



## Complexité Et Algorithmes

TD :

Problème ONEZERO et DIGIT JUMP

**RAPPORT**

**Rédigé par :**

KAFANDO Rodrique

KOUADIO Kouamé Olivier

**Enseignant :**

M. Do Phan Thuan

INTRODUCTION .....	3
PARTIE I : L'APPROCHE ONESANDZEROS .....	5
I- Le probleme .....	5
I.1- Description du problème .....	5
I.2- Domaine d'application du problème .....	5
II- La methode de base utilisee pour resoudre le problème.....	6
II- l'agorithme de resolution .....	7
IV- Analyse de la complexité du problème .....	8
Code source du programme .....	9
V- Evaluation (test d'acceptation sur sphere on ligne judge) .....	11
VI- Expérimentation sur notre algorithme .....	12
VII- Evaluation du temps de calcul avec themis.....	15
PARTIE II : L'APPROCHE DIGIT JUMP .....	19
I- Description complète du problème .....	19
II- Exemples d'application du problème dans la vie réelle .....	19
III- Écrire brièvement la connaissance de base de la méthode utilisée pour résoudre le problème .....	19
IV- Écrire l'algorithme en détail pour résoudre le problème et la preuve de l'exactitude de l'algorithme .....	20
V-) Analyse de la complexité du problème .....	21
VI-) Implémentation.....	21
VII-) Analyser du pseudo code .....	21
VIII-) Capture de l'évaluation sur le site CodeChef.com .....	23
IX-) Présentation des résultats issus des différents tests .....	23
CONCLUSION GENERALE.....	24
REFERENCES.....	25

## INTRODUCTION

La complexité algorithmique consiste en l'analyse du comportement asymptotique des algorithmes en termes de temps de calcul et d'espace mémoire. Ce comportement est étudié toujours par rapport à la taille des données fournies en entrée.

Par exemple, pour un algorithme traitant en entrée un tableau, on considérera la taille de ce tableau. Pour un graphe, on pourra considérer suivant le cas le nombre de nœuds et/ou le nombre de liens. Pour un arbre, on considérera aussi suivant le cas et la nature de l'arbre, sa profondeur, son nombre de feuilles, etc.

# **PREMIERE PARTIE : PROBLEME ONEZERO**

## PARTIE I : L'APPROCHE ONESANDZEROS

### I- Le probleme

#### I.1- Description du problème

#### La description

Certains entiers positifs ont leur représentation décimale composée uniquement de zéros et d'au moins un chiffre, par exemple 101. Si un entier positif n'a pas une telle propriété, on peut essayer de la multiplier par un entier positif pour savoir si le produit a cette propriété.

#### Spécification d'entrée

Le nombre K des cas de test dans chacune des lignes K suivantes est un entier n ( $1 \leq n \leq 20000$ ).

#### Spécification de sortie

Pour chaque cas de test, le programme doit calculer le plus petit multiple du nombre n composé uniquement des chiffres 1 et 0 (en commençant par 1).

#### Exemple d'entrée

```
3
17
11011
17
```

#### Exemple de sortie

```
11101
11011
11101
```

Dans ce problème le programme doit calculer le multiple minimum d'un nombre donné, ce multiple minimum ne peut contenir que les chiffres 1 et 0.

#### I.2- Domaine d'application du problème

Notre problème peut s'appliquer dans le domaine d'indexation. À partir des données en entrées le programme donne le résultat dans un type bien spécifique.

## II- La methode de base utilisee pour resoudre le problème

L'algorithme de base utilisé par notre problème de ONEZERO est l'algorithme BFS. Il est important de noter que **Breadth-first search (BFS)** est un algorithme pour parcourir ou rechercher des structures de données arborescentes ou graphiques. Il commence à la racine de l'arbre (ou à un nœud arbitraire d'un graphique, parfois appelé «clé de recherche») et explore d'abord les nœuds voisins avant de passer aux voisins du prochain niveau.

### ❖ Pseudo code

```
❖ Breadth - First - Search ( Graph , Root , Goal ) :  
❖  
❖     créer un jeu vide Vérifié créer une file d'attente vide Queue  
❖  
❖     Ajoutez Root to Checked  
❖     Queue . enqueue ( Root )  
❖  
❖     alors que la file d'attente n'est pas vide {  
❖         Current = Queue . dequeue ()  
❖         si courant . a ( but ) {  
❖             retour actuel  
❖         }  
❖         pour chaque nœud qui est adjacente à courant {  
❖             si nœud est pas dans Vérifié {  
❖                 Vérifié . ajouter ( Node )  
❖                 Queue . enqueue ( Node )  
❖             }  
❖         }  
❖     }
```

### ❖ Illustration

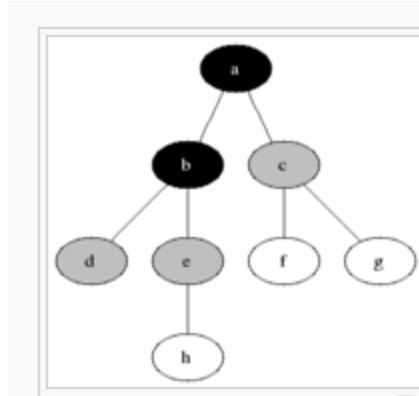


Figure 1: illustration de l'algorithme de BFS

## II- l'agorithme de resolution

Dans cette étape de notre rapport nous expliquons l'idée de ce problème soumis à notre étude. Nous supposons un nombre  $X$  tel que  $X \bmod N = 0$ . Nous stockons uniquement les états de prend  $N$  au cours de l'exécution c'est-à-dire de 0 à  $n-1$  tout en commençant par 1. Nous mettons dans notre cas l'approche BFS en œuvre donc si l'état actuel modulo est  $Y$ , nous ajoutons 0 a cela et nous calculons  $Y * 10 + 0$  et nous trouvons son modulo qui conduira à un nouveau modulo a un nouvel état. Ensuite nous refaisons de la même façon nous ajoutons 1 puis nous calculons  $Y * 10 + 1$  et nous trouvons son modulo.

EX : si  $Y=11$  on ajoute 0 pour obtenir 110 et on ajoute 1 pour obtenir 111.

À la suite de cela nous stockons dans un tableau parent l'état du modulo précédent à partir duquel l'état modulo est atteint. Ce tableau parent a pour principal rôle de point de contrôle pour vérifier si un état modulo est déjà vérifié ou non. Puis dans un autre tableau nous stockons les valeurs (1 ou 0) qui est ajouté à l'état parent pour obtenir le modulo actuel. Une fois que l'état modulo 0 est atteint nous arrêtons le BFS et le Back Track en utilisant le tableau parent et le tableau des valeurs pour obtenir le nombre c'est-à-dire le plus petit nombre  $N$  composé uniquement des chiffres 1 et 0 commençant par 1.

```
#define REP(i,n) for(int i=0; i<n; i++)
#define FOR(i,st,end) for(int i=st; i<end; i++)
#define db(x) cout << (#x) << "=" << x << endl;
#define mp make_pair
#define pb push_back
#define mod 1000003
int parent[20000];
typedef long long int ll;
```

```
queue<int>q;
int temp,currentState;
```

```

int value[20005];
void solve(int n){
    q.push(1);
    parent[1]=0;
    while(!q.empty()){
        currentState=q.front();
        q.pop();
        if(currentState==0){
            stack<int> s;
            while(parent[currentState]){
                s.push(value[currentState]);
                currentState=parent[currentState];
            }
            s.push(1);
            while(!s.empty()){
                printf("%d", s.top());
                s.pop();
            }
            printf("\n");
            break;
        }
        temp=(currentState*10)%n;

        if(parent[temp]==-1){
            q.push(temp);
            parent[temp]=currentState;
            value[temp]=0;
        }
        temp=currentState*10+1;
        temp%=n;
        if(parent[temp]==-1){
            q.push(temp);
            parent[temp]=currentState;
            value[temp]=1;
        }
    }
}

```

#### IV- Analyse de la complexité du problème

Après plusieurs tests nous avons constaté que nous ne devons pas avoir une valeur supérieure 20000. À cette valeur le programme ne parvient pas à afficher le résultat. Après analyse, notre algorithme a une complexité linéaire  $O(n)$ .



## Code source du programme

```
#include
<vector>

#include <list>
#include <map>
#include <set>
#include <deque>
#include <queue>
#include <stack>
#include <string>
#include <bitset>
#include <algorithm>
#include <functional>
#include <numeric>
#include <utility>
#include <sstream>
#include <iostream>
#include <iomanip>
#include <cstdio>
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <cstring>
#include <climits>
#include <stdlib.h>
#include <stdio.h>

using namespace std;

#define REP(i,n) for(int i=0; i<n; i++)
#define FOR(i,st,end) for(int i=st;i<end;i++)
#define db(x) cout << (#x) << "=" << x << endl;
#define mp make_pair
#define pb push_back
#define mod 1000003

int parent[20005];
typedef long long int ll;

queue<int>q;
```

```

int temp,currentState;
int value[20005];
void solve(int n){
    q.push(1);
    parent[1]=0;
    while(!q.empty()){
        currentState=q.front();
        q.pop();
        if(currentState==0){
            stack<int> s;
            while(parent[currentState]){
                s.push(value[currentState]);
                currentState=parent[currentState];
            }
            s.push(1);
            while(!s.empty()){
                printf("%d",s.top());
                s.pop();
            }
            printf("\n");
            break;
        }
        temp=(currentState*10)%n;

        if(parent[temp]==-1){
            q.push(temp);
            parent[temp]=currentState;
            value[temp]=0;
        }
        temp=currentState*10+1;
        temp%=n;
        if(parent[temp]==-1){
            q.push(temp);
            parent[temp]=currentState;
            value[temp]=1;
        }
    }
}

int main(){
    int t,n;
    scanf("%d",&t);
    while(t--){
        while(!q.empty()){

```

```

        q.pop();
    }
    REP(i,20000)
        parent[i]=-1;

    scanf("%d",&n);
    solve(n);
}
}

```

#### V- Evaluation (test d'acceptation sur sphere on ligne judge)

Nous avons évalué notre code sur sphere on ligne judge proposé dans le cadre de cette études les images suivants sont les résultats obtenus. Nous test d'acceptation est encadré en noir.

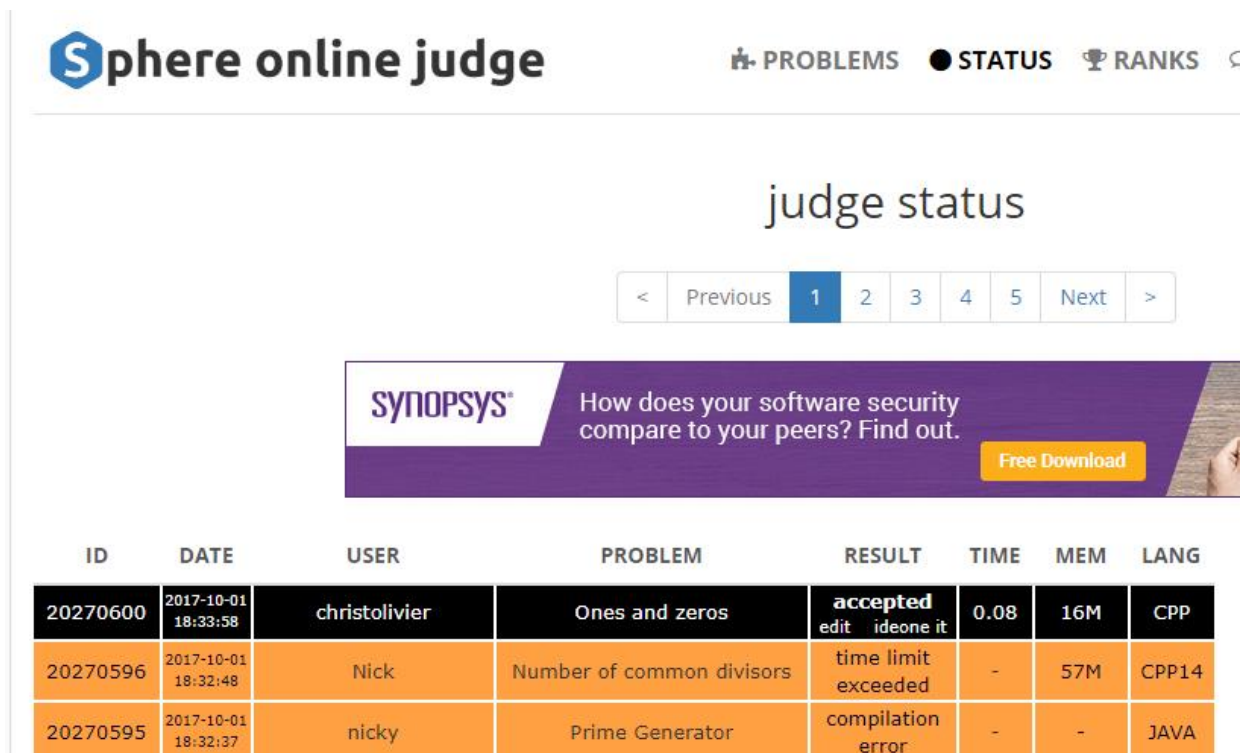


Figure 2: deuxième présentation du resultat

20270607	2017-10-01 18:35:42	arjun8115	Traversing Grid	accepted	0.01	15M	CPP
20270606	2017-10-01 18:34:58	Nurlita	DNA Sequences	accepted	0.46	6.5M	C++ 4.3.2
20270605	2017-10-01 18:34:44	gmannepalli	Marbles	accepted	0.00	16M	CPP
20270603	2017-10-01 18:34:21	chanquade123	ABC Path	wrong answer	0.00	2.7M	C++ 4.3.2
20270601	2017-10-01 18:34:10	varun6325	Chop Ahoy! Revisited!	wrong answer	0.00	15M	CPP14
20270600	2017-10-01 18:33:58	christolivier	Ones and zeros	accepted edit ideone it	0.08	16M	CPP
20270596	2017-10-01 18:32:48	Nick	Number of common divisors	time limit exceeded	-	57M	CPP14
20270595	2017-10-01 18:32:37	nicky	Prime Generator	compilation error	-	-	JAVA
20270591	2017-10-01 18:32:08	Ashish Lavania	Greatest Common Divisor Of Three Integers	accepted	0.00	2.7M	C++ 4.3.2
20270589	2017-10-01 18:32:00	Rajershi gahol(epsilon123)	Balika Vadhu and Alok Nath	accepted	0.03	25M	CPP14

Figure 3: deuxième présentation du resultat

## VI- Expérimentation sur notre algorithme

Dans cette partie nous allons effectuer nos propres test avec 20 cas d'entrées et nous allons stocker les résultats obtenus dans un tableau .

Nombre de test	INPUT	OUTPUT
1	3 17	11101 11101 11101
2	2 10	10 10
3	2 50	100 100
4	2 2015	1110011110 1110011110
5	4 89	11010101 11010101 11010101

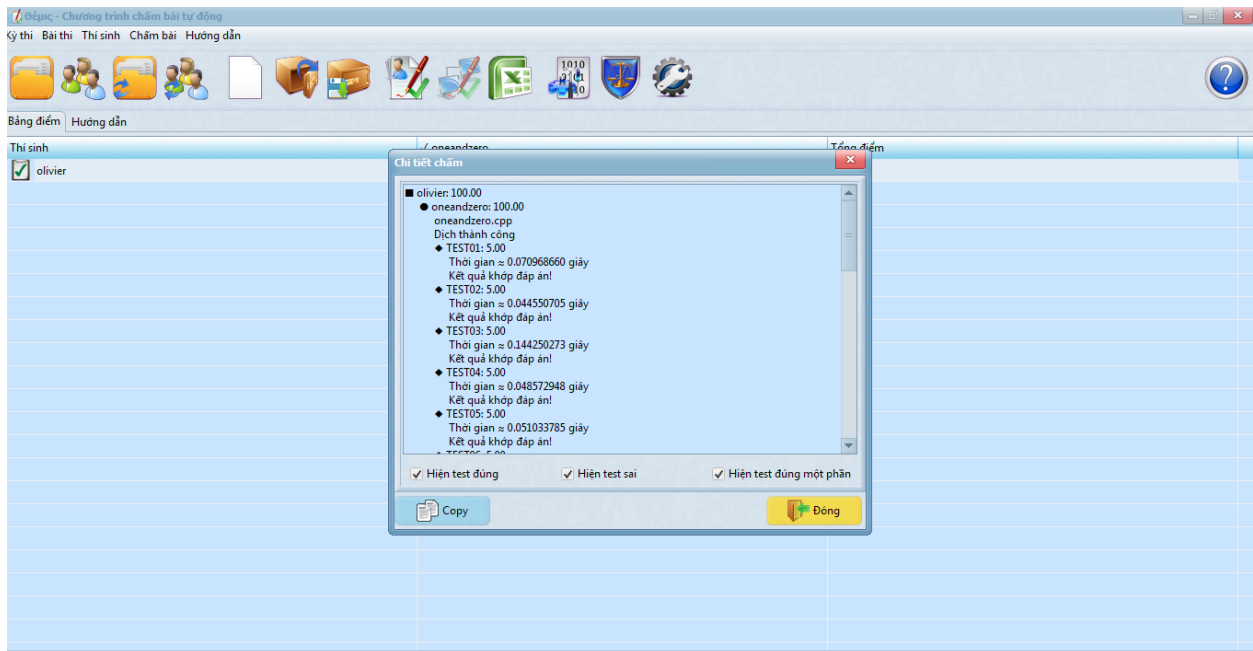
		11010101
5	3 67	1101011 1101011 1101011
7	3 97	11100001 11100001 11100001
8	5 396	111111111111111100 111111111111111100 111111111111111100 111111111111111100 111111111111111100
9	6 1	10 10 10 10 10 10
10	3 2000	10000 10000 10000
11	2 1000000	No Output
12	5 4587	111000001101 111000001101 111000001101 111000001101 111000001101
	3 1110	1110

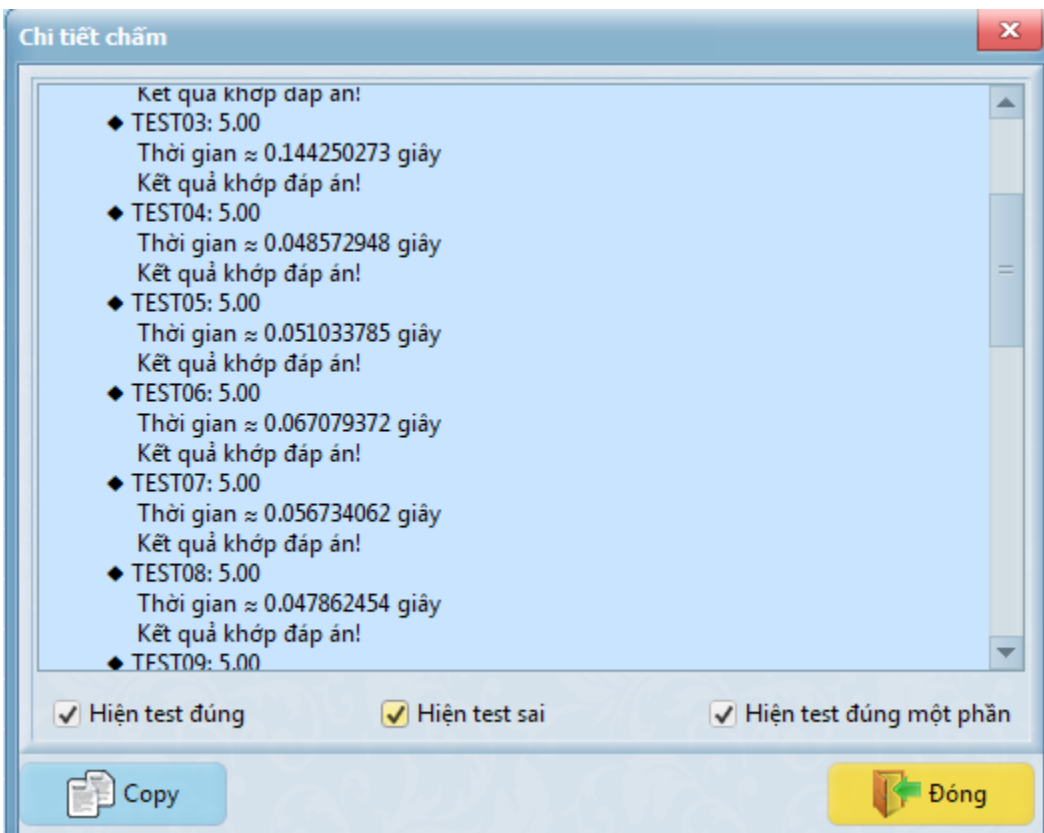
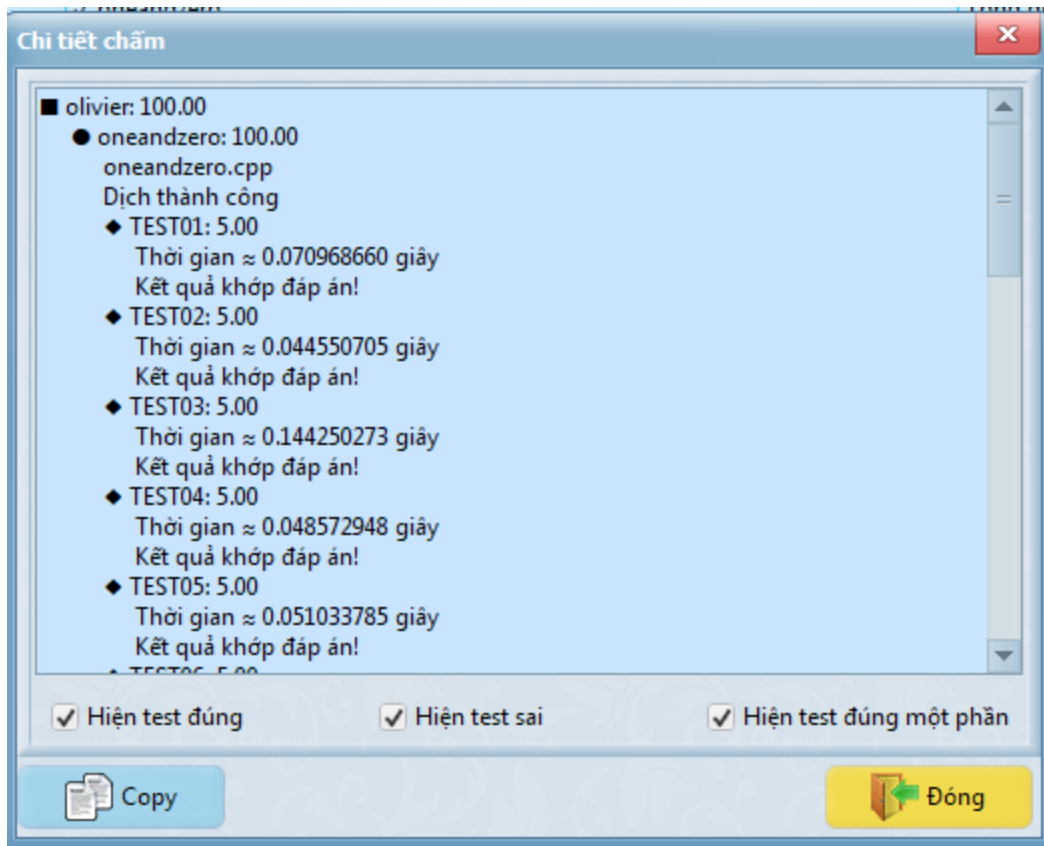
13	58	11011010 11011010
14	3 1101 101	1101 101 101
15	4 22 1111	110 1111 1111 1111
16	5 687 111 44	11110111101 111 1100 1100 1100
17	7 101 110 111	101 110 111 111 111 111 111
18	3 11 10011 87 69547	11 10011 11010111
19	1 2000 698 1111	10000
20	8 789542 1478	No Output

	111 25647	
--	--------------	--

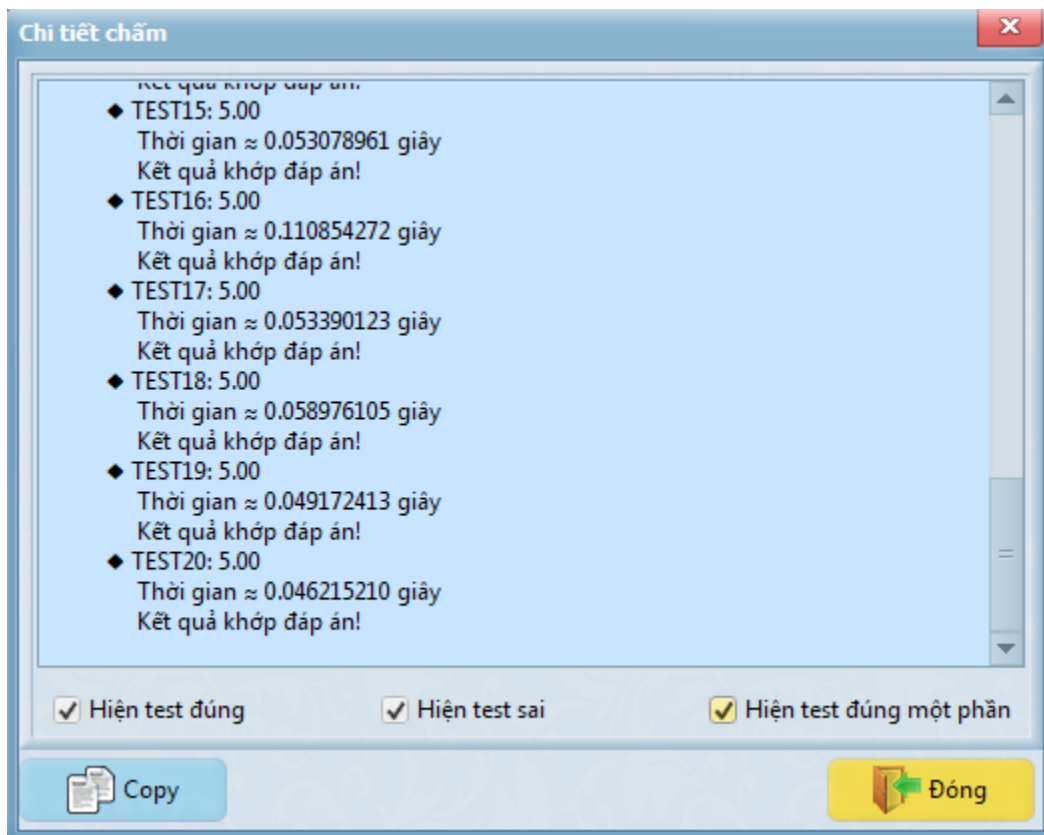
## VII- Evaluation du temps de calcul avec themis

Nous lançons notre programme pour le test. Nous utilisons l'outil themis a cet effet, nous avons utiliser cette outils pour avoir le temps dexécution de notre problème sur 20 cas de test. Les images ci-dessous representent les resultats obtenus.










# **PREMIERE PARTIE : PROBLEME DIGIT JUMP**


## PARTIE II : L'APPROCHE DIGIT JUMP

### I- Description complète du problème

Étant donné une séquence  $S$  de longueur  $N$ . Nous devons passer du début de la séquence (index 0) à la fin de la séquence (index  $N - 1$ ). De la position  $i$ , nous devons pouvoir passer à la position suivante ( $i + 1$ ) ou précédente ( $i - 1$ ). Nous pouvons également passer de la position actuelle aux indices où le chiffre est identique à celui du chiffre actuel  $S[i]$ .

Exemple :

 Input: 01234567890  
Output: 1

 Input: 012134444444443  
Output: 4

### II- Exemples d'application du problème dans la vie réelle

Le sujet auquel nous sommes soumis est un problème de mise en correspondance entre deux éléments d'une même séquence. Il s'agit donc des instructions de branchement. Une instruction de branchement peut être définie comme un point dans une séquence d'instructions d'un programme pour lequel l'instruction suivante n'est pas forcément celle qui suit dans la mémoire d'instructions.

Ce sujet peut donc être appliqué dans les techniques de branchement conditionnel que inconditionnel sur les processus de traitement de données dans un processeur.

### III- Écrire brièvement la connaissance de base de la méthode utilisée pour résoudre le problème

Dans ce problème, nous pouvons considérer les opérations de mouvement d'une position à l'autre comme des bords d'un graphe et des indices de la séquence en tant que des nœuds. Trouver un nombre minimum d'opérations pour passer de 0 à  $N - 1$  revient à déterminer le chemin le plus court dans le graphe formé.

Afin de pouvoir résoudre ce problème, il est nécessaire d'avoir des notions sur les algorithmes de parcours de graphes. Dans ce problème, il est question de trouver un plus court chemin permettant d'atteindre le sommet de la séquence soit en effectuant le nombre minimal de sauts, donnant donc la notion du plus court chemin. Dans l'ensemble des algorithmes connus, celui de Dijkstra est le plus souvent utilisé pour la détermination de plus court chemin. En plus de l'algorithme de Dijkstra, nous avons le BFS (Breadth First Search) utilisé pour les parcours en largeur des graphes afin de résoudre les problèmes de recherche de plus court chemin dans les graphes non pondérés.

Cependant, l'algorithme de Dijkstra est plus utilisé pour les graphes pondérés à valeur différentes. Nous pouvons donc utiliser le BFS au détriment de Dijkstra dans ce cas car nous avons des arrêtes de poids unitaires.

Nous pouvons appliquer donc un BFS du nœud de départ (index 0) vers le nœud final (index  $n - 1$ ). Le nombre de nœuds dans le graphe étant  $N$ .

#### IV- Écrire l'algorithme en détail pour résoudre le problème et la preuve de l'exactitude de l'algorithme

Dans cette partie nous présentons l'algorithme détaillé que nous avons choisi pour la résolution du problème.

\*\*\*\*\*Algorithme détaillé\*\*\*\*\*

*Début*

*Initialiser la queue avec le premier élément de la séquence S*

*Tant que la queue n'est pas vide:*

*Supprimer la valeur courante de la queue et la stocker*

*Enregistrer chaque déplacement (chaque suppression de nœud)*

*Si le nombre de nœud en de-queue est égale à la longueur de la séquence,*

*Alors, arrêter*

*Ajouter le nombre de déplacement pour chaque de-queue*

*Si le nœud adjacent gauche est valide et non visité, alors*

*Visité ce nœud*

*Ajouter ce nœud dans la queue*

*Mettre à jour le nombre de mouvement*

*Si le nœud adjacent droit est valide et non visité*

*Visité ce nœud*

*Ajouter ce nœud dans la queue*

*Mettre à jour le nombre de mouvement*

*S'il existe un nœud absent dans la liste des nœuds en de-queue*

*Pour tout nœud dans le graphe*

*Si un nœud n'est pas visité, alors Visiter*

*Visité ce nœud*

*Ajouter ce nœud dans la queue*

*Mettre à jour le nombre de mouvement*

*FIN Tant que*

## Preuve

i / p: 01213443

étapes:

(1) file d'attente: 0 (0)

(2) file: 1 (1) // dequeue 0 et enqueue 1.

(3) file: 2 (2), 1 (2) // dequeue 1 et enqueue 2 et 1.

(4) file: 1 (2) // dequeue 2 (pas de nœud non visité à partir de 2).

(5) file: 3 (3) // dequeue 1 et enqueue 3.

(6) file: 4 (4), 3 (4) // dequeue 3 et enqueue 4 et 3.

Nous avons atteint le dernier chiffre, donc la réponse est 4.

### V-) Analyse de la complexité du problème

Suite à l'analyse de notre algorithme, nous avons remarqué que le nombre minimum d'opérations ne peut être supérieur à 19.

Preuve:

Nous pouvons commencer à partir de la première position et aller directement à l'index ayant la même valeur que le caractère courant.

Ensuite, de cette position, passez à la position suivante et continuez à répéter l'étape précédente. Par conséquent, nous pouvons donc effectuer 19 mouvements au maximum, car le premier chiffre sera visité une fois. Le cas pratique dans cette situation est la suivante: 001122334455667788999.

Dans l'ensemble des mouvements possible, chaque chiffre ne sera visité plus de deux fois.

Preuve:

Le cas où le nombre de mouvements pour aller d'un chiffre à l'autre est supérieur à deux, il est possible de les réduire tout simplement à deux en passant de la position de départ à la finale. Ainsi, nous conservons au maximum 2 mouvements pour passer d'un chiffre à l'autre à savoir le mouvement gauche gauche (i-1) et droite (i+1).

Nous déduisons donc une complexité  $O(20 * N)$ . Ici 20 est le nombre maximum d'itérations. Soit une complexité égale à  $O(N)$ .

### VI-) Implémentation

L'implémentation de cet algorithme est inspirée de la solution proposée sur le site du problème [6].

### VII-) Analyser du pseudo code

```
//Parcours dfs
while(!q.isEmpty())
{
    //vérifier le sommet
    int u = q.remove();
    moves = q.remove();
    int v = sequence[u]-'0';
    //verifier si la taille de la séquence est atteinte
```

```

    if(u == length-1)
    {
        minJump.append(moves).append("\n");
        break;
    }
    moves++;
    //si le sommet adjacent est valide et non visité
    if(u > 0 && !noeudVisite[u-1])
    {
        //marquer comme visité
        noeudVisite[u-1] = true;
        q.add(u-1); //ajouter la valeur adjacente à la queue
        q.add(moves);
    }
    //si le sommet adjacent est valide et non visité
    if(u < length-1 && !noeudVisite[u+1])
    {
        //marquer comme visité
        noeudVisite[u+1] = true;
        q.add(u+1);
        q.add(moves);
    }

    if(!red[v])
    {
        red[v] = true;
        //passer la liste d'adjacence des nœud
        for( int visitlist: graphe.get(v))
        {
            //si un noeud est deja visité, il n'est pas prise en compte
            if(!noeudVisite[visitlist])
            {
                noeudVisite[visitlist] = true;
                q.add(visitlist);
                q.add(moves);
            }
        }
    }
}

```

## VIII-) Capture de l'évaluation sur le site CodeChef.com

Home » Practice(easy) » Chef and Digit Jumps » kaf » Submissions

**rdius07's SUBMISSIONS FOR DIGJUMP**

Lang : All ▼    Result : All ▼    GO

ID	Date/Time	User	Result	Time	Mem	Lang	Solution
15589638	1 min ago	rdius07	✓	0.11	4284M	JAVA	<span>View</span>

Illustration 1: Capture de validation

## IX-) Présentation des résultats issus des différents tests

Les tests suivants ont été effectué sur l'IDE Java en ligne de geeksforgeeks. Le lien est le suivant: <http://ide.geeksforgeeks.org/CSfUBR>

Input	Output	Durée (sec)
0123456789	9	0,06
00112233445566778899	19	0,06
001125698753265422892148	5	0,06
1126585212120424887785212547787865442	3	0,06
1259387125697542332892210000000054522656	4	0,07
125888798965678966398775668886554579566561	1	0,06
45698235488633585254569886354757444444444	1	0,06
58879353322544212358997785422536878526578	2	0,04
11111125555555588888877799999996666663	13	0,06
6666669999999888888555555444444478888	5	0,06
3333333333333254888887985444444412698889	6	0,06
48999999999552000000369752148550528789652	4	0,06
158889852126697753002145896575787754542215	2	0,07
87598585857285269324586221639825878957569	3	0,06
875985222487989652323236988445542211599877	2	0,07

78995658224110326699998845744225486522885	3	0,06
012563210245621202556325201248785866965423	6	0,06
657745658895422325556555448899665223355664		
778899665544225666332211447788995661256897		
6666666669999998888888888777777755555	9	0,06
111111111122222233333355555555777777888	11	0,07

Tableau1 : Résultat de tests

Le tableau ci-dessus indique les différentes entrées et sorties avec le temps de calcul de chaque opération. Nous remarquons que le temps mis varie en fonction de chaque entrée. Le plus petit étant de 0,04 et le plus élevé est de 0,07. La capture suivante représente une vue d'un exemple d'entrée-sortie.

The screenshot shows a web application interface with the following elements:

- Input Field:** Contains a long alphanumeric string: 012563210245621202556325201248785866965423657745658895422325556555448899665223355664778899665544225666332211447788995661256897. There is a 'Copy' button next to it.
- Buttons:** 'Run' and 'Run+URL (Generates URL as well)'.
- Generated URL:** A text box showing 'http://ide.geeksforgeeks.org/CSFUBR' with a 'Copy' button.
- Time(sec) :** 0.06
- Memory(MB) :** 53.4609
- Output:** A text box showing the number '6' with a 'Copy' button.

*Illustration 2: Capture de entrée-sortie*

## CONCLUSION GENERALE

Au cours de ce TD nous avons étudié deux algorithmes qui se basent sur l'algorithme BFS. Il s'agissait de l'algorithme ONEZERO et DIGIT JUMP. Dans chacun de ces algorithmes, nous avons implémenté un programme qui résout les différents problèmes posés. Puis nous avons fait un test d'acceptation en ligne des différents programmes. Nous avons effectué nos différentes expérimentations et avons analysé les différents résultats qui ont été stockés dans des tableaux. Ce TD nous a permis d'approfondir nos recherches sur l'algorithme BFS.



## REFERENCES

- [1] : <http://www.geeksforgeeks.org/breadth-first-traversal-for-a-graph/>
- [2] : <http://www.spoj.com/problems/ONEZERO/>
- [3] : <https://www.codechef.com/problems/DIGJUMP>
- [4] : <https://codinghangover.wordpress.com/2015/05/01/spoj-onezero-ones-and-zeros/>
- [5] : <http://www.mediafire.com/download/6gcicnduut7273r/ThemisInstaller.exe>
- [6] : <https://www.codechef.com/viewsolution/9052772>