①

NOM : MAURAS Prénom : Simon Jury :

Sujet choisi : Graphes : représentations et algorithmes (925)

Autre sujet :

Ⓘ **Définitions et premiers algorithmes**

Def 1 (Graphe) Soit V un ensemble fini (sommets). On prend $E \subseteq V^2$ (arêtes). Le couple $G = (V, E)$ est un graphe.

Remarque 2 • On pourra munir le graphe d'une fonction $\omega : E \to \mathbb{N}$ et parler de graphe pondéré.
• On n'autorise pas la présence de plusieurs arêtes entre deux sommets. Il s'agit de graphe simple.

Def 3 (Liste d'adjacence) Dans la plupart des algorithmes présentés, le graphe est stocké sous la forme de liste d'adjacence. Pour chaque sommet on dispose de la liste des arêtes sortantes. La complexité spatiale pour un graphe G est donc $O(|V| + |E|)$.

Def 4 (Matrice d'adjacence) Pour tout graphe G, on note $M_G = (m_{ij})$ sa matrice d'adjacence

$$m_{ij} = \begin{cases} 1 & \text{si } (i,j) \in E \\ 0 & \text{si } (i,j) \notin E \end{cases}$$

Stocker la matrice d'adjacence a une complexité spatiale de $O(|V|^2)$.

Def 5 Soit G un graphe. Si la matrice M_G est symétrique, on parle de graphe non orienté. Sinon on parle de graphe orienté. La fonction de poids d'un graphe non orienté est symétrique.

Def 6 Un chemin est une succession de sommets $v_1 \to v_2 \to \dots \to v_n$ avec $(v_i, v_{i+1}) \in E$ pour tout $1 \leq i < n$. Lorsque le graphe est orienté et que $v_1 = v_n$ on parle de cycle.

Def 7 Un graphe orienté sans cycle est dit acyclique. On notera "DAG".

Prop 8 Dans un graphe G, le nombre de chemins de taille l entre s et t dans V est le coefficient (s,t) de M_G^l. ($l \in \mathbb{N}$)

Prop 9 Un graphe orienté G est acyclique si et seulement si les coefficients diagonaux de M_G^l sont nuls pour tout $1 \leq l \leq |V|$.

Def 10 (Tri topologique) Soit G un DAG. Un tri des sommets $v_1, \dots, v_n$ vérifiant la propriété suivante est appelé tri topologique
$\forall 1 \leq i < j \leq n$, il n'existe pas de chemin de v_j à v_i.

Def 11 On définit une relation d'équivalence sur les sommets d'un graphe G :

$$u \sim v \iff \begin{cases} \text{il existe un chemin de } u \text{ à } v \\ \text{il existe un chemin de } v \text{ à } u \end{cases}$$

Les classes d'équivalence pour cette relation sont les composantes connexes (graphe non orienté) ou les composantes fortement connexes (graphe orienté).

Question 12 Comment calculer efficacement
• le plus court chemin entre deux sommets
• Un tri topologique dans un DAG
• Les composantes (fortement) connexes.

Algo 13 (Parcours en largeur) En partant d'un sommet s, on construit et on itère simultanément une liste de sommets en ajoutant à la fin les sommets accessibles depuis le sommet en cours d'exploration.

Prop 14 La complexité du parcours en largeur est $O(|V| + |E|)$.

Prop 15 Lors d'un parcours en largeur on peut maintenir la distance et le plus court chemin à la source s.

Algo 16 (Parcours en profondeur) Récursivement, on explore le graphe en utilisant à chaque fois la première arête menant à un sommet pas encore visité.

Prop 17 La complexité du parcours en profondeur est $O(|E| + |V|)$.

Prop 18 Avec l'algorithme du parcours en profondeur on peut calculer :

- Un tri topologique dans un DAG
- les composantes connexes d'un graphe non orienté.
- Les composantes fortement connexes d'un graphe orienté. (Plus difficile, algorithme de Tarjan ou algorithme de Kosaraju).

DEVELOPPEMENT 1 : Ponts et points d'articulation d'un graphe non orienté.

II Couverture par arbres et chemins

Dans cette partie on considère des graphes non orientés

Def 19 Un chemin eulérien est un chemin utilisant exactement une fois chaque arête du graphe. On parle de cycle eulérien lorsque le point de départ et d'arrivée coïncident

Def 20 Dans un graphe non orienté, on définit le degré d'un sommet par son nombre de voisins.

Thm 21 Dans un graphe non orienté :
- Il existe un cycle eulérien ssi le degré de tous les noeuds est pair.
- Il existe un chemin eulérien ssi le nombre de sommets de degré impair est au plus 2.

Algo 22 (Cycle Eulérien)

En partant d'un sommet on parcourt le graphe en utilisant au plus une fois chaque arête. S'il existe un cycle eulérien, on ne peut se retrouver bloqué qu'en revenant au point de départ. On recommence tant qu'il existe une arête qui n'a pas été parcourue.

Prop 23 On peut implémenter cet algorithme avec une complexité en $O(|V|+|E|)$.

Def 24 Un chemin hamiltonien est un chemin passant exactement une fois par chaque sommet.

Thm 25 Décider s'il existe un chemin Hamiltonien dans un graphe est un problème NP-Complet.

Def 26 On munit un graphe G d'une fonction de poids sur ses arêtes $\omega : E \to \mathbb{N}$. Un arbre couvrant est un sous ensemble $A \subseteq E$ de taille $2|V|-2$ tel que le graphe (V, A) soit connexe. Un arbre couvrant minimal minimise la quantité $\sum_{a \in A} \omega(a)$.

Algo 27 (Prim & Kruskal) Il existe deux algorithmes gloutons permettant de trouver un arbre couvrant minimal dans un graphe pondéré connexe (une seule composante connexe). Les algorithmes de Prim et Kruskal ont une complexité de $O(|E| \log |E|)$

Def 28 (Arbre de Steiner) Soit $G=(V,E)$ un graphe pondéré par $\omega : E \to \mathbb{N}$. Soit $S \subseteq V$ un ensemble de terminaux. Un arbre de Steiner est un sous graphe non orienté (V', E') avec :
- $S \subseteq V' \subseteq V$
- (V', E') connexe
- $|E'| = 2|V'| - 2$

Thm 29 Trouver un arbre de Steiner de poids inférieur à W (donné en entrée) est un problème NP-Complet.

III Plus courts chemins

Dans cette partie on considère des graphes orientés pondérés.

Def 30 On définit le plus court chemin pondéré comme le chemin qui minimise la somme des poids des arêtes du chemin.

Remarque 31 S'il y a un cycle de poids négatif, le plus court chemin est mal défini.

Algorithme 32 (Dijkstra) Dans un graphe G pondéré positivement on cherche tous les plus courts chemins partant d'un sommet s.
On travaille avec avec l'ensemble des sommets accessibles en une étape à partir d'un sommet déjà exploré. À chaque itération on explore le sommet de cet ensemble dont la distance à s est minimale.

Prop 33 En utilisant une file à priorité, on peut implémenter l'algorithme de Dijkstra en $O((|E|+|V|) \log |V|)$.

Algorithme 34 (Bellman-Ford) Dans un graphe G pondéré (poids négatifs autorisés) on calcule par programmation dynamique dist[l][v] = "plus court chemin entre s et v qui utilise au plus l arêtes".
Il existe un cycle de poids négatif ssi il existe v : dist[|V|][v] > dist[|V|+1][v]

Prop 35 On peut implémenter l'algorithme de Bellman-Ford avec une complexité de $O(|V|.|E|)$ en temps et $O(|V|+|E|)$ en espace.

Algorithme 36 (Floyd-Warshal) Dans un graphe G pondéré sans cycles de poids négatif, on calcule par programmation dynamique :

dist[p][d][f] = "Plus court chemin entre d et f qui n'utilise que les sommets numérotés entre 1 et p"

Prop 37 En utilisant la matrice d'adjacence du graphe on peut calculer l'ensemble des plus courts chemin en $O(|V|^3)$ en temps et $O(|V|^2)$ en mémoire.

DEVELOPPEMENT 2 (Johnson)

Dans un graphe G pondéré sans cycles négatifs, on repondère les arête de manière à ce que :

- Les plus court chemin restent inchangés (pas leur poids)
- les arêtes soient pondéré positivement.

De cette manière on peut utiliser l'algorithme de Dijkstra et avoir tous les plus courts chemin en $O(|V| \times ((|V|+|E|)\log|V|))$.

III Flots, coupe, matching et couverture

Def 38 (Réseau de transport)

On considère un graphe orienté G. On munit les arêtes d'une capacité ($c : E \to \mathbb{N}$). On choisit une source s et un puit t dans V. Un flot est une fonction $f : E \to \mathbb{N}$ tq

- $\forall e \in E \quad f(e) \leq c(e)$
- $\forall v \in V \setminus \{s,t\}, \quad \sum_{(u,v)\in E} f(u,v) = \sum_{(v,u)\in E} f(v,u)$

Remarque 39 Soit f un flot compatible avec un réseau de transport (G, c, s, t)

$$\sum_{(s,v)\in E} f(s,v) = \sum_{(v,t)\in E} f(v,t) = F$$

Le problème de flot max consiste en maximiser F

DEVELOPPEMENT 3 (Chemin augmentant)

Algorithmes de Flot max à base de chemin augmentant (Ford Fulkerson, Edmonds-Karp, Capacity Scaling)

Def 40 (Coupe) Soit (G, c, s, t) un réseau de transport

Une coupe est une partition $V = V_s \sqcup V_t$ avec

- $V_s \cap V_t = \emptyset$ • $s \in V_s$ • $t \in V_t$

Une coupe min minimise $C = \sum_{\substack{u\in V_s \\ v \in V_t}} \sum_{(u,v)\in E} c(u,v)$

Thm 41 (Min Cut - Max Flow) Pour tout réseau de transport, C minimal et F maximal. Alors C=F.

Def 42 Un graphe G est bipartite si V se mets sous la forme $V = V_1 \cup V_2$ avec :

- $V_1 \cap V_2 = \emptyset$ • $E \subseteq (V_1 \times V_2) \cup (V_2 \times V_1)$

Def 43 Soit G un graphe non orienté, on identifie (u,v) et (v,u) pour $u, v \in V$. Un matching est un sous ensemble $M \subseteq E$ tel que tout sommet appartient à au plus une arête de M.

Algo 44 ("Blossom" d'Edmonds) L'algorithme "Blossom" d'Edmonds permet de trouver un matching de cardinal maximal en $O(|V|^4)$ à l'aide de chemin augmentants (⚠ difficile)

Prop 45 Trouver un matching maximal dans un graphe bipartite peut se formuler comme un problème de flot max.

Def 46 Une couverture par arêtes est un sous ensemble des arêtes qui couvre tous les sommets

Algo 47 On peut construire de manière gloutonne une couverture par arête maximal à partir d'un matching maximal en temps linéaire.

Def 48 Couverture par sommet : Sous ensemble des sommet qui couvre toutes les arêtes.

Thm 49 Trouver une couverture par sommet de cardinal inférieur à K (donné en entrée) est un problème NP-Complet.

Thm 50 (König) Dans un graphe bipartite, le cardinal minimal d'une couverture par sommet est égal au cardinal maximal d'un matching.