Projet Chemin

Equipe

identifiant : G2S1

- Daniel Gilardoni
- Felipe Paris Mollo Christondis
- Leopold Abignoli

Instructions

1. Cet article contient une annexe, un projet développé par l'équipe, qui montre certains algorithmes mentionnés dans l'article. Pour accéder à cette annexe, voir le lien suivant: https://github.com/parismollo/MD5-Projet
2. La version originale de l'article a été écrite en "markdown", si vous voulez voir la version originale, au lieu de la version pdf, voir le lien suivant: https://github.com/parismollo/MD5-Projet/blob/main/Article.md

Introduction

L'objectif de ce projet est l'étude de chemins de taille minimum entre 2 points dans une grille.
Un point est situé à l'extremité d'une arête. La grille peut contenir des murs, ce sont des arêtes de la grille par lesquelles on ne peut pas passer.

Questions

1. Longueur et nombre de chemins de longueurs minimales dans une grille

La taille d'un chemin est le nombre d'arêtes par lequel on doit passer pour aller du premier point du chemin au dernier. Un point qui a pour coordonnée (x, y) correspond au point de la colonne x et de la ligne y.

A) Longueur du chemin minimal

Les coordonnées du point de départ sont (x_1, y_1) et (x_2, y_2) sont celles du point d'arrivée.
On passe par $|x_1 - x_2|$ arêtes en ligne puis par $|y_1 - y_2|$ en colonne, depuis le point de depart pour atteindre le point d'arrivée.
Un chemin minimal dans une grille sans mur, est donc de longueur :
$|x_1 - x_2| + |y_1 - y_2|$ (valeurs absolues).

B) Nombre de chemins de longueur minimale

Si le point d'arrivée et de depart sont sur la même ligne ($y_1 = y_2$) ou la même colonne ($x_1 = x_2$) alors il n'y a qu'un chemin de longueur minimale.
Si ce n'est pas le cas ($x_1 \neq x_2$ et $y_1 \neq y_2$), alors, choisir un chemin de taille minimale c'est donc choisir $|x_1 - x_2|$ déplacements en abcisse et $|y_1 - y_2|$ deplacements en ordonnée dans l'ordre que l'on veut car il n'y a pas de murs.
Cela revient à compter le nombre de mots différents de taille $|x_1 - x_2| + |y_1 - y_2|$, composés de $|x_1 - x_2|$ lettres x et de $|y_1 - y_2|$ lettres y. Ce qui se calcule à l'aide du coeficient binomial.

Le nombre de chemins de taille minimale possible est donc :

$$\frac{(|x_1 - x_2| + |y_1 - y_2|)!}{|x_1 - x_2|! \times |y_1 - y_2|!}$$

ou

$$\binom{|x_1 - x_2| + |y_1 - y_2|}{|y_1 - y_2|}$$

ou

$$\binom{|x_1 - x_2| + |y_1 - y_2|}{|x_1 - x_2|}$$

2. Borne maximale du nombre de chemins dans une grille

Dans une grille sans mur de longueur l et hauteur h, et lorsque l'on se deplace uniquement en se rapprochant du point d'arrivée, c'est à dire en diminuant $|x_1 - x_2| ou |y_1 - y_2|$ de 1 à chaque deplacement, la taille maximale d'un chemin est $l + h$
C'est la taille de tous chemins entre le point de depart de coordonnée $(0, 0)$, et le point d'arrivée de coordonnée (h, l) par exemple.
Le nombre de chemins de taille minimale entre ces 2 points est donc la borne maximale du nombre de chemin entre 2 points dans une grille, car le nombre de deplacements à choisir en abcisses et en ordonnées est maximal.
En effet le nombre de chemins entre ces 2 points, d'après les conclusions de la partie précédente, est:

$$\frac{(l+h)!}{(h!+l!)}$$

> Une grille peut être représentée par un graphe où on associe un sommet à chaque case de la grille.
> Une arête relie 2 sommets si les 2 cases qu'ils représentent sont voisines dans la grille.

---

3. L'algorithme de parcours en largeur

Type

- (BFS) est un algorithme de recherche qui permet de parcourir un arbre ou un graphe.

Input

- Entrées : graphe $G = (V, E)$ et sommet $s \in V$, où V represente les sommets du graphe G et E represente les aretes (u, v) de G.

Complexité

- $O(V + E)$ pour les listes d'adjacence
- $O(V^2)$ pour les matrices d'adjacence

Pseudo code

```
début
   créer file(Q);
   marquer(départ);
   enfiler(Q, départ);
   Tant que Q != ∅ faire
      u ← défiler(Q);
      Pour tous les u,v ∈ E faire
         si v non marqué alors
            marquer(v);
            enfiler(Q, v);
```

Description

L'algorithme BFS (Breadth-first search) est un algorithme de recherche. Il réalise sa recherche à travers un parcours transversal d'un graphe, ce qui signifie qu'on visite tous les sommets d'un même niveau avant de passer à un autre.

Afin de visiter tous les sommets d'une graphe. BFS catégorise chaque sommet en deux types - visité et non visité. À partir d'un noeud choisi, BFS visite tous les noeuds adjacents au noeud sélectionné et ainsi de suite[1].

Déroulement de l'algorithme

Étape 1:

- Déclarer une **file d'attente** (*FIFO*) et insérer le sommet de départ.
- Initialiser un **tableau de marquage** (*tableau de booléan*) et marquer le sommet de départ comme visité.

Étape 2
Suivre le processus ci-dessous jusqu'à ce que la file d'attente soit vide :

- Supprimer le premier sommet de la file d'attente.
- Marquer ce sommet comme visité.
- Insérer tous les voisins non visités du sommet dans la file d'attente.

4. Programme calculant un des plus court chemins dans une grille

Entrées : Graphe orienté $G = (S, A)$ et un sommet s ∈ S, la source, et un sommet t ∈ S, l'arrivée.
Sorties : Distance de s aux autres sommets [2].

```
créer filePriorité(pq);
enfiler(pq, (s, 0));

Pour tous i de 0 à taille(distances) faire
   distances(i) ← +∞;
   precedents(i) ← null;

precedents(s) ← None;
distances(s) ← 0;

Tant que pq != ∅ faire
  u ← defiler(pq);

  Si u = t:
    Fin;

  Pour tous les (u, v) ∈ E faire:
    dist ← distances(u) + 1
    Si dist < distances(v) alors
      distances(v) ← dist
      priorite ← dist + heuristic(t, v)
      enfiler(pq, (v, priorite))
      precedents(v) ← u
```

Le chemin le plus court entre le sommet s et un sommet arrivée est donné par le tableau precedents. La case du sommet d'arrivée dans precedents contient le sommet précédent dans le chemin le plus court, et ainsi de suite.
Heuristic est une fonction qui peut aider à trouver le chemin le plus court plus rapidement.
Par exemple elle peut renvoyer la valeur de la distance entre le sommet courant et le sommet arrivée si il n'y avait pas de murs. [3]

5. Complexité de l'algorithme calculant le plus court chemin

Soit un graphe $G = (V, E)$.
Le graphe G représente une grille de longueur l et de hauteur h.
$|V|$ est le nombre de sommets (soit $h \times l$) et $|E|$ le nombre d'arêtes (soit $(h + 1) \times l$).
On initialise les tableaux precedents et distances (qui designe la distance à la source de chaque sommet) en temps $O(|V|)$.

A chaque tour de boucle, on récupère le sommet avec la priorité la plus faible avec une file de priorité. On compare la distance du sommet courant avec la distance de chacun de ses voisins dans le tableau distances. Chaque comparaison se fait en temps constant.
Si la distance du voisin est la plus grande alors on va la modifier et ajouter ce sommet à la file de priorité.

Il n'y a pas de poids et avec la file de priorité on regarde les sommets dans l'ordre de leur distance à la source, donc on ne va jamais modifié plusieurs fois la distance d'un sommet et donc on ne va jamais ajouté plus d'une fois un sommet à la file de priorité au cours de l'execution.

Dans le pire des cas, l'algorithme va visiter tous les sommets et les ajouter tous une fois dans la file. On a donc $|V|$ tours de boucles et $|V|$ ajout/suppression dans la file qui se font en temps $O(log(V))$. On a aussi $2 * E$ soit $O(E)$ operations constantes car on compare les distances de tous les voisins pour chaque sommets.

La complexité dans le pire des cas est donc de $O(|V| + |V| \times log(|V|) + |E|)$ soit $O(|V| \times log(|V|) + |E|)$.
La fonction heuristic n'est pas à prendre en compte ici car elle peut permettre une execution plus rapide mais ce n'est pas toujours le cas.
De plus, le graphe G représente une grille donc $|E|$ c'est $(h + 1) \times l$ donc $|V| \times log(|V|)$ croit plus vite en l'infini donc la complexité de l'algorithme est $O(v * log(v))$ [4].

Une méthode naïve qui ferait la liste des chemins de la grille avant d'enlever ceux bloqués par des murs puis de prendre l'un des chemins restant de longueur minimale, serait de complexité :

- $O(\binom{h+l}{h})$ dans le pire des cas car on fait la liste de tous les chemins possibles. Ce serait beaucoup moins efficaces.

Bibliographie

---

1. About BFS ↩
2. Shortest Path Problems ↩
3. Introduction to A* ↩
4. A* Complexity ↩