

Institut Francophone International

COMPLEXITÉ ET ALGORITHMES

Rapport de travail domicile

**Sujet : “La solution du problème de mobile équilibré
utilisant l’arbre binaire”**

Enseignant	: Prof. DO Phan Thuan
Étudiante	: DAO Thuy Hong
Promotion	: 20

SOMMAIRE.....	2
1. Introduction :.....	4
2. Description du problème :.....	4
3. Introduction de l'arbre binaire :.....	6
4. Algorithme du problème et mis en place :	8
4.1 Traitement de l'entrée :.....	8
4.2 Parcours l'arbre binaire :	10
4.3 Complexité :	10
5. Résultat :.....	11
5.1 Évaluation sur "UVa Online Judge" :.....	11
5.2 Évaluation dans le logiciel Themis :.....	12

TABLEAU DE FIGURE

Figure 1 : Un mobile simple.....	4
Figure 2 : Un mobile complexe.....	5
Figure 3 : L'exemple de l'entrée.....	6
Figure 4 : Un arbre binaire.....	7
Figure 5 : Un arbre binaire binaire stocké utilisant le pointeur.....	7
Figure 6 : La structure du noeud.....	8
Figure 7: L'initiation de noeud de racine.....	8
Figure 8 : La procédure récursive pour créer un arbre binaire.....	9
Figure 9 : Le parcours de l'arbre binaire.....	10
Figure 10 : Le résultat sur https://uva.onlinejudge.org	11
Figure 11 : Indiquer le chemin du répertoire "Tasks".....	12
Figure 12 : I indiquer le chemin du répertoire "Tasks" "Contestants".....	13
Figure 13: Le résultat.....	14

1. Introduction :

Dans le cadre du cours complexité et algorithmes, j'ai choisi la catégorie de graphe pour mon travail domicile. À partir de mon choix, le professeur m'a donné le problème "Not So Mobile" qui provient du site <https://uva.onlinejudge.org>.

Dans ce rapport, je vais présenter des travaux que j'ai fait et le résultat que j'ai obtenu. Le rapport comprend cinq parties principales. Dans la partie 2, je vais décrire mon problème et la forme d'entrées et sorties. La connaissance de base de la méthode utilisée pour résoudre le problème va être abordée dans la partie 3. La partie 4 va présenter l'algorithme en détail avec la complexité. La partie 5 va donner quelques parties de code importantes dans le programme et l'analyser. Le résultat qui obtient après de lancer le programme va être cité dans la partie 6.

2. Description du problème :

Le problème aborde le mobile d'équilibre. Un mobile est une structure qui comprend des cordes et des fils afin de pendre des choses. La figure suivante illustre un mobile simple. Il comprend un fil, pendu par une chaîne, avec un objet de chaque côté.

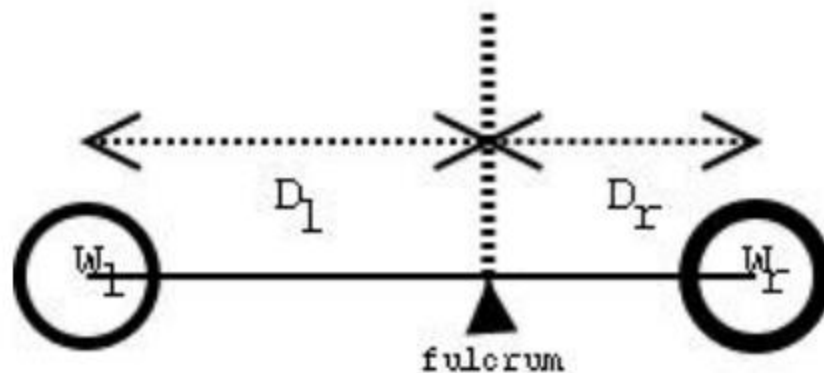


Figure 1 : Un mobile simple.

Un mobile simple est équilibré lorsque le produit du poids des objets par leur distance par rapport au point d'appui doit être égal. C'est-à-dire : $W_l \times D_l = W_r \times D_r$.

Dans un mobile plus complexe, un objet peut être remplacé par un sous-mobile, comme indiqué sur la figure suivante. Dans ce cas, il n'est pas simple pour vérifier si le mobile est équilibré ou non. Alors, on va écrire un programme qui lit la description d'un mobile d'entrée et puis vérifie l'équilibre de ce mobile.

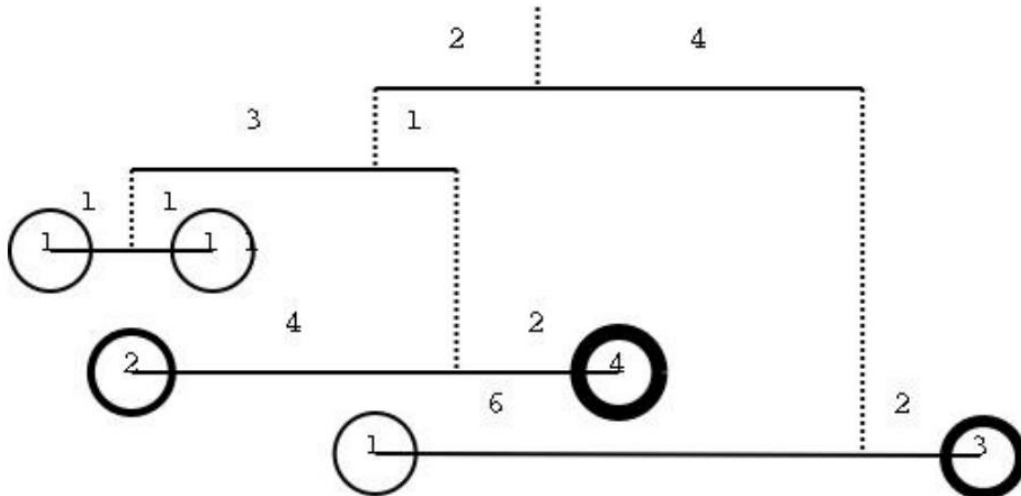


Figure 2 : Un mobile complexe.

Dans ce problème, l'entrée et la sortie sont en forme suivante. L'entrée est commencée par un nombre entier positif sur une ligne. Ce nombre indique le nombre de cas de test. L'entrée est composée de plusieurs lignes, chaque ligne contient 4 nombres entiers positifs en forme : $W_l D_l W_r D_r$.

Si W_l ou W_r est égal à zéro alors il y a une suspension de sous-mobile et les lignes suivantes indiquent le sous-mobile. Dans ce cas, le poids du sous-mobile est la somme des poids de tous ses objets. Si les deux W_l et W_r sont zéro, deux sous-mobiles sont cités par les lignes suivantes : d'abord la gauche et puis le droit.

1
0 2 0 4
0 3 0 1
1 1 1 1
2 4 4 2
1 6 3 2

Figure 3 : L'exemple de l'entrée.

Pour la sortie, pour chaque cas de test, le programme va écrire "YES" si le mobile est équilibré et écrire "NO" dans autre cas. Les sorties de deux cas consécutifs sont séparés par une ligne blanche.

3. Introduction de l'arbre binaire :

À partir de la description des entrées, si on considère chaque côté du système de mobile comme un nœud, on pourra utiliser l'arbre binaire afin de stocker le poids et la distance de chaque côté. Et puis, on va utiliser le parcours postfixe pour calculer le résultat. Dans cette partie, je vais présenter brièvement la connaissance de base de l'arbre binaire.

Un arbre binaire est un type spécial de la graphe. Un arbre binaire est un graphe connexe acyclique, tel que le degré de chaque nœud (ou vertex) soit au plus 3. La racine d'un arbre binaire est le nœud d'un graphe de degré maximum 2. Avec une racine ainsi choisie, chaque nœud aura un unique parent défini et deux fils. La figure au-dessous illustre un arbre binaire.

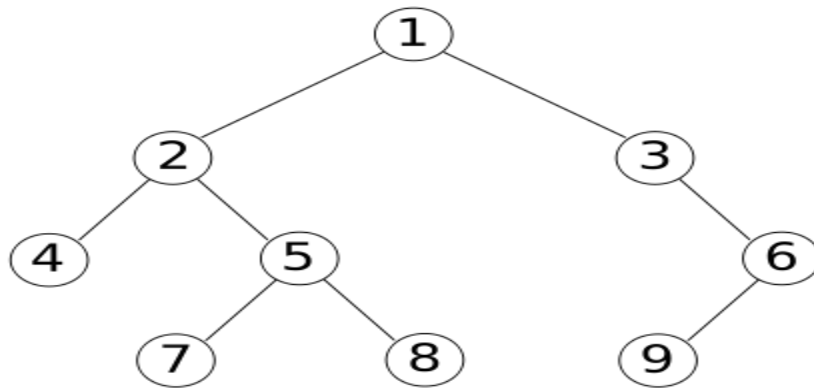


Figure 4 : Un arbre binaire.

Pour stocker des arbres binaires, on peut utiliser le pointeur ou le tableau. Ici, je s'intéresse seulement le pointeur, donc je vais présenter en détail le façon pour stocker un arbre binaire utilisant le pointeur. Vous pouvez consulter le ligne que je mets dans la partie de référence pour le tableau.

Dans l'arbore binaire stocké utilisant le pointeur, chaque nœud contient quelques données et pointeurs vers son fils droit et son fils gauche. Parfois, il contient également un pointeur vers son unique parent. Si un nœud possède moins de deux fils, l'un des deux pointeurs peut être affecté de la valeur spéciale nulle . La figure suivant présente un arbre binaire binaire stocké utilisant le pointeur.

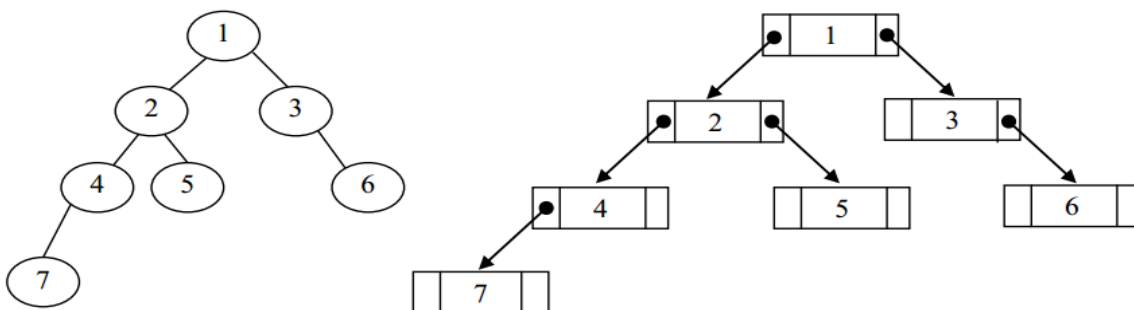


Figure 5 : Un arbre binaire binaire stocké utilisant le pointeur.

Il existe plusieurs ordres dans lesquels les nœuds peuvent être visités comme : parcours préfixe, infixé et postfixé; parcours en profondeur; parcours en largeur. Je appliquerai le parcours postfixé. Ce type de parcours sera décrit dans la partie suivante.

4. Algorithme du problème et mis en place :

Dans cette partie, je vais décrire l'algorithme que je applique et donner des parties de code de mon programme afin de inllustrer l'algorithme. Je utilise le langage C++ pour implémenter l'algorithme.

4.1 Traitement de l'entrée :

Comme j'ai déjà abordé au-dessus, je vais utiliser l'arbre binaire afin de stocker des données dans chaque cas de test. Dans cet arbre, chaque noeud comprend des informations suivantes : poids d'objet W, distance par rapport au point d'appui D, pointeurs qui citent le noeud gauche et le noeud de droit.

```
typedef struct node {  
    int w, d;  
    struct node *left, *right;  
} treenode;
```

Figure 6 : La structure du noeud.

Le processus de creation d'un arbre binaire aura lieu comme ça : d'abord, un noeud de racine est initié.

```
void init(treenode ** root){  
    treenode *tmp;  
    tmp = new treenode;  
    tmp->w = 0;  
    tmp->d = 0;  
    tmp->left = NULL;  
    tmp->right = NULL;  
    *root = tmp;  
    ok = true;  
}
```

Figure 7 : L'initiation de noeud de racine.

Avec les information dans chaque ligne dans l'entrée, on va créer deux noeuds : noeud gauche et noeud droit et attacher ces noeuds à le noeud de parent courant. Et puis, on va considérer s'il y a le sous-mobiles. Si le poids (W) de noeud est égal zéro, la ligne suivante va être lu pour les information de ce sous-mobile. Au niveau de codage, le processus est implémenté par une procédure récursive. Dans le langage C++, cette procédure est mis en place comme la figure suivante .

```
void make_tree(treenode **root){
    int w1, d1, w2, d2;

    //Lire l'entrée
    cin>>w1>>d1>>w2>>d2;

    treenode *tmp_left, *tmp_right;

    //Créer le noeud gauche
    tmp_left = new treenode;
    tmp_left->w = w1;
    tmp_left->d = d1;
    tmp_left->left = NULL;
    tmp_left->right = NULL;

    //Créer le noeud droit
    tmp_right = new treenode;
    tmp_right->w = w2;
    tmp_right->d = d2;
    tmp_right->left = NULL;
    tmp_right->right = NULL;

    //Attacher le noeud gauche et le noeud droit à le noeud courant
    (*root)->left = tmp_left;
    (*root)->right = tmp_right;

    //Si il y a le sous-mobile
    if(w1 == 0)
        make_tree(&tmp_left);
    if(w2 == 0)
        make_tree(&tmp_right);
}
```

Figure 8 : La procédure récursive pour créer un arbre binaire.

4.2 Parcours l'arbre binaire :

Pour déterminer si le mobile est équilibré ou non, je vais considérer l'équilibre du sous- mobile en premier. Des sous-mobiles vont être considéré de la gauche du droit, de bas en haut. Pour faire ça, je vais parcourir l'arbre binaire dans ordre postfixe. On doit calculer le somme des poids des objets dans chaque sous-mobile et le stocker.

Au niveau de codage, je utilise stack afin de réaliser ce processus. D'abord, on vérifie la valeur poids du noeud courant. Si cette valeur n'est pas égal zéro, on la mettre dans stack. Si non, on obtiendra deux valeurs dans stack. C'est les valeurs de noeud gauche et noeud droit du sous-mobile. On va tester l'équilibre de ce sous-mobile et calculer le somme des poids des objets dans ce sous-mobile et mettre ce somme dans stack. Le code de cette partie est comme la figure au-dessous.

```
stack<treenode> stack_tree;
//Parcours postfixe
void postOrder(treenode *root){
    if (root != NULL){
        postOrder(root->left);
        postOrder(root->right);
        //Si il n'y a pas le sous-mobile, stocker le noeud dans stack
        if(root->w != 0){
            stack_tree.push(*root);
        }else{
            //Si il y a le sous-mobile, considérer l'équilibre de sous-mobile
            treenode left, right;

            //Obtenir la valeur de noeud gauche dans le sous-mobile
            right = stack_tree.top();
            stack_tree.pop();

            //Obtenir la valeur de noeud droit dans le sous-mobile
            left = stack_tree.top();
            stack_tree.pop();

            //Mettre à jour la valeur de noeud de parent
            root->w = left.w + right.w;
            stack_tree.push(*root);

            //Considérer l'équilibre de sous-mobile
            if((left.w * left.d) != (right.w * right.d)) ok = false;
        }
    }
}
```

Figure 9 : Le parcours de l'arbre binaire.

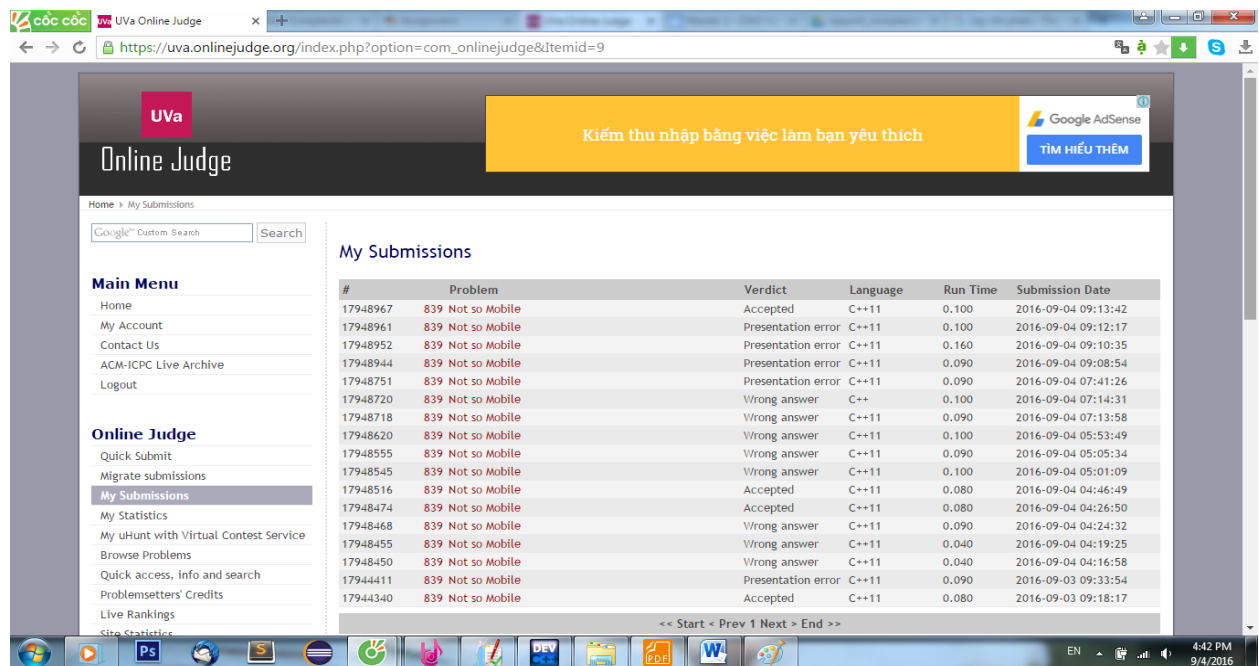
4.3 Complexité :

Dans tous les cas, l'opération de parcourir l'arbre (postOrder) de la racine jusqu'à une feuille : le temps d'exécution est donc proportionnel à la profondeur de l'arbre qui vaut n dans le pire des cas, d'où une complexité maximale en $O(n)$ (n est la quantité de noeud). À l'égard de l'opération "make_tree", la complexité est aussi comme l'opération "postOrder". Alors, la complexité de l'algorithme est $O(n)$.

5. Résultat :

5.1 Évaluation sur "UVa Online Judge" :

"UVa Online Judge" (<https://uva.onlinejudge.org>) est un site qui fournit des centaines de problèmes. Nous pouvons soumettre nos sources dans une variété de langues, en essayant de résoudre l'un des problèmes disponibles sur ce site. Mon problème est un des problèmes dans ce site, donc j'ai soumis mon programme sur <https://uva.onlinejudge.org> afin de l'évaluer et il est accepté. Le résultat comme suivant :



#	Problem	Verdict	Language	Run Time	Submission Date
17948967	839 Not so Mobile	Accepted	C++11	0.100	2016-09-04 09:13:42
17948961	839 Not so Mobile	Presentation error	C++11	0.100	2016-09-04 09:12:17
17948952	839 Not so Mobile	Presentation error	C++11	0.160	2016-09-04 09:10:35
17948944	839 Not so Mobile	Presentation error	C++11	0.090	2016-09-04 09:08:54
17948751	839 Not so Mobile	Presentation error	C++11	0.090	2016-09-04 07:41:26
17948720	839 Not so Mobile	Wrong answer	C++	0.100	2016-09-04 07:14:31
17948718	839 Not so Mobile	Wrong answer	C++11	0.090	2016-09-04 07:13:58
17948620	839 Not so Mobile	Wrong answer	C++11	0.100	2016-09-04 05:53:49
17948555	839 Not so Mobile	Wrong answer	C++11	0.090	2016-09-04 05:05:34
17948545	839 Not so Mobile	Wrong answer	C++11	0.100	2016-09-04 05:01:09
17948516	839 Not so Mobile	Accepted	C++11	0.080	2016-09-04 04:46:49
17948474	839 Not so Mobile	Accepted	C++11	0.080	2016-09-04 04:26:50
17948468	839 Not so Mobile	Wrong answer	C++11	0.090	2016-09-04 04:24:32
17948455	839 Not so Mobile	Wrong answer	C++11	0.040	2016-09-04 04:19:25
17948450	839 Not so Mobile	Wrong answer	C++11	0.040	2016-09-04 04:16:58
17944411	839 Not so Mobile	Presentation error	C++11	0.090	2016-09-03 09:33:54
17944340	839 Not so Mobile	Accepted	C++11	0.080	2016-09-03 09:18:17

Figure 10 : Le résultat sur <https://uva.onlinejudge.org>

5.2 Évaluation dans le logiciel Themis :

Après d'être accepté sur <https://uva.onlinejudge.org>, j'ai créé 20 cas de test pour évaluer mon programme dans Themis. Themis est un logiciel de correction automatique basé sur des cas de test. Les auteurs de ce logiciel sont le professeur Le Minh Hoang et le professeur Do Duc Dong.

Pour lancer ce logiciel, on doit créer les répertoires "Tasks" et "Contestants". Le répertoire "Tasks" contient des cas de test et le répertoire "Contestants" contient des sources de code. On appuie le bouton F2 pour indiquer le chemin du répertoire "Tasks" et le bouton F3 pour indiquer le chemin du répertoire "Tasks" "Contestants".

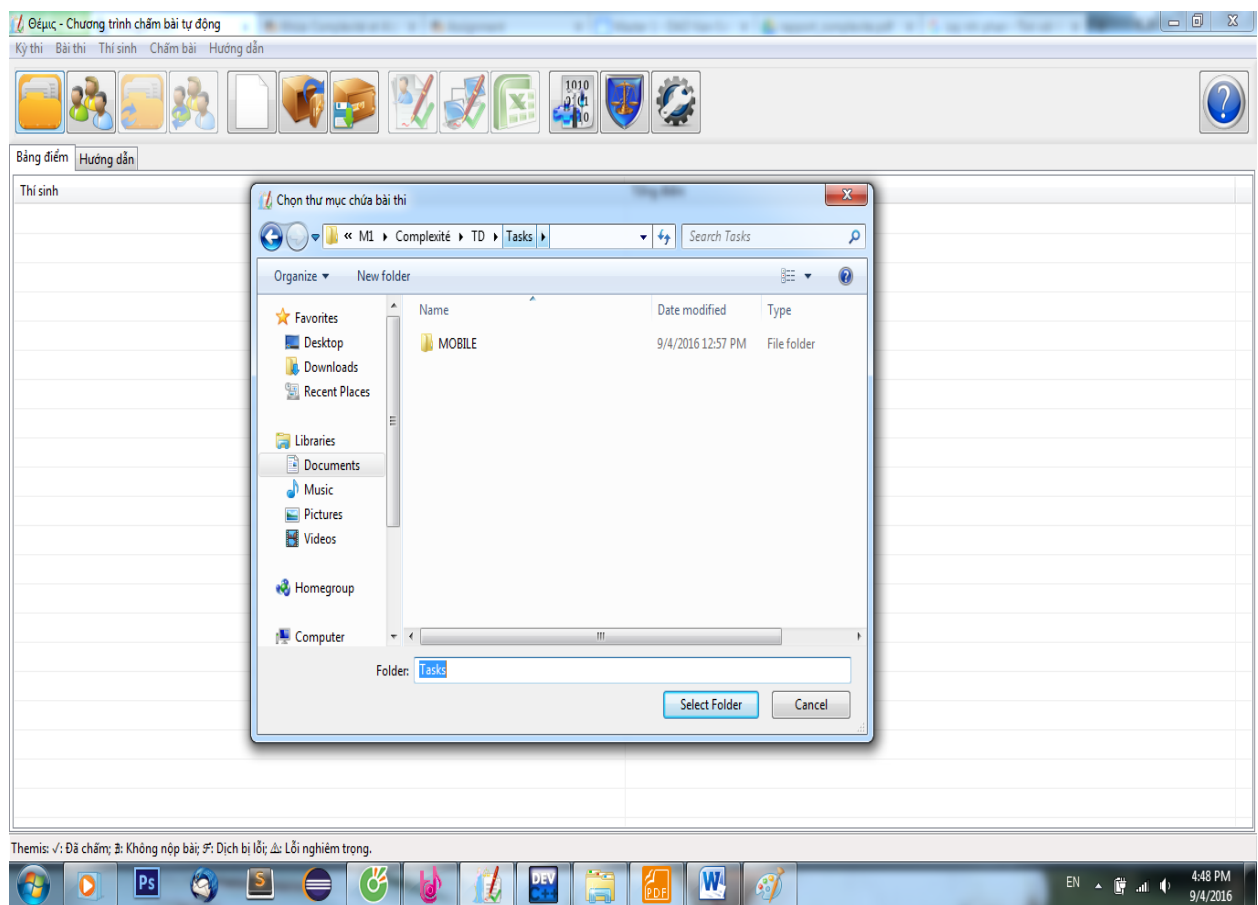


Figure 11 : Indiquer le chemin du répertoire "Tasks".

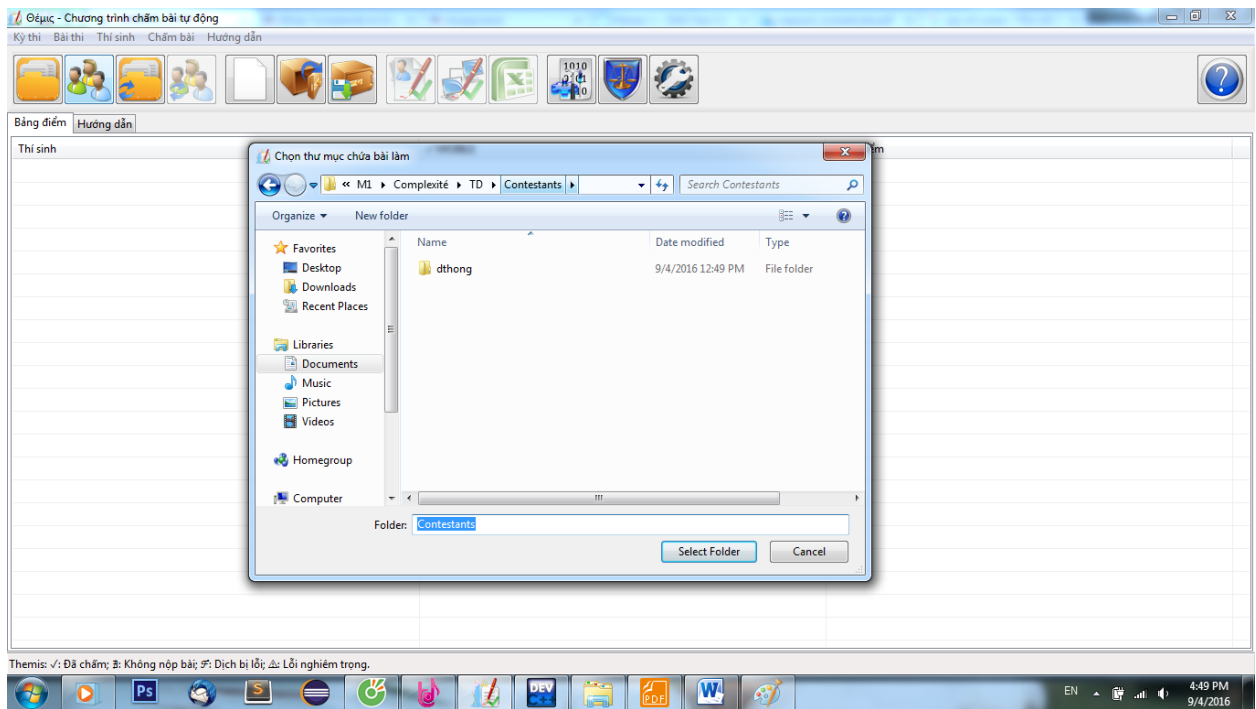


Figure 12 : J'indiquer le chemin du répertoire "Tasks" "Contestants".

Et puis, on doit appuyer le bouton F9 pour lancer la fonction de correction. Je configure la note que on va obtenir si le programme passe un cas de test est 1. Après de lancer sur Themis, je gagne 20, donc mon programme a passé tous les cas de test. Le résultat est illustré dans la figure 13.

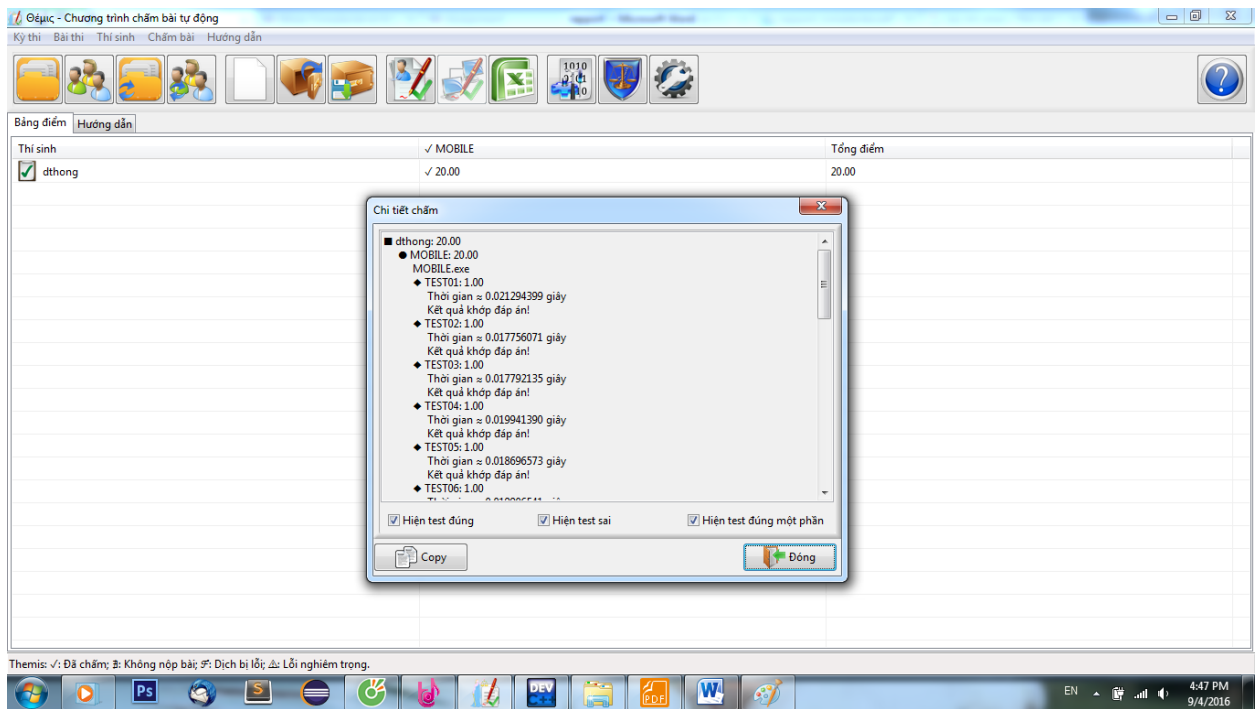


Figure 13 : Le résultat.

Référence :

- [1] https://vi.wikipedia.org/wiki/C%C3%A2y_t%C3%ACm_ki%E1%BA%BFm_nh%E1%BB%8B_ph%C3%A2n
- [2] https://fr.wikipedia.org/wiki/Arbre_binaire_de_recherche
- [3] https://uva.onlinejudge.org/index.php?option=com_frontpage&Itemid=1
- [4] <https://dsapblog.wordpress.com/2013/12/24/themis/>
- [5] Nguyễn Duy Phương - PTIT, Cấu trúc dữ liệu và giải thuật.
- [6] https://fr.wikipedia.org/wiki/Arbre_binaire