Parcours de graphes

Partie 4 : Complexité des algorithmes présentés

Alain Cournier

Grandes lignes

- Définir la complexité
- Taille de la donnée ?
- À quoi sert-il de faire ces calculs ?
- Complexité en espace de l'algorithme de parcours générique
- Complexité en temps de l'algorithme de parcours générique
- Complexités des autres algorithmes

Définir la complexité

- Calculer une complexité c'est définir l'application qui lie la taille de la donnée fournie à notre algorithme avec la quantité d'utilisation d'une ressource critique (mémoire, processeur, bande passante, nombre de messages échangés…).
  - Ce calcul ne peut pas être précis on calcule l'ordre de grandeur d'une borne inférieure (Ω), d'une borne supérieure (O) ou un équivalent (Θ), lorsque la taille de la donnée devient très grandes (tend vers ∞).
  - La complexité se calcule dans le pire des cas ou en moyenne.
  - Dans notre cas, nous regarderons la complexité en espace (mémoire) et en temps (nombre d'opérations) dans le pire des cas, puisque les algorithmes sont conçus pour tourner sur un ordinateur isolé.

La taille de la donnée

- On compte comme la taille de la donnée, le nombre d'informations nécessaires pour décrire l'objet par un codage. Voici quelques exemple de taille :
  - Un entier k : le nombre de bits nécessaires pour écrire cet entier k c'est-à-dire $\log_2(k+1)$ arrondi à l'entier supérieur (en base 10 remplacer 2 par 10)
  - Un tableau T : Le nombre de cases qui le compose
  - Une Liste L : Le nombre d'élément de la liste
- Et pour un graphe ?

Taille de la donnée : pour un graphe

- Il faut une information par sommet,
- Il faut une information par arc.
- Si n est le nombre de sommets et m le nombre d'arcs, la taille d'un graphe est n + m.
- Toutefois compte tenu de la variabilité du nombre d'arc dans un graphe, on compte :
  - n^2 si le graphe est représenté par une matrice
  - n + m si le graphe est représenté par listes d'adjacences

A quoi sert-il de faire ces calculs de complexité ?

- Tout d'abord c'est une mesure que l'on peut associer à un algorithme
- Elle permet de se questionner sur les choix que nous avons fait :
  - Dans la façon dont nous avons écrit le programme
  - Dans le choix des structures de données
- D'avoir une idée des ressources nécessaire pour exécuter l'algorithme
- De comparer deux algorithmes afin de choisir le mieux adapté à l'environnement dans lequel il doit s'exécuter.

Complexité en espace de l'algorithme de parcours générique

- En fonction de la représentation :
  - $\Theta(n^2)$ pour la matrice $\Theta(n+m)$ pour une représentation par listes d'adjacences
  - $\Theta(n)$ pour notre ensemble Exploré
  - $\Theta(n)$ ou $\Theta(m)$ pour notre ensemble Atteint (en fonction de la représentation)

- Notre algorithme n'entraîne pas de surcoût en mémoire
- Il en est de même pour les deux autres parcours.

Complexité en temps de l'algorithme de parcours générique

- Sur notre ensemble Exploré
  - 1 initialisation
  - n insertions d'un élément (après chaque choix dans atteint)
  - m tests d'appartenance (à chaque découverte d'un arc dans le graphe)
- Si on utilise un tableau de booléen pour représenter cet ensemble :
  - Initialisation : $\Theta(n)$
  - n insertions : $\Theta(n)$; une opération par insertion
  - m tests : $\Theta(m)$; une opération par test d'appartenance

Complexité en temps de l'algorithme de parcours générique

- Sur notre ensemble Atteint
  - 1 initialisation par appel à visite graphe
  - m insertions d'un élément
  - n test ensemble vide
  - n choix
- Si on utilise un tableau de booléen pour représenter cet ensemble :
  - n initialisations : $\Theta(n^2)$; n opérations par initialisation (Pire des cas graphe sans arcs)
  - m insertions : $\Theta(m)$; une opération par insertion
  - n choix : $\Theta(n^2)$; n opérations par choix (idem pour tester ensemble vide)

Complexité en temps de l'algorithme de parcours générique

- Sur notre ensemble Atteint
  - 1 initialisation par appel à visite graphe
  - m insertions d'un élément
  - n choix et test ensemble vide
- Si on utilise une liste de sommet pour représenter cet ensemble :
  - n initialisations : Θ(n) ; 1 opérations par initialisation
  - m insertions : Θ(nm) ; n opérations par insertion Il faut vérifier que l'élément n'est pas déjà présent dans la liste
  - n choix : Θ(n) ; 1 opérations par choix (idem pour tester ensemble vide)

Complexité en temps de l'algorithme de parcours générique

- Exploration des successeurs de chaque sommet
  - n explorations ; 1 exploration par choix de sommets
  - Coût $\Theta(n^2)$ pour une représentation par matrice : chaque exploration des successeur d'un sommet x demande la visite complète de la $x^{ième}$ ligne de la matrice soit $\Theta(n)$ pour chaque exploration
  - Coût $\Theta(n+m)$ pour une représentation par listes d'adjacences : chaque exploration des successeurs d'un sommet x demande la visite d'une liste de sommets et coûte $\Theta(d^+(x))$ (i. e. le nombre de successeurs de x).

Complexité en temps de l'algorithme de parcours générique

- Bilan de complexité :
  - La complexité de notre algorithme est $\Theta(n^2)$ pour une représentation par matrice ou par listes d'adjacence si on représente les ensembles comme des tableaux de booléen.
  - Cette complexité est parfaite dans le cas d'une représentation par matrice car pour connaitre le graphe et donc l'explorer il faut lire chacune des n^2 cases de la matrice.
  - Par contre nous ne sommes pas capables pour l'instant d'atteindre une complexité en $\Theta(n+m)$ pour une représentation par listes d'adjacences.

Vers un algorithme en Θ(n+m)

- Pour atteindre cet objectif nous devons sur Atteint:
  - Réduire le coût de l'initialisation, Atteint devra être géré comme une liste
  - Faire en sorte que le choix se fasse en temps constant (choisir le premier)
  - Que le retrait de l'élément choisi se fasse en temps constant (Suite)
  - Que l'insertion d'un élément se fasse en temps constant (insertion en tête)
  - Par contre, nous ne pourrons pas tester l'appartenance à Atteint trop couteuse
  - Il faudra donc tester au moment du choix si l'élément est déjà dans exploré (auquel cas le traitement est déjà fait) ou non (il est à faire)

Algorithme de Base (Entête)

- Algorithme VisiteGraphe
  - Données :
    - G = (X,U) un graphe
    - x un sommet de G
  - Donnée/Résultat
    - Exploré : ensemble de sommets
  - Variables
    - Atteint : liste de sommets
    - u, v : Sommets de G

Algorithme de Base (Code)

- DébutCode
  - Atteint ← ListeVide()
  - Si non(x ϵ Exploré) alors Atteint ← InsereTete(x,Atteint); Finsi
  - Tant que Atteint ≠ ListeVide() faire
    - u ← Premier (Atteint); Atteint ← Suite(Atteint);
    - Si non(u ϵ Exploré) alors
      - Exploré ← Exploré U {u}; TraiterSommet (u);
      - Pour chaque v ϵ Succ(u) faire
        - TraiterArc(uv)
        - Atteint ← InsereTete(v,Atteint)
      - FinPour
    - Finsi
  - FinTQ
- FinCode

Rappel

- J’ai utilisé sur les listes les opérations suivantes :
  - ListeVide() : fonction qui renvoie la liste vide
  - Premier(L) : fonction qui renvoie le premier élément de la liste (s’il existe)
  - Suite(L) : fonction qui renvoie une liste égale à la liste L privée de son premier élément
  - InsereTete(x,L) : fonction qui renvoie une liste L’ telle que
    - Suite(L’) = L
    - Premier(L’) = x

Complexité en temps de l'algorithme de parcours générique

- Sur notre ensemble Exploré
  - 1 initialisation
  - n insertions d'un élément (après chaque choix dans atteint)
  - n+m tests d'appartenance (Test à l'initialisation de Atteint et après les choix)
- Si on utilise un tableau de booléen pour représenter cet ensemble :
  - Initialisation : $\Theta(n)$
  - n insertions : $\Theta(n)$; une opération par insertion
  - n+m tests : $\Theta(n+m)$; une opération par test d'appartenance

Complexité en temps de l'algorithme de parcours générique

- Sur notre ensemble Atteint
  - 1 initialisation par appel à visite graphe
  - n+m insertions d'un élément
  - n choix et test ensemble vide
- Si on utilise une liste de sommet pour représenter cet ensemble :
  - n initialisations : Θ(n) ; 1 opération par initialisation
  - m insertions : Θ(m) ; 1 opération par insertion
  - n choix : Θ(n) ; 1 opération par choix (idem pour tester si Atteint est vide)

Complexité en temps de l'algorithme de parcours générique

- Exploration des successeurs de chaque sommet
  - n explorations ; 1 exploration par choix de sommet
  - Coût $\Theta(n+m)$ pour une représentation par listes d'adjacences : chaque exploration des successeurs d'un sommet x demande la visite d'une liste de sommet et coûte $\Theta(d^+(x))$ (i. e. le nombre de successeurs de x).

Complexité en temps de l'algorithme de parcours générique

- Bilan de complexité :
  - Nous sommes maintenant capable d'atteindre une complexité en $\Theta(n+m)$ pour une représentation par listes d'adjacences.

- Il existe une structure de données particulière ensemble qui permet d'obtenir le même résultat mais je ne pense pas que vous l'ayez étudié pour l'instant.

Complexité des autres algorithmes

- Avec une représentation par matrice nous aurons pour les algorithmes de parcours en largeur et en profondeur des complexités de :
  - Complexité en espace : $\Theta(n^2)$
  - Complexité en temps : $\Theta(n^2)$
- Avec une représentation par listes d'adjacences nous aurons pour les algorithmes de parcours en largeur et en profondeur des complexités de :
  - Complexité en espace : $\Theta(n+m)$
  - Complexité en temps : $\Theta(n+m)$