Rapport de projet de Complexité

Carac spéciaux :

∀ , ∃,

## TABLE DES MATIERES



## I. STRUCTURE DE DONNEES

Afin de réaliser ce projet, nous avons décidé de commencer en utilisant une implémentation matricielle pour les graphes, et une implémentation en liste chainée pour les ensembles de sommets.

Obtenant des résultats satisfaisants, nous avons décidé de conserver nos structures pour l'ensemble du projet.

```
#define n_max 1000

typedef int sommet;

typedef struct {
	Int a[n_max][n_max];
```

```
        int n;
        int degre[n_max];
} Graph_m;


typedef struct maillon{
        sommet st;
        struct maillon *suiv;
} maillon;


typedef maillon *liste;
```

# II. PARSING DES FICHIERS ET INITIALISATION DES STRUCTURES

➔ Symétrisation des graphes afin de les rendre non-orientés quelques soit la déclaration.

# III. VERIFICATION SOUS-GRAPHE DESERT

## A) PRINCIPE DE L'ALGORITHME :

L'algorithme a pour but de vérifier qu'un ensemble de sommets passé en paramètre est un sous-graphe désert.

Afin de vérifier le caractère « désert » de l'ensemble de sommets, on vérifie que chaque sommet n'est pas voisin d'un des autres sommets de l'ensemble. On évitera de faire des comparaisons inutiles en ne prenant que les sommets suivants dans l'ensemble.

## B) PSEUDO-CODE :

```
Verification_graphe_desert (Graph g, liste X) {
        Pour tout sommet x de X {
                Pour tout sommet x' de X-{x0,….,x,x'} {
                        Si x et x' sont adjacents, retourner faux.
                }
        }
        Retourner vrai.
}
```

### C) VALIDITE DE L'ALGORITHME :

L'algorithme est valide puisque l'on crée tous les couples (x , y) possibles en assemblant deux sommets de X, et que pour chacun de ces couples, on vérifie si x possède un arc vers y (et inversement). On vérifie ainsi qu'aucun des sommets de l'ensemble n'est voisin avec un quelconque autre sommet de l'ensemble.

### D) COMPLEXITE :

Soient n le nombre de sommet du graphe G = (S, A), et n' le nombre de sommets de l'ensemble X donnée en entrée.

Dans le pire des cas, n' = n et X est désert : c'est le cas lorsque G est un nuage de points et que l'on choisit X = S.
On parcourt alors l'ensemble X, et pour chaque sommet, on parcourt le reste de l'ensemble, donc en prenant 1 sommet de moins à chaque itération, soit :

$$\sum_{i=0}^{n} i = \frac{n(n-1)}{2}$$

Pour chaque itération, on effectue un nombre linéaire d'opérations fondamentales (ici deux comparaisons, puis si le test est positif, une affectation).
D'où une complexité en

$$\theta(n^2).$$

## IV. VERIFICATION DE LA MAXIMALITE

## A) PRINCIPE DE L'ALGORITHME :

L'algorithme a pour but de vérifier si l'ensemble passé en paramètre est un sous-graphe désert maximal, c'est-à-dire qu'il n'existe pas d'autre sous-graphe désert dans lequel il serait strictement inclus, pour le graphe passé en paramètre.

Afin de traiter la question, nous reformulons la propriété de maximalité :

> *Un sous-graphe désert G' = (S', A') de G=(S, A) est maximal s'il n'existe pas de sommet $x \in S$, $x \notin S'$, tel que x n'est adjacent à aucun sommet $y \in S'$.*

Ainsi, nous pouvons répondre en problème en parcourant l'ensemble de sommets X passé en paramètre, et en marquant tous les sommets appartenant à X ou étant adjacent à l'un d'eux.
A la fin du parcours, s'il reste au moins un sommet non-marqué, alors on pourrait ajouter ce sommet à X et créer un sous-graphe désert X' dans lequel serait inclus X. X ne serait alors pas maximal. Dans le cas contraire, il ne reste aucun sommet à ajouter à X, du moins sans casser le caractère « désert » du sous-graphe G'= (X, A').

## B) PSEUDO-CODE :

Le marquage des sommets se fera par le biais d'un tableau de booléens de taille n_max :

```
Verification_maximalité (Graph g, liste X) {

    Si X n'est pas un sous-graphe désert, retourner faux.
    Crée tableau_adjancence[n_max].
    Initialiser toutes les cases de tableau_adjacence à 1.
    Pour tout sommet x de X {
        tableau_adjacence[x] = 0.
        Pour tout sommet y de g {
            Si x et y sont adjacents, tableau_adjacence[y] = 1.
        }
    }
    Pour tout entier i de 0 à la taille du graphe g {
        Si tableau_adjacence[i] == 1, retourner faux.
    }
    Retourner vrai.

}
```

### C) VALIDITE DE L'ALGORITHME :

Pour le problème donné, il n'existe que trois cas possibles. Si l'algorithme répond correctement à chaque cas, il sera valide :
- X n'est pas un sous-graphe désert : l'algorithme répond que X n'est pas désert maximal, puisque non-désert.
- X est désert mais pas maximal : l'algorithme parcourt le graphe G et marque les différents sommets associés directement ou indirectement à X. A la fin du parcours, il détecte les sommets appartenant à l'ensemble X' maximal incluant X, et répond que X n'est pas maximal.
- X est maximal : l'algorithme parcourt le graphe G, marque les sommets. A la fin du parcours, tous les sommets sont marqués, l'algorithme répond que X est donc maximal.

Finalement, l'algorithme répondant correctement aux différents cas auxquels il peut être confronté, il est valide.

### D) COMPLEXITE :

Soient n le nombre de sommet du graphe G = (S, A), et n' le nombre de sommets de l'ensemble X donnée en entrée.

Dans le pire des cas, n' = n et X est maximal. : c'est le cas lorsque G est un nuage de points et que l'on choisit X = S.
On parcourt alors l'ensemble X, et pour chaque sommet, on parcourt le graphe à la recherche de ses adjacents. L'implémentation de nos graphes étant matricielle, la recherche des adjacents est une fonction linéaire en le nombre de sommets du graphe G (parcours d'une colonne/ligne de la matrice d'adjacence). On fait ainsi $n^2$ itérations de boucles, dans lesquelles on fait un nombre constant d'opérations fondamentales.
On parcourt enfin le tableau d'adjacence, de taille n, ce qui se fait en temps linéaire en fonction de n.

La complexité de cet algorithme de vérification de la maximalité d'un ensemble X dans un graphe G est donc en

$$\theta(n^2).$$

## V. ALGORITHME DE RECHERCHE D'UN SOUS-GRAPHE DESERT MAXIMAL

### A) PRINCIPE DE L'ALGORITHME :

### B) PSEUDO-CODE :

C) VALIDITE DE L'ALGORITHME :

D) COMPLEXITE :

## VI. ALGORITHME DE RECHERCHE INEXACTE D'UN SOUS-GRAPHE DESERT MAXIMUM

A) PRINCIPE DE L'ALGORITHME :

B) PSEUDO-CODE :

C) VALIDITE DE L'ALGORITHME :

D) COMPLEXITE :

## VII. ALGORITHME DE RECHERCHE EXACTE D'UN SOUS-GRAPHE DESERT MAXIMUM

A) PRINCIPE DE L'ALGORITHME :

B) PSEUDO-CODE :

C) VALIDITE DE L'ALGORITHME :

## D) COMPLEXITE :