MP2I: Graphes (introduction)

Cours

- Définition d'un graphe $G = (V, E)$ avec $E \subseteq \mathcal{P}_2(V)$ ou $E \subseteq V^2$ symmétrique.
- Graphe orienté avec $E \subseteq V^2$.
- Arretes, sommet, degré, degré sortant/entrant, chemin, cycle, connexité, distance, boucle
- Graphes pondéré sur les sommets, sur les arêtes
- Graphes simples comme sans boucles connexes (HP)
- Multigraphes et Hypergraphes (HP)
- Marche et circuit comme chemin/cycle ou l'on peut repasser par le meme sommet (HP)
- Arbre comme graphes connexes acyclique
- Parcours en largeur, parcours en profondeur
- Algorithme de Dijsktra: poids positifs
- Algorithme de BellmanFord: poids négatif (HP)
- Algorithme de Floyd-Warshall: poids négatif & toutes les paires
- Graphes induits, sous-graphes (HP)
- Coloration, cliques, indépendants (HP)
- Graphes biparti

Questions de Cours

- Montrer que la somme des degré est 2 fois le nombre d'arêtes
- Montrer qu'un graphe est biparti ssi il ne contient pas de cycle de taille impairs
- Montrer qu'un graphe G acyclique est tel que $|A| \leq |V| - 1$
- Montrer qu'un graphe G connexe est tel que $|A| \geq |V| - 1$
- Montrer qu'un graphe G est un arbre ssi il est connexe et $|A| \leq |V| - 1$ ssi il est acyclique et $|A| \geq |V| - 1$
- Donner le pseudo-code de Floyd-Warshall
- Donner le pseudo-code de Dijsktra

Exercices à rajouter

- https://arxiv.org/abs/1904.01210 Implémentation incorrecte de Floyd-Warshall
- Graphes de flot !

Autour des degrés

Soit $G = (S, A)$ un graphe sans boucle. Montrer que

- $2|A| = \sum_{v \in V} \deg(v)$
- Le nombre de sommet de degré impair est pair
- Il existe forcément deux sommets de même degré.
- Soit A un arbre ou chaque arc est orienté d'un noeud vers ses enfant. On note N l'ensemble des noeuds qui ne sont pas des feuilles et f le nombre de feuilles. Montrer que

$$\sum_{x \in N} (\deg^+(x) - 1) = f - 1.$$

  En déduire une relation dans le cadre des arbres binaire.

Base des graphes connexes

Soit G un graphe connexe.

- Montrer que deux chemins de longueur maximale s'intersectent.
- Montrer qu'il existe toujours un sommet v tel que $G - v$ reste connexe.

Le puits et le roi

Soit $G = (V, E)$ un graphe orienté *simple*, c'est à dire que $\forall (u, v) \in E, (v, u) \notin E$. On dit que $p \in V$ est un puits si pour tout $u \in V \setminus \{p\}$, il y a une arête de u dans p (dans ce sens).

Question 1 Montrer qu'un tel puits est unique.

Question 2 On donne G par une matrice d'adjacence de taille $|V| \times |V|$. Proposer un algorithme en C qui calcule le puits s'il existe, ou qui renvoie -1 sinon.

Question 3 Montrer que l'on peut faire un tel algorithme en $O(|V|)$.

On dit qu'un graphe orienté est un *tournoi* si pour tout $u, v \in V$, on a soit $(u, v) \in E$ soit $(v, u) \in E$, mais pas les deux. On dit que dans un tournois, x est le *roi* si tous les sommets sont à distance d'au plus 2 de x.

Question 4 Montrer qu'un tel roi existe toujours, et proposer un algorithme en C qui prend une matrice d'adjacence et qui le calcule.

Graphes à unique sortie

On dit que $G = (V, E)$ un graphe orienté est un *graphe à unique sortie* si $\forall v \in V, \deg_+(v) = 1$. Dans ce cas, on le représentera par un tableau T de longueur $|V|$ tel que $T[i]$ corresponde à l'unique sommet $u \in V$ tel que $(i, u) \in E$.

Question 1 Dessiner le graphe correspondant au tableau $[0, 2, 1, 2, 4, 1, 3]$

Question 2 Proposer un algorithme en C qui prend en argument une matrice d'adjacence représenant un graphe à unique sortie et qui renvoie le tableau T le représentant. La signature de la fonction sera `int* get_table(int **adj, int n);`

Question 3 Montrer que pour tout sommet x, il existe un chemin allant de x à un sommet dans un cycle.

Question 4 Proposer un algorithme en C qui à partir d'un tableau et de 2 élements, décide s'ils sont dans la même composante connexe. On proposera un algorithme en $O(C_x + C_y)$ avec C_v le nombre de sommet que l'on peut atteindre à partir de v

Question 5 Montrer qu'il existe un algorithme en $O(C_x + C_y)$ de temps et $O(1)$ d'espace.

Reconnaitre les graphes biconnecté en $O(|V| + |E|)$

On dit qu'un graphe connexe $G = (V, E)$ est *biconnecté* si quel que soit $x \in V$, si on retire x à G alors le graphe reste connexe. On cherche à obtenir un algorithme en $O(|E| + |V|)$ pour tester si G est biconnecté.

Soit $T = (V_T, E_T)$ l'arbre d'un DFS d'un graphe $G = (V_G, E)$.

Question 1 On note $T(x)$ le sous-arbre de racine $x \in V_G$. Montrer que $x \preceq y \iff x \in T(y)$ est une relation d'ordre.

Question 2 Montrer que pour tout $(x, y) \in E_G$, on a que $x \preceq y$ ou $y \preceq x$.

On dit que $(x, y) \in E_G$ est une *arête de retour* si $(x, y) \notin E_T$ avec $x \preceq y$.

Question 3 Montrer que G est biconnecté ssi pour tout $x \in V_G$:
- Si x est la racine alors il existe une arête de retour de la forme (y, x).
- Sinon, il existe une arête de retour de la forme (y, x') avec $y \preceq x \prec x'$ (strict)

Question 4 En déduire un algorithme en $O(|E| + |V|)$ pour tester si un graphe est biconnecté.

Algorithme de Johnson[1]

Soit $G = (S, A)$ un graphe orienté et $w : S \times S \longrightarrow \mathbb{R}$ une pondération des poids potentiellement négative. On va considérer ici un algorithme de calcul des plus courts chemins intitulé *algorithme de Johnson*

1. Rapeller la complexité de l'algorithme de Dijkstra

Soit $h : S \longrightarrow \mathbb{R}$, on pose $w_h(u, v) = w(u, v) + h(u) - h(v)$.

2. Montrer que tous les plus courts chemins pour w sont les memes que pour w_h et qu'il existe un cycle de poids négatif pour w si et seulement si il en existe un pour w_h
3. On suppose que G ne possède pas de cycle à poids négatifs. Trouver un $h : S \longrightarrow \mathbb{R}$ tel que $\forall a, b \in S, w_h(a, b) \geq 0$. On pourra considerer le fait d'ajouter un sommet r relié à tous les autres sommets par un arc de poid nul.
4. En déduire un algorithme permettant de calculer tous les plus courts chemins entre tous les sommets. Comparer sa complexité avec Floyd-Warshall.

[1] Mines Télécom 2024

Reconnaitre un DFS

On se donne $G = (V, E)$ un graphe et T un arbre couvrant de G. On cherche à tester en temps linéaire si T est un graphe qui peut être obtenu en faisant un DFS de G.

Pour T un arbre enraciné de G, on note $x \underset{T}{\preceq} y$ si x est un enfant de y

1. Montrer que $\underset{T}{\preceq}$ est une relation d'ordre
2. Montrer que si T est un DFS, alors $\forall (x, y) \in E(G) \setminus E(T), x \underset{T}{\preceq} y \vee y \underset{T}{\preceq} x$
3. Montrer la réciproque de la question précédente
4. En déduire un algorithme en $O(|V|)$ qui teste si un arbre T est un DFS d'un graphe

Conversions de DAG

Un graphe orienté $G = (V, E)$ sans boucle est un DAG (directed acyclic graph) si c'est un graphe acyclique simplement connexe.

On se donne les types suivants en OCaml:

```
type 'a dag = ('a * int list) array;
type 'a tree = N of 'a * tree list;
```

On supposera écrite la fonction `val count: 'a tree -> int` qui compte le nombre de noeud dans un arbre.

Question 1 Crée une fonction `val get_nb: () tree -> int tree` tel qu'un noeud soit étiqueté par un i s'il est le i-ème noeud dans le parcours préfixe.

Question 2 En déduire une fonction qui converti un arbre en un dag.

Dans un DAG, on appelle un sommet x tel que $\deg^-(x) = 0$ *une source* et un un sommet x tel que $\deg^+(x) = 0$ *un puits*.

Question 3 Donner une fonction `val get_sources: int dag -> int list` qui renvoie la liste de tous les puits.

On supposera écrite la même fonction pour les puits.

Question 4 Proposer un algorithme `val get_shortest: int dag -> int` qui prend un DAG pondéré par des entiers et qui renvoie le chemin le plus court entre une source et un puits.

Chemin hamiltonien dans un tournois

On dit que $G = (V, E)$ un graphe orienté sans boucle est un *tournois* si pour tout $u, v \in V$ distincts, on a exactement une des deux arêtes (u, v) ou (v, u).

Question 1 Proposer le code d'une fonction C `bool is_tournois(int **graph, int n);` qui teste si un graphe `graph` ayant `n` sommets représenté par matrice d'adjacence est bien un tournois.

Un chemin $s_1, ..., s_n$ avec $n = |V|$ est dit *hamiltonien* s'il passe exactement 1 fois par chaque sommet, avec $\forall i < n, (s_i, s_{i+1}) \in E$.

Question 2 Proposer le code d'une fonction C `bool is_hamiltonien(int **graph, int n, int *chemin);` qui teste si `chemin` est un tableau représentant bien un chemin hamiltonien du graphe `graph` ayant `n` sommets (de 0 à n-1) représenté par matrice d'adjacence.

Question 3 Montrer qu'un tournois possède toujours un chemin hamiltonien.

Question 4 Proposer un algorithme en C `int* get_hamiltonien(int **graph, int n)` qui renvoie un chemin hamiltonien. Quel est sa complexité ?

Question 5 Montrer que le graphe ne contient pas de cycle de longueur 3 ssi il possède un unique chemin hamiltonien.

Degrée minimal[2]

On représente un graphe orienté $G = (S, A)$ avec $n = |S|$ par liste d'adjascence implémenté par des tableaux en C comme suit:

```
struct graph_s {
  int n ;
  int* degree;
  int** voisins;
};
```

On aura `degre` et `voisins` qui sont deux tableaux de taille n. Pour tout i, `degree[i]` représente le degrée sortant du sommet i et on a `voisins[i]` qui est le tableau de taille `degree[i]` indiquant les voisins de `i` (n'incluant pas lui-meme).

1. Donner le code en C d'une fonction qui calcule le degrée sortant maximal du graphe.

Pour $s \in S$, on pose $\mathcal{A}(s)$ l'ensemble des sommets accesible depuis s. Pour $X \subseteq S$ on note $d^*(X) = \max\{d^+(s) : s \in X\}$. Pour $x \in S$ on note $d^*(x) = d^*(\mathcal{A}(x))$. On cherche à calculer $d^*(s)$ pour chaque sommet.

2. Ecrire une fonction en C `bool* accesible(int s, struct graph_s g)` qui renvoie un tableau `t` tel que `t[i]` est `true` ssi `i` est accesible depuis `s`.
3. Ecrire une fonction en C `int* d_star_table(struct graph_s g);` qui calcule le tableau des $d^*(s)$ pour tout s. Quel est sa complexité?
4. On suppose que G est acyclique. A l'aide d'un tri topologique, proposer un algorithme qui calcule le tableau des $d^*(s)$ en $O(|S| + |A|)$
5. On ne suppose plus le graphe acyclique. Décrire un algorithme permettant de calculer le tableau des $d^*(s)$ en $O(|S| + |A|)$

[2] CCINP sujet 0

Graphes d'Halin[3] sont 3-connexes

On dit que $G = (V, E)$ est un graphe d'halin si il peut se décomposer comme $E = C \sqcup T$ (disjoint) avec T un arbre sans noeuds de degré 2 et C un cycle passant par toutes les feuilles.

Un graphe est dit *3-connecté* si quel que soit $u, v \in V$, $G - u - v$ le graphe obtenu en retirant u, v et leur arêtes incidentes est toujours connecté.

1. Montrer que si il existe 3 chemins disjoints entre toute paire de sommets alors le graphe est 3-connecté.
2. En déduire que les graphes de Halin sont 3-connexes

[3] On étudie une classe légèrement différente car on ne demande pas que le cycle soit dans l'ordre permettant la planarité ici.

Graphes d'Halin[4] sont hamiltoniens

On dit que $G = (V, E)$ est un graphe d'halin si il peut se décomposer comme $E = C \sqcup T$ (disjoint) avec T un arbre sans noeuds de degré 2 et C un cycle passant par toutes les feuilles.

On dit qu'un graphe G est hamiltonien si il existe un cycle $\langle v_1, ...v_n \rangle$ passant par tous les sommets une fois

1. Montrer que tout arbre sans noeuds de degré 2 peut être obtenue en itérativement transformant une feuille en un sommet de degré k avec $k - 1$ feuilles attaché.
2. Montrer que les graphes d'Halin sont hamiltonien.

Dans les vrais graphes d'Halin, si on retire un sommet alors il reste hamiltonien.

[4] On étudie une classe légèrement différente car on ne demande pas que le cycle soit dans l'ordre permettant la planarité ici.

Nombre cyclomatiques[5]

Si $G = (S, A)$ est un graphe à k composantes connexes, on définit alors son *nombre cyclomatique* $c(G) = |A| - |S| + k$

1. Quel est le nombre cyclomatique d'un arbre ?
2. Soit G un graphe acyclique. Donner et prouver son nombre cyclomatique.
3. Montrer que si $c(G) = 0$ alors G possède un sommet de degré inférieur ou égal à 1.
4. Est-ce que la réciproque de la question 2 est vraie ?

[5] Tiré du TD de la martinière de MPI

Calcul des triangles

Pour $G = (V, E)$ un graphe avec $V = \{1, ..., n\}$, on dit que $\{x, y, z\} \subseteq V$ est un *triangle* si $\{x, y\}, \{y, z\}, \{z, x\} \in E$. On cherche à calculer tous les triangles sans doublons d'un graphe G.

1. Proposer un algorithme en $O(|V|^3)$
2. Soit L_1, L_2 deux listes trié d'entiers de 1 à n, montrer que l'on peut calculer la liste triée des élément appartenant aux deux listes en temps $O(|L_1| + |L_2|)$.
3. Donner un algorithme en $O(|E| \times \Delta)$ pour calculer tous les triangles sans doublons d'un graphe G ou Δ est le degré maximal de G. On supposera que G est donné sous la forme d'une liste d'adjacence.
4. Dans quels cas est-ce que l'algorithme de la question 4 est meilleur que celui de la question 2 ?

Graphes orienté semi-connexes[6]

Soit $G = (S, A)$ un graphe orienté. On dit que G est *semi-connexe* si pour toute paire de sommets $s, t \in S$, soit il existe un chemin de s à t soit il existe un chemin de t à s.

1. Donner un exemple d'un graphe simplement connexe non semi-connexe.
2. On suppose que G possède un tri topologique $(s_1, ..., .s_n)$ de ses sommets. Montrer que G est semi-connexe ssi $\forall i \in [\![1; n-1]\!], (s_i, s_{i+1}) \in A$
3. Dans quel cas un graphe n'admet pas de tri topologique ?
4. Donner/Rapeller un algorithme pour calculer un tri topologique.
5. En considérant une décomposition en composante fortement connexe, proposer un algorithme pour tester si un graphe est semi-connexe.

[6] Tiré du TD de la martinière de MPI

Graphes Cordaux

Un graphe $G = (V, E)$ est dit cordal si tout cycle de longeur ≥ 4 possède une *corde*, c'est à dire une arrete qui relie deux sommets non consécutif du cycle.

On représente G par une matrice d'adjacence `bool adj[n][n]`.

1. Indiquer pour chacun des graphes suivants s'il sont cordaux ou non :
   - K_5, Le graphe complet à 5 éléments
   - C_5 le cycle de longeur 5
   - P_5 le chemin de longeur 5
   - Le carré C_4 avec une des deux diagonales.
2. On dit qu'un sommet v est *simplicial* si l'ensemble de ses voisins forme une clique. Donner une fonctioon `bool is_simplicial(bool **adj, int n, int v);` qui teste si un sommet est simplicial ou non.
3. Montrer que si v est simplicial et si $G - v$ est cordal, alors G est cordal.

On admet le théorème suivant : Tout graphe cordal non vide possède un sommet simplicial.

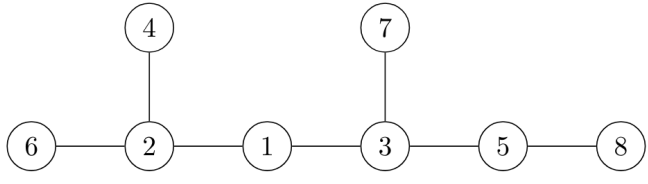
4. En déduire un algorithme `bool is_chordal(bool **adj, int n);` de reconnaissance des graphes cordaux. On donnera sa complexité.
5. Prouver la correction de l'algorithme.
6. Comparer à la complexité de l'algorithme naif.
7. Un ordre $v_1, ..., v_n$ des sommets est appelé *ordre d'élimination parfait* si, pour tout i, le sommet v_i est simplicial dans le sous-graphe induit par $\{v_i, ..., v_n\}$. Montrer qu'un graphe est cordal si et seulement s'il possède un ordre d'élimination parfait, et en déduire le théorème.

Codage de Prufer[7]

Soit $G = (S, A)$ un abre dont les sommets sont numérotés de 1 à $|S|$, on définit son *codage de prufer* comme la suite de sommets de longueur $|S| - 2$ qui est donné par cet algorithme:

$L \longleftarrow [\,]$
tant que $|S| > 2$:
    $s \longleftarrow \min\{s \in S \mid \deg(s) = 1\}$
    $s' \longleftarrow$ l'unique $s' \in S$ tel que $\{s, s'\} \in A$
    Ajouter s' en fin de L
    Retirer s et toutes ses arêtes à G
fin tant que
renvoyer L

1. Montrer le fait la ligne $s \longleftarrow \min\{s \in S \mid \deg(S) = 1\}$ ne pose pas de problème et montrer la terminaison de l'algorithme.
2. Donner un codage de prufer de l'arbre suivant:



3. L'opération qui à un arbre sans étiquettes associe un arbre de prufer est-elle injective ? Justifier
4. Donner un algorithme qui prend un codage de prufer et qui renvoie l'arbre dont c'est le codage
5. En déduire le cardinal du nombre d'arbres à n sommets étiqueté de 1 à n

[7] Tiré du TD de la martinière de MPI