cas le code au fur et à mesure dans une liste.

La seconde partie du problème nous informe qu'en réalité, le boîtier n'est pas de forme carrée mais est le résultat d'un design innovant comprenant des lettres et des chiffres. Il nous faut alors retravailler les contraintes de déplacement et ajouter la possibilité d'obtenir des lettres pour obtenir le même résultat.

```
ULL
RRDD
LURDL
UUUUD
```

- You start at "5" and move up (to "2"), left (to "1"), and left (you can't, and stay on "1"), so the first button is **1**.
- Starting from the previous button ("1"), you move right twice (to "3") and then down three times (stopping at "9" after two moves and ignoring the third), ending up with **9**.
- Continuing from "9", you move left, up, right, down, and left, ending with **8**.
- Finally, you move up four times (stopping at "2"), then down once, ending with **5**.

So, in this example, the bathroom code is **1985**.

PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE

```
(base) clement@MacBook-Pro-de-Clement Day 2 % cd Clément
(base) clement@MacBook-Pro-de-Clement Clément % gcc -Wall -o main main.c list.c
(base) clement@MacBook-Pro-de-Clement Clément % ./main
1 9 8 5 %
```

1
2 3 4
5 6 7 8 9
A B C
D

You still start at "5" and stop when you're at an edge, but given the same instructions as above, the outcome is very different:

- You start at "5" and don't move at all (up and left are both edges), ending at **5**.
- Continuing from "5", you move right twice and down three times (through "6", "7", "B", "D", "D"), ending at **D**.
- Then, from "D", you move five more times (through "D", "B", "C", "C", "B"), ending at **B**.
- Finally, after five more moves, you end at **3**.

So, given the actual keypad layout, the code would be **5DB3**.

PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE

```
(base) clement@MacBook-Pro-de-Clement Day 2 % cd Clément
(base) clement@MacBook-Pro-de-Clement Clément % gcc -Wall -o main2 main2.c list.c
(base) clement@MacBook-Pro-de-Clement Clément % ./main2
5
D
B
3
(base) clement@MacBook-Pro-de-Clement Clément %
```

**iii. Jour 3**

Le problème du troisième jour consiste à déterminer le nombre de triangles possibles donnés par l'input. L'input est constituée de lignes de 3 distances, si ces trois distances respectent les inégalités du triangle, alors on l'ajoute au nombre des triangles. Le problème est plutôt simple, mais la difficulté réside dans l'extraction de chaque ligne en C et de chaque nombre dans chaque ligne sachant que ces dernières n'étaient pas de même longueur.

La deuxième partie du problème consiste cette fois à calculer le nombre de triangles créés cette fois-ci avec trois distances verticalement trois par trois (sans compter une même distance dans 2 triangles).

4	21	894
419	794	987
424	797	125
651	305	558
655	631	963
2	628	436
736	50	363
657	707	408
252	705	98
532	173	878
574	792	854
157	737	303
468	76	580
502	503	434
467	567	310
911	911	391
791	913	925
174	49	532
796	803	426
800	132	710
273	722	711

*Extrait de l'input*

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
(base) clement@macbook-pro-de-clement Day 3 % gcc -Wall -o main main.c
(base) clement@macbook-pro-de-clement Day 3 % ./main
reponse 1 : 983
reponse2: 1836
(base) clement@macbook-pro-de-clement Day 3 %
```

## iv. Jour 4

Le problème du quatrième jour consiste, à partir d'une liste contenant l'ensemble des pièces réelles du lieu ainsi que des pièces imaginaires, à trouver toutes ces pièces réelles. Une fois que cela est fait, l'objectif est alors de retourner la somme des identifiants (Id) de ces pièces. Une pièce est alors décrite par une ligne d'instruction de la forme "`a-b-c-d-e-f-g-h-987[abcde]`" par exemple et une pièce est dite réelle (et non un leurre) si la somme de contrôle correspond aux cinq lettres les plus courantes du nom crypté, dans l'ordre, les ex-æquo étant départagés par l'ordre alphabétique. Dans cet exemple, c'est bien le cas car les lettres sont listés dans l'ordre de l'alphabet tout comme l'est la somme de contrôle. L'Id de la chambre, ici 987, est alors retenu dans la somme totale.

Ce problème est bien plus difficile que ceux des jours précédents et nous avons alors du mettre en place la notion de dictionnaire pour pouvoir obtenir ce que l'on souhaitait. En effet, dans un premier temps, notre algorithme consiste à lire le début de l'instruction jusqu'à atteindre le début de l'Id et on ajoute alors dans un dictionnaire la fréquence des éléments qui apparaissent. Ensuite, on lit l'Id qu'on enregistre pour la suite et une fois qu'on atteint le premier crochet, on peut alors commencer à lire la somme de contrôle. Toutefois, il nous faut ici savoir quels sont les 5 caractères avec la plus grande fréquence dans le dictionnaire. On réalise alors un parcours simple de recherche des 5 "meilleurs" puis on crée un nouveau dictionnaire comprenant ces 5 éléments auquel on ajoute cette fois-ci non plus leur fréquence mais leur classement parmi les meilleurs. Ainsi, en lisant la somme de contrôle caractère par caractère, on peut facilement savoir si l'ordre lexicographique comme l'ordre de contrôle sont vérifiés à partir du classement car il faut obtenir une suite croissante.

La seconde partie du problème consiste à décrypter le nom d'une pièce à partir de son code en sachant que le nom correspond à la rotation de chaque lettre par un nombre de fois égal à l'Id de la pièce. Si l'Id vaut 1 par exemple, alors un 'a' devient un 'b'. On cherche alors à trouver l'Id de la pièce intitulée "*north pole objects*". Ici le code consiste à traduire chacune des lignes jusqu'à obtenir celle que l'on souhaite.

Remarque: Ici, pour s'assurer le découpage de chaque ligne d'instruction comme nous le souhaitions, nous avons utilisé à nouveau le code ASCII qui nous permet d'avoir un contrôle parfait des conditions d'arrêt.

```
- aaaaa-bbb-z-y-x-123[abxyz] is a real room because the most common
  letters are a (5), b (3), and then a tie between x, y, and z, which
  are listed alphabetically.
- a-b-c-d-e-f-g-h-987[abcde] is a real room because although the letters
  are all tied (1 of each), the first five are listed alphabetically.
- not-a-real-room-404[oarel] is a real room.
- totally-real-room-200[decoy] is not.
```

Of the real rooms from the list above, the sum of their sector IDs is 1514.

PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE

```
(base) clement@MacBook-Pro-de-Clement Day 4 % gcc -Wall -o main main.c dico.c
(base) clement@MacBook-Pro-de-Clement Day 4 % ./main
sum IDs = 1514%
(base) clement@MacBook-Pro-de-Clement Day 4 % █
```



## v. Jour 5

Le problème du cinquième jour consiste à décoder un mot de passe à partir d'un identifiant. Le processus se déroule ainsi :

- On crée un hash MD5 à partir de l'ID et d'une série de chiffres
- Si le hash commence par 5 zéros, le premier caractère après les 0 est ajouté dans le mot de passe
- On réitère cette boucle pour toutes les séries de chiffres, jusqu'à qu'on obtienne le mot de passe de la même taille que l'ID

Pour réaliser ce défi, on a dû importer une fonction MD5.cpp déjà existante, il faut donc la compiler avec le code principal avant d'exécuter le challenge. Par ailleurs, cette même fonction sera utilisée à chaque fois qu'un défi nécessitera un hachage MD5.

La deuxième partie du problème nous propose de changer légèrement le projet en regardant les deux caractères qui suivent les 5 zéros. Le sixième détermine la position dans le mot de passe et le septième détermine le caractère à ajouter dans le mot de passe.

For example, if the Door ID is `abc`:

- The first index which produces a hash that starts with five zeroes is `3231929`, which we find by hashing `abc3231929`; the sixth character of the hash, and thus the first character of the password, is `1`.
- `5017308` produces the next interesting hash, which starts with `000008f82...`, so the second character of the password is `8`.
- The third time a hash starts with five zeroes is for `abc5278568`, discovering the character `f`.

In this example, after continuing this search a total of eight times, the password is `18f47a30`.

PROBLEMS   OUTPUT   TERMINAL   DEBUG CONSOLE

```
MD5.cpp:237:3: warning: 'register' storage class specifier is deprecated and incompatible with C++17 [-Wdeprecated-register]
register uint32 a, b, c, d;
~~~~~
MD5.cpp:237:3: warning: 'register' storage class specifier is deprecated and incompatible with C++17 [-Wdeprecated-register]
register uint32 a, b, c, d;
~~~~~
MD5.cpp:237:3: warning: 'register' storage class specifier is deprecated and incompatible with C++17 [-Wdeprecated-register]
register uint32 a, b, c, d;
~~~~~
MD5.cpp:237:3: warning: 'register' storage class specifier is deprecated and incompatible with C++17 [-Wdeprecated-register]
register uint32 a, b, c, d;
~~~~~
4 warnings generated.
(base) clement@macbook-pro-de-clement Day 5 % ./main
symbole obtenu!
symbole obtenu!
symbole obtenu!
symbole obtenu!
symbole obtenu!
symbole obtenu!
symbole obtenu!
symbole obtenu!
reponse 1 : 18f47a30
reponse 2 : 05ace8e3
(base) clement@macbook-pro-de-clement Day 5 %
```

## vi. Jour 6

Le problème du sixième jour consiste à décoder le message du père Noël car ce dernier est brouillé. Heureusement, le signal n'est que partiellement brouillé, et le protocole dans des situations comme celle-ci consiste à passer à un simple code de répétition pour faire passer le message. Dans ce modèle, le même message est envoyé de manière répétée. Tout ce qu'il faut faire est de déterminer quel caractère est le plus fréquent pour chaque colonne. Ici, le problème réside dans le fait qu'on travaille sur les colonnes alors qu'on ne peut lire que ligne par ligne en C. Nous avons alors utilisé à nouveau la notion de dictionnaire avec cette fois-ci deux clés (caractère et numéro de colonne), et pour chaque ligne lue, nous avons ajouté dans ce dernier le caractère lu, sa fréquence ainsi que son numéro de colonne, ou incrémenté la fréquence si l'association existait déjà. Il suffit alors ensuite de renvoyer colonne par colonne le caractère avec la fréquence la plus élevée.

Dans la seconde partie du problème, on réalise alors exactement le même décodage mais cette fois-ci en sachant que la véritable code de répétition à utiliser est celui où on prend les caractères les moins fréquents par colonne. Il suffit alors simplement d'inverser ce que l'on cherche dans notre dictionnaire et le reste du code reste identique.

```
eedadn
drvtee
eandsr
raavrd
atevrs
tsrnev
sdttsa
rasrtv
nssdts
ntnada
svetve
tesnvt
vntsnd
vrdear
dvrsen
enarar
```

The most common character in the first column is `e`; in the second, `a`; in the third, `s`, and so on. Combining these characters returns the error-corrected message, `easter`.

In the above example, the least common character in the first column is `a`; in the second, `d`, and so on. Repeating this process for the remaining characters produces the original message, `advent`.

PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE

```
(base) clement@MacBook-Pro-de-Clement Day 6 % gcc -Wall -o main main.c dico.c
(base) clement@MacBook-Pro-de-Clement Day 6 % ./main
```

```
most common = easter
least common = advent
(base) clement@MacBook-Pro-de-Clement Day 6 %
```

**vii. Jour 7**

Le problème du septième jour consiste à trouver parmi une série d'adresse IP celles qui supportent le TLS (transport-layer snooping). Une adresse supporte le TLS si elle contient en dehors des crochets une chaîne de type « abba » : les deux mêmes lettres à l'intérieur et les deux mêmes lettres à l'extérieur mais différentes des deux premières. Ici, la difficulté est pour chaque ligne d'enregistrer le chaîne de caractère sans enregistrer ce qu'il y a dans les crochets. Ensuite, il faut vérifier qu'une chaîne de type « abba » se trouve dans chaque chaîne enregistrée.

En deuxième partie, le problème demande de vérifier les adresses qui supportent le SSL (super-secret listening). Une adresse supporte le SSL si elle contient entre crochet une chaîne de type « bab » et hors des crochets un homologue de type « aba ». La technique d'enregistrement des chaînes de caractères est proche de celle de la première partie. Mais, cette fois-ci, on utilise une seconde chaîne différente pour garder ce qu'il y a dans les crochets et ainsi faire la recherche initiale du motif de type « aba ». On enregistre alors tous les motifs « aba » de cette dernière puis on cherche les motifs correspondants dans la chaîne principale.

- `abba[mnop]qrst` supports TLS (`abba` outside square brackets).
- `abcd[bddb]xyyx` does not support TLS (`bddb` is within square brackets, even though `xyyx` is outside square brackets).
- `aaaa[qwer]tyui` does not support TLS (`aaaa` is invalid; the interior characters must be different).
- `ioxxoj[asdfgh]zxcvbn` supports TLS (`ioxxo` is outside square brackets, even though it's within a larger string).

PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE

```
(base) clement@macbook-pro-de-clement Day 7 % gcc -Wall -o main main.c
(base) clement@macbook-pro-de-clement Day 7 % ./main
S1=2
S2=0
(base) clement@macbook-pro-de-clement Day 7 %
```

## viii. Jour 8

Le problème du huitième jour consiste à nouveau à décoder une liste d'instruction pour cette fois-ci ouvrir une porte bloquée par une authentification à deux facteurs. L'écran de contrôle de la porte a pour dimensions 50x6 pixels et chacun des pixels est initialement éteint. La liste d'instruction à suivre nous indique alors quels sont les pixels que l'on va allumer ou éteindre. La difficulté provient ici du fait qu'il existe des instructions difficiles qui consistent à faire une rotation des lignes ou colonnes de pixels qui ont d'ores et déjà pu être en fonction. Par exemple, l'instruction "rotate row y=A by B" décale tous les pixels de la rangée A (0 est la rangée supérieure) vers la droite de B pixels. Les pixels qui tomberaient à l'extrémité droite apparaissent à l'extrémité gauche de la rangée. Pour réaliser cette tâche, il faut alors créer un dictionnaire qui permet de garder en mémoire les états de chacun des pixels à un instant t. Chaque pixel aura pour clé ses coordonnées sur l'écran et sa valeur sera son état. Ensuite, il faut lire chacune des lignes d'instruction et réaliser les modifications nécessaires dans le dictionnaire. En apparence, cela paraît simple à comprendre mais il s'avère que c'est l'un des problèmes les plus longs à mettre en oeuvre que l'on a affrontée dans ce challenge. En effet, non seulement il faut savoir manipuler chacune des instructions mais il faut aussi s'assurer que la modification faite sur le dictionnaire soit bien faite, en particulier lorsqu'il faut revenir au départ pour allumer un pixel et qu'il faut faire attention que des erreurs ne s'introduisent lorsqu'on doit de cette manière allumer un pixel qui est déjà allumé. Une fois la liste d'instruction terminée, on cherche alors à savoir le nombre de pixels allumés.

La seconde partie du problème consiste à déchiffrer le code en sachant que l'écran est seulement capable d'afficher des lettres capitales de taille 5x6 pixels. Pour réaliser cette tâche, une fois le traitement des instructions terminé et ces dernières inscrites dans un nouveau fichier avec des signes différents selon que le pixel est allumé ou éteint, on a lu de manière assez rudimentaire le code qui apparaît dans ce nouveau fichier car nous ne voyons pas d'autres solutions simples. Dans l'exemple ci-dessous, on reconnaît sans trop de problèmes le code UPOJFL.

Remarque: Dans la construction de nos algorithmes, pour découper les instructions, nous utilisons autrefois le code ASCII des caractères en lisant chacun des caractères. Nous avons alors découvert grâce aux compléments de cours la fonction `SSCANF` qui permet de découper une ligne d'instruction comme nous le souhaitions à supposer qu'on connaisse sa forme.

+--+	---+
+--+	---+
+--+	---+
+--+	---+
+--+	+++
+--+	+++
	++++
+--+	++---
+--+	++---
+--+	++---
+--+	++---
+---	++---
+---	++---
	++---
+--+	++---
+--+	++---
+--+	++---
+--+	++---
+--+	++---
+--+	++---
+---	++++

**ix. Jour 9**

Le problème du huitième jour demande de travailler sur de la décompression. La compression opère ainsi : les chaînes de caractères normales se lisent normalement (« ABC » se lit comme « ABC »). Dès qu'un motif se répète plusieurs fois, la compression opère. On stockera alors la longueur du motif  $n$  ainsi que le nombre de fois  $m$  qu'il se répète au début du motif à l'aide d'un (nxm) « ABBBC » qui sera compressé en « A(1x3)BC ». Notre but est de calculer le nombre de caractères que contient un fichier décompressé avec comme input le fichier compressé. Pour résoudre ce problème, nous avons procédé itérativement en ajoutant 1 à la longueur du fichier quand on rencontre un caractère normal et  $n \times m$  quand nous rencontrons une partie compressée (en ne comptant pas la chaîne qui suit qui correspond à la compression bien sûr). Dans la première partie du problème, les parties compressées dans les parties compressées sont considérées comme des caractères décompressés, donc il ne sert à rien de stocker une quelconque chaîne de caractères. Ainsi, en effectuant les étapes évoquées ci-dessus, on arrive au bout de l'étape 1.

Pour la deuxième partie du problème, les compressions dans les compressions ne sont plus ignorées ce qui augmente drastiquement la taille de la chaîne décompressée. Pour ces raisons, nous avons utilisé des entiers en 64 bits pour pouvoir les stocker. Ici, nous avons dû stocker chaque motif compressé pour vérifier si ces derniers contiennent eux-mêmes des motifs compressés. On effectue ensuite une récursion sur ces chaînes de caractères pour calculer leur taille décompressée.

```
- A(2x2)BCD(2x2)EFG doubles the BC and EF, becoming ABCBCDEFEG for a
  decompressed length of 11.

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

(base) clement@macbook-pro-de-clement Day 9 % gcc -Wall -o main main.c
main.c:27:26: warning: length modifier 'l64' results in undefined behavior or no effect with 'd' conversion specifier [-Wformat]
    printf("reponse 2 : %l64d\n", length2(chaine));    // Essayer %lld si problème
                          ~~~~~
1 warning generated.
(base) clement@macbook-pro-de-clement Day 9 % ./main
reponse 1 : 11
```

## x. Jour 10

Le problème du dixième jour consiste à organiser un essaim de robots transporteurs de puces. Chaque robot peut porter au maximum deux puces en même temps et sa fonction principale est activée lorsque ce dernier possède exactement deux puces, et dans ce cas il donne ses deux puces à d'autres robots ou un conteneur et est désactivé. Ainsi, il nous est donné une liste d'instruction dans un ordre totalement aléatoire qui précise quels robots reçoivent initialement une puce et quelles sont ensuite les transmissions qui s'effectuent entre les différents robots. Notre objectif est alors, étant deux IDs de puce, de savoir quel est le robot qui a eu entre ses mains ces deux puces.

Remarque: Les consignes pour ce jour étaient très peu claires et nous avons un léger doute quant au fait que ce soit cela qui était attendu. Toutefois, l'exemple donné suppose une telle démarche.

Pour ce faire, il faut tout d'abord initialiser un dictionnaire qui permet de savoir quel robot possède quelle puce à un instant donné. Ensuite, il faut lire l'intégralité du fichier une première fois pour savoir à quels robots sont données les premières puces. Une fois ceci réalisé, il faut alors lire à nouveau un certain nombre de fois (à vrai dire jusqu'à obtenir le résultat voulu) pour que les transmissions possibles soient réalisées. La difficulté ici est que le fichier n'est pas ordonné et il se peut que la toute première instruction concerne un robot qui n'a pas encore de puce. Il faut donc ajouter cette instruction en queue de liste dans chaque boucle jusqu'à ce que ce robot possède ses deux puces. La stratégie de notre algorithme consiste exactement à suivre ce procédé. Afin d'optimiser le temps de calcul et pour avoir une condition d'arrêt nette, nous avons fait le choix d'avoir tout au long de notre algorithme un tableau qui garde en mémoire les indices qu'il faut encore traiter. La taille du tableau est donc nécessairement décroissante ce qui assure que notre algorithme se terminera.

Malheureusement, et en lien avec la remarque ci-dessus, nous avons un doute sur la consigne car nous n'avons jamais réussi à obtenir le résultat voulu. Pour autant, avec notre compréhension de la consigne, notre algorithme fonctionne bien sûr les exemples ainsi que sur une modification de l'ordre des instructions de ce dernier. Après plus de 6h sur cet algorithme, nous avons décidé d'abandonner pour essayer de regarder la suite.

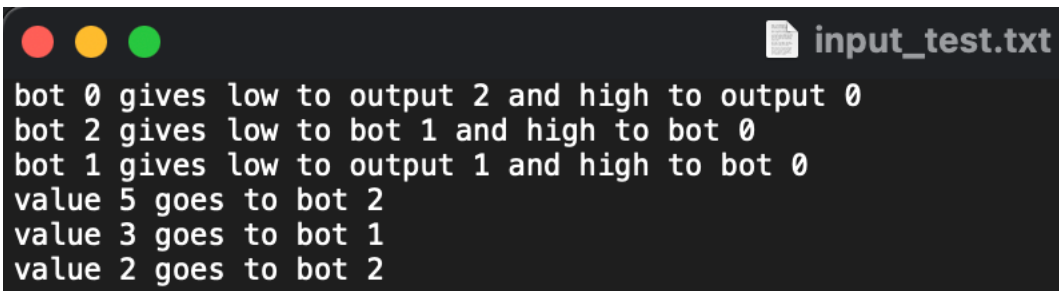
```
value 5 goes to bot 2
bot 2 gives low to bot 1 and high to bot 0
value 3 goes to bot 1
bot 1 gives low to output 1 and high to bot 0
bot 0 gives low to output 2 and high to output 0
value 2 goes to bot 2
```

- Initially, bot 1 starts with a value-3 chip, and bot 2 starts with a value-2 chip and a value-5 chip.
- Because bot 2 has two microchips, it gives its lower one (2) to bot 1 and its higher one (5) to bot 0.
- Then, bot 1 has two microchips; it puts the value-2 chip in output 1 and gives the value-3 chip to bot 0.
- Finally, bot 0 has two microchips; it puts the 3 in output 2 and the 5 in output 0.

In the end, output bin 0 contains a value-5 microchip, output bin 1 contains a value-2 microchip, and output bin 2 contains a value-3 microchip. In this configuration, bot number 2 is responsible for comparing value-5 microchips with value-2 microchips.

PROBLEMS   OUTPUT   TERMINAL   DEBUG CONSOLE

```
(base) clement@macbook-pro-de-clement Day 10 % g++ -std=c++11 -Wall -o test_try test_try.cpp
(base) clement@macbook-pro-de-clement Day 10 % ./test_try
Le bot 2 est responsable de la comparaison
(base) clement@macbook-pro-de-clement Day 10 %
```



```
bot 0 gives low to output 2 and high to output 0
bot 2 gives low to bot 1 and high to bot 0
bot 1 gives low to output 1 and high to bot 0
value 5 goes to bot 2
value 3 goes to bot 1
value 2 goes to bot 2
```

PROBLEMS   OUTPUT   TERMINAL   DEBUG CONSOLE

```
(base) clement@macbook-pro-de-clement Day 10 % g++ -std=c++11 -Wall -o test_try test_try.cpp
(base) clement@macbook-pro-de-clement Day 10 % ./test_try
Le bot 2 est responsable de la comparaison
(base) clement@macbook-pro-de-clement Day 10 %
```

## xi. Jour 11

Le jour 11 était clairement l'un des défis les plus compliqués qu'on ait eu à faire. Il s'agit de transporter des objets d'étages en étages jusqu'à qu'ils soient tous au dernier étage de l'immeuble. Il y a quatre étages et un ascenseur qui permet de se déplacer. Le but est de trouver le nombre de pas minimum au transport de ces objets jusqu'au dernier étage : le quatrième. Les objets sont des générateurs et des microchips associés. Il y a différentes règles à respecter lors du déplacement qui sont détaillées sur la page du challenge et que j'ai mis en commentaire dans notre code au moment où elles interviennent. On a rajouté deux règles qui permettent de converger plus rapidement vers la solution sans déformer le problème :

- On ne descend pas s'il n'y a pas d'objet dans l'étage du bas.
- On ne monte pas sans objet avec nous.
- On ne repasse deux fois par le même état que si on trouve un temps plus faible pour y arriver.

Si les deux premières règles sont simples à implémenter, la troisième a nécessité l'implémentation d'une map contenant tous les états parcourus.

Pour cette raison, notre idée initiale de créer des classes dynamiques spécifiquement pour ce problème n'est pas adapté car chaque état prendrait une mémoire trop importante. On a donc implémenté chaque état par des bitsets, dont les indices correspondent aux objets et les valeurs correspondent à l'étage des objets, chaque étage est donc constitué de 2 bits (pour coder de l'étage 1 à l'étage 4).

L'algorithme revient à faire une recherche récursive dans un arbre de décision et en ne gardant que les chemins valides selon nos règles. On ne regarde pas non plus les chemins qui prennent plus longtemps que le temps auquel on a déjà trouvé une solution.

**Remarque:** Concrètement, on utilise la fonction `set_item_floor` : les microchips et générateurs de même types ont leurs premiers entiers respectifs qui se suivent : ainsi 0 et 1 sont le microchip et le générateur d'une même matière (peu importe la matière d'ailleurs). Le deuxième entier est l'étage de l'objet.

### Input:

```
'The first floor contains a polonium generator, a thulium generator, a thulium-compatible microchip, a promethium generator, a ruthenium generator, a ruthenium-compatible microchip, a cobalt generator, and a cobalt-compatible microchip.
The second floor contains a polonium-compatible microchip and a promethium-compatible microchip.
The third floor contains nothing relevant.
The fourth floor contains nothing relevant.'
```

```
State initial_state;

/*
initial_state.set_item_floor(0,1);
initial_state.set_item_floor(1,0);
initial_state.set_item_floor(2,2);
initial_state.set_item_floor(3,0);
*/

//en première position on place le numéro de l'objet.
//Les nombres pairs sont les générateurs
//les nombres impairs qui les suivent sont les chips correspondants
//en deuxième position est l'étage de l'objet qui va de 0 à 3.
initial_state.set_item_floor(0,0);
initial_state.set_item_floor(1,1);
initial_state.set_item_floor(2,0);
initial_state.set_item_floor(3,0);
initial_state.set_item_floor(4,0);
initial_state.set_item_floor(5,1);
initial_state.set_item_floor(6,0);
initial_state.set_item_floor(7,0);
initial_state.set_item_floor(8,0);
initial_state.set_item_floor(9,0);

initial_state.set_item_floor(10,0);
initial_state.set_item_floor(11,0);
initial_state.set_item_floor(12,0);
initial_state.set_item_floor(13,0);
```



```
C:\Users\dansi\OneDrive - Universite de Lorraine\Bureau\adventofcode2016\day111>g++ -o hello main.cpp  
C:\Users\dansi\OneDrive - Universite de Lorraine\Bureau\adventofcode2016\day111>.\hello  
reponse: 71
```

**xii. Jour 12**

Le problème du douzième jour consiste à mettre à jour des registres initialisés à 0 à partir d'une liste d'instruction à suivre dans l'ordre et contenant quatre types d'instruction:

- *cpy x y* qui demande de copier la valeur du registre x ou la valeur de l'entier x dans le registre y
- *inc x* qui demande à augmenter la valeur du registre x de 1
- *dec x* qui demande à diminuer la valeur du registre x de 1
- *jnz x y* qui demande à ce qu'on saute ou revienne à l'instruction située à y déplacement(s) (-2 par exemple signifie qu'il faut revenir deux instructions au-dessus) et si seulement la valeur du registre x est non nulle.

Résoudre ce problème semble en soi pas si difficile et nous avons opté pour stratégie un algorithme qui garde en mémoire dans un dictionnaire les valeurs des quatre registres a, b, c et d puis qui lit chacune des instructions une à une en réalisant les changements nécessaires.

Remarque: Pour la première partie du problème, notre algorithme malgré que ce dernier met tout de même une minute à trouver la solution mais nous n'avons pas réussi à optimiser plus que cela le code.

La seconde partie du problème consiste exactement au même problème sauf qu'on initialise cette fois-ci la valeur du registre c à 1. Le code ne change donc pas mais son efficacité faisant d'ores et déjà défaut pour la première partie du problème est encore plus visible pour la seconde partie de ce problème. Nous avons arrêté le chrono car ce dernier dépassait largement la dizaine de minutes même si nous avons obtenu le résultat.

Commentaire: Si une optimisation est possible ou s'il existe un défaut dans notre algorithme, nous serions ravis d'avoir un retour du lecteur.

```
cpy 41 a
inc a
inc a
dec a
jnz a 2
dec a
```

The above code would set register `a` to `41`, increase its value by `2`, decrease its value by `1`, and then skip the last `dec a` (because `a` is not zero, so the `jnz a 2` skips it), leaving register `a` at `42`. When you move past the last instruction, the program halts.

PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE

```
(base) clement@macbook-pro-de-clement Day 12 % g++ -std=c++11 -Wall -o main main.cpp
(base) clement@macbook-pro-de-clement Day 12 % ./main
42
(base) clement@macbook-pro-de-clement Day 12 %
```

## xiii. Jour 13

Le problème du treizième jour consiste simplement en l'implémentation de l'algorithme de chemin optimal A\*. Initialement, il nous faut créer la grille de déplacement à l'aide d'un système binaire qui n'est pas d'une grande difficulté lorsqu'on utilise `std::bitset`. Une fois ceci fait, il suffit alors de réaliser l'algorithme A\* à partir d'un point de départ et d'arrivée. La réelle complexité de ce problème repose essentiellement dans la maîtrise des données que nous utilisons, surtout lorsqu'on utilise sous C++ le type file prioritaire `std::queue`. Dans notre cas, nous avons eu pas mal de petites erreurs qui se sont introduites dans le code, ce qui a rendu difficile et long à arriver au résultat mais nous y sommes arrivés. La seconde partie du problème consistait toujours sur l'utilisation de l'algorithme du chemin optimal sauf que cette fois-ci on souhaitait obtenir le nombre de points possibles avec au maximum 50 pas. Il faut donc boucler sur l'ensemble des arrivées possibles dans un carré 50x50 pour obtenir le résultat.

Now, suppose you wanted to reach `7,4`. The shortest route you could take is marked as `0`:

```

0123456789
0 .#.####.##
1 .0#..#...#
2 #000.##...
3 ###0#.###.
4 .##00#00#.
5 ..##000.#.
6 #...##.###

```

Thus, reaching `7,4` would take a minimum of `11` steps (starting from your current location, `1,1`).

PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE

```

(base) clement@MacBook-Pro-de-Clement Day 13 % g++ -std=c++11 -Wall -O3 -o main main.cpp
(base) clement@MacBook-Pro-de-Clement Day 13 % ./main
il faut au moins 11 pas
(base) clement@MacBook-Pro-de-Clement Day 13 % █

```

**xiv. Jour 14**

Le problème du quatorzième jour consiste à trouver un ensemble de clés basé sur le cryptage MD5. Prenons alors une chaîne de caractères “abc0”. Il est alors possible de faire un hachage MD5 de cette chaîne. On définit alors dans cet exercice qu’un hachage MD5 est une clé si:

- Le hachage contient 3 caractères à la suite identique, par exemple 666
- L’un des 1000 hachages suivants, c’est-à-dire parmi ceux de “abc1” à “abc1000” possède 5 caractères identiques à la suite et similaire à la chaîne de départ, donc la suite de notre exemple serait 66666.

Pour résoudre ce problème, il faut et suffit donc de réfléchir assez naïvement en calculant le hachage MD5 des chaînes “abcI” avec  $I \in \mathbb{N}$ , vérifier si cette dernière possède 3 caractères identiques à la suite et si c’est le cas on vérifie alors si parmi les 1000 chaînes suivantes, il est possible de trouver un hachage contenant 5 caractères identiques à la suite comme définis plus haut. On répète alors le processus jusqu’à obtenir la 64ème clé qui est l’élément de réponse attendu.

La seconde partie du problème consiste exactement au même problème sauf que cette fois-ci on ne hache pas la chaîne seulement une fois mais 2016 fois de plus à chaque fois qu’on doit réaliser un hachage MD5. L’algorithme ne change donc pas hormis pour la partie hachage.

Remarque: La première partie du problème se résout très bien avec notre algorithme. En revanche, ce dernier est trop naïf pour la seconde partie du problème. Sans avoir été mis au point pour des raisons de temps et de complexité, nous avons tout de même imaginé une solution algorithmique plus élégante pour potentiellement faire face au temps de calcul important. Au lieu de réaliser naïvement à chaque fois la recherche d’un triplet et d’un quintuple, au risque de travailler un grand nombre de fois sur les mêmes chaînes, notre astuce serait de commencer par trouver la première chaîne contenant un triplet. Une fois, cette dernière trouvée, on l’ajoute alors son index  $I$  et le caractère du triplet à une liste d’index sur lesquelles on travaille et on commence ensuite la recherche d’un quintuple. Toutefois, pour accélérer le processus, on va chercher à travailler une unique fois sur les 1000 chaînes suivantes en recherchant parmi les hachages MD5 de ces dernières les triplets, et on ajoute alors l’index et le caractère associé à la liste d’index si le triplet existe, et on recherche par la même occasion le quintuple voulu, ainsi que d’autres quintuples, et s’ils existent, on les ajoute à un dictionnaire prenant en clé l’index ainsi que le caractère. De cette manière, on réalisera alors une et une seule fois le calcul de triplet sur chaque chaîne et de même pour les quintuplés, car nous aurons simplement besoin d’aller les informations nécessaires dans un dictionnaire.

- The first index which produces a triple is 18, because the MD5 hash of abc18 contains ...cc38887a5... However, index 18 does not count as a key for your one-time pad, because none of the next thousand hashes (index 19 through index 1018) contain 88888.
- The next index which produces a triple is 39; the hash of abc39 contains eee. It is also the first key: one of the next thousand hashes (the one at index 816) contains eeeee.
- None of the next six triples are keys, but the one after that, at index 92, is: it contains 999 and index 200 contains 99999.
- Eventually, index 22728 meets all of the criteria to generate the 64th key.

PROBLEMS   OUTPUT   TERMINAL   DEBUG CONSOLE

```
(base) clement@macbook-pro-de-clement Day 14 % g++ -std=c++11 -Wall -O3 -o main main.cpp md5.cpp
(base) clement@macbook-pro-de-clement Day 14 % ./main
22728 c
(base) clement@macbook-pro-de-clement Day 14 %
```

## xv. Jour 15

Le problème du quinzième jour consiste à trouver le temps à partir duquel il est préférable de jouer à une machine à jeu lorsqu'on connaît son mécanisme. La machine à jeu est composée d'un set de disque, qui est ici notre input, et chaque disque possède un certain nombre de positions possibles ainsi qu'une position initiale à  $t=0$ . Par ailleurs, à chaque nouvel instant  $t$ , chaque disque voit sa position être incrémentée de 1. De plus, pour que l'objet traverse un disque, il faut que ce dernier soit sur sa position 0 ! Ainsi, si à  $t=0$  l'objet est lancé et que le premier disque est initialement en position 0 et possède deux positions possibles, alors l'objet arrive sur ce disque à l'instant  $t=1$  et le disque sera en position 1, ce qui fait que l'objet n'est pas gagné. Il aurait fallu pour gagner lancer la balle à l'instant  $t=1$  car, dans ce cas, à l'instant  $t=2$ , le disque aurait été en position 1. Pour résoudre ce problème, il suffit alors de résoudre le problème d'optimisation suivant:

$$\min\{t \in \mathbb{R} \mid \sum_{i=1}^{N_{\text{disque}}} [(pos_{\text{initiale}} + t + i) \bmod nb_{\text{position}}] = 0\}$$

Dans la seconde partie du problème, le problème est exactement le même sauf qu'on ajoute un nouveau disque. Il suffit alors simplement de modifier et d'ajouter ce nouveau disque à l'input sans changer le code.

```
For example, at time=0, suppose you see the following arrangement:
Disc #1 has 5 positions; at time=0, it is at position 4.
Disc #2 has 2 positions; at time=0, it is at position 1.

If you press the button exactly at time=0, the capsule would start to fall;
it would reach the first disc at time=1. Since the first disc was at
position 4 at time=0, by time=1 it has ticked one position forward. As a
five-position disc, the next position is 0, and the capsule falls through
the slot.

Then, at time=2, the capsule reaches the second disc. The second disc has
ticked forward two positions at this point: it started at position 1, then
continued to position 0, and finally ended up at position 1 again. Because
there's only a slot at position 0, the capsule bounces away.

If, however, you wait until time=5 to push the button, then when the
capsule reaches each disc, the first disc will have ticked forward 5+1 = 6
times (to position 0), and the second disc will have ticked forward 5+2 = 7
times (also to position 0). In this case, the capsule would fall through
the discs and come out of the machine.
```

PROBLEMS

OUTPUT

TERMINAL

DEBUG CONSOLE

```
(base) clement@macbook-pro-de-clement Day 15 % g++ -std=c++11 -Wall -O3 -o main main.cpp
(base) clement@macbook-pro-de-clement Day 15 % ./main
Il faut jouer à partir du temps t=5
(base) clement@macbook-pro-de-clement Day 15 %
```

## xvi. Jour 16

Le problème du seizième jour consiste à trouver une chaîne de caractères particulière à partir d'une chaîne donnée en entrée. Pour ce faire, on part d'une chaîne donnée en entrée ainsi que d'une longueur de chaîne voulue. Si la chaîne en entrée n'est pas de la taille de la chaîne voulue, alors on lui applique un protocole de *dragon curve* qui suit le protocole suivant:

- Prendre une chaîne a, et la copier dans une autre chaîne b
- Inverser b et appliquer une négation de ses éléments, ici 0 devient 1 et inversement
- Renvoyer la chaîne  $a = a + "0" + b$

afin d'obtenir une chaîne plus longue. Il est possible qu'après ce procédé, la chaîne soit devenue trop longue. On conserve alors seulement les caractères permettant d'avoir la taille de chaîne voulue. On cherche alors ensuite la première chaîne de longueur impaire qui passe dans le processus suivant. Si la chaîne est de longueur impaire, alors on crée une nouvelle chaîne où tous les doublets de caractères de la chaîne initiale sont transformés en unique caractère avec:

- "01" ou "10" qui devient "0"
- "11" ou "00" qui devient "1"

On aura alors nécessairement une chaîne qui sera de plus en plus courte et l'algorithme se terminera bien car dans le pire des cas, on retombera sur une chaîne de longueur 1 qui est bien impaire.

Les deux parties du problème correspondent à une longueur différente en entrée mais le code est exactement le même.

Combining all of these steps together, suppose you want to fill a disk of length 20 using an initial state of 10000:

- Because 10000 is too short, we first use the modified dragon curve to make it longer.
- After one round, it becomes 10000011110 (11 characters), still too short.
- After two rounds, it becomes 1000001111001000011110 (23 characters), which is enough.
- Since we only need 20, but we have 23, we get rid of all but the first 20 characters: 10000011110010000111.
- Next, we start calculating the checksum; after one round, we have 011110101, which 10 characters long (even), so we continue.
- After two rounds, we have 01100, which is 5 characters long (odd), so we are done.

In this example, the correct checksum would therefore be 01100.

PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE

```
(base) clement@macbook-pro-de-clement Day 16 % g++ -std=c++11 -Wall -O3 -o main main.cpp
(base) clement@macbook-pro-de-clement Day 16 % ./main
Le résultat est: 01100
(base) clement@macbook-pro-de-clement Day 16 %
```

## xvii. Jour 17

Le problème du 17ème jour consiste à trouver comme au jour 16 le plus court chemin entre deux points sauf que cette fois-ci, le labyrinthe est en perpétuel déplacement, c'est-à-dire que les points qui sont inatteignables changent au cours du temps. Ainsi, le point de départ est le point de coordonnées (0,0) et, initialement, nous avons en input une chaîne de caractères. Cette dernière peut être hachée par MD5 pour obtenir une nouvelle chaîne de caractères. Seuls les quatre premiers caractères du hachage sont utilisés; ils représentent, respectivement, les portes en haut, en bas, à gauche et à droite de notre position actuelle. Tout 'b', 'c', 'd', 'e' ou 'f' signifie que la porte correspondante est ouverte ; tout autre caractère (tout chiffre ou 'a') signifie que la porte correspondante est fermée et verrouillée. Lorsque l'on avance avec une porte ouverte, par exemple 'R' (pour Right), on travaille alors ensuite avec une nouvelle chaîne de caractères qui est la concaténation de la première avec 'R', ce qui permet d'obtenir 4 nouvelles combinaisons d'ouverture/fermeture de porte pour cette nouvelle position. L'objectif est alors de trouver la combinaison de déplacement la plus courte pour aller au point de coordonnées (6,6).

Pour ce faire, il faut alors reprendre l'algorithme A\* sauf que cette fois-ci, il faut prendre en compte et mettre à jour la chaîne initiale pour savoir comment on peut se déplacer étant donné un point du plan. Le réel changement s'opère donc dans la recherche des voisins mais également dans la façon dont on travaille les noeuds. En effet, il faut désormais considérer des nouvelles doublement chaînés pour savoir à tout moment qui est le père et les fils d'un noeud. Ceci est obligatoire car il y a possibilité de revenir en arrière sur ces pas tout en considérant dans ce problème que le retour en arrière fait partie intégrante du chemin (même si cela vous ramène dans la pièce où vous avez commencé, votre chemin ouvre un ensemble différent de portes). Pour le type noeud, on travaille toujours avec un vecteur d'entiers cette fois-ci plus long et de la forme suivante:

**Node := (position\_x, position\_y, coût, heuristique, parent\_x, parent\_y, fils\_x, fils\_y, mvt\_x, mvt\_y)**

où (mvt\_x, mvt\_y) est le déplacement partant du père pour arriver à ce noeud.

La deuxième partie du problème consiste à l'inverse à trouver le chemin le plus long et non le chemin le plus court. Il suffit en théorie donc de chercher tous les chemins possibles en étudiant toutes les possibilités possibles et en ne s'arrêtant pas dès le plus court trouvé. Malheureusement, nous ne sommes jamais arrivés à atteindre un nombre de pas aussi important que données dans leurs exemples et nous ne comprenons pas la démarche qu'ils ont utilisée pour atteindre un aussi grand nombre sur un damier 7x7.

- If your passcode were `ihgpwlah`, the shortest path would be `DDRRRD`.
- With `kglvqrro`, the shortest path would be `DDUDRLRRUDRD`.
- With `ulqzkmiv`, the shortest would be `DRURDRUDDLLDLUURRDULRLDUUDDRR`.

PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE

```
(base) clement@MacBook-Pro-de-Clement Day 17 % g++ -std=c++11 -Wall -O3 -o main main.cpp md5.cpp
(base) clement@MacBook-Pro-de-Clement Day 17 % ./main
password: ihgpwlah
chemin : DDRRRD, nb pas: 6
(base) clement@MacBook-Pro-de-Clement Day 17 % g++ -std=c++11 -Wall -O3 -o main main.cpp md5.cpp
(base) clement@MacBook-Pro-de-Clement Day 17 % ./main
password: kglvqrro
chemin : DDUDRLRRUDRD, nb pas: 12
(base) clement@MacBook-Pro-de-Clement Day 17 % g++ -std=c++11 -Wall -O3 -o main main.cpp md5.cpp
(base) clement@MacBook-Pro-de-Clement Day 17 % ./main
password: ulqzkmiv
chemin : DRURDRUDDLLDLUURRDULRLDUUDDRR, nb pas: 30
(base) clement@MacBook-Pro-de-Clement Day 17 %
```





### III/ Conclusion

Pour conclure ce projet, nous sommes fiers d'avoir atteint le jour 18 malgré quelques embûches. Notre beau sapin de Noël peut d'ailleurs témoigner de ces dernières. Plus globalement, ce projet nous a permis d'obtenir de nombreuses compétences en C/C++ et ce de manière très rapide par rapport aux langages précédents que nous aurions pu apprendre. Ainsi, nous en retenons une expérience très positive et sommes sûr que les compétences acquises nous seront fortement utiles dans la suite de nos parcours respectifs.

