AUFORT William

Leçon d'informatique

Sujet Choisi : 925 Graphes, représentation et algorithmes

Autre Sujet : 924 Théories et Modèles en logique du premier ordre

I) Définition et représentations

a) Définitions

Déf 1 : Un graphe orienté est la donnée d'un ensemble fini V d'éléments appelés sommets, et d'un ensemble $E \subseteq V \times V$ d'éléments appelés arêtes. On note $G = (V, E)$.

Déf 2 : Dans un graphe non orienté $G = (V, E)$, E contient des paires non ordonnées $\{u, v\}$ (que l'on notera encore (u,v)).

Déf 3 : Si $(u,v) \in E$, on dira que v est adjacent à u.

Remarque 4 : Si G est non orienté, c'est une relation symétrique.

Exemple 5 : Sur l'annexe 1a on a représenté informellement le graphe orienté $G = (V,E)$ où $V = \{1,2,3,4,5,6\}$ et $E = \{(1,2),(1,4),(4,2),(2,5),(5,4),(3,5)(3,6),(6,6)\}$

Les arêtes relient les sommets via des flèches (dans le cas orienté) ou des segments (cas non orienté).

b) Première représentation : liste d'adjacence

Idée 6 : Pour chaque sommet $u \in V$, on garde la liste des sommets adjacents à u.

Déf 7 : La représentation en liste d'adjacence de $G = (V,E)$ est la donnée d'un tableau Adj contenant $|V|$ listes, où $\forall u \in V$, $Adj[u]$ contient la liste des sommets adjacents à u.

Ex 8 : Voir annexe 1b pour la représentation du graphe de l'ex 5.

Remarque 9 : Les opérations élémentaires sur les graphes sont la recherche des voisins d'un sommet et l'appartenance d'une arête.

Théorème 10 : Mémoire requise : $\Theta(|V| + |E|)$

Liste des voisins de u : $\Theta(|Adj[u]|) = O(|V|)$

Appartenance de (u,v) à G : $\Theta(|Adj[u]|) = O(|V|)$

c) Seconde représentation : matrice d'adjacence

Idée 11 : On veut répondre rapidement à une requête sur une arête : "$(u,v) \in E$?"

Déf 12 : La représentation en matrice d'adjacence de $G = (V,E)$ est la donnée d'une matrice de taille $|V| \times |V|$, notée $A = (a_{ij})$ où $a_{ij} = \begin{cases} 1 & \text{si } (i,j) \in E \\ 0 & \text{sinon} \end{cases}$

Rem 13 : On suppose les sommets de G numérotés de 1 à $|V|$.

Ex 14 : Voir annexe 1c pour la représentation du graphe de l'ex 5.

Théorème 15 : Mémoire requise : $\Theta(V^2)$, Appartenance d'une arête : $\Theta(1)$

Liste des voisins : $\Theta(|V|)$.

Remarque 16 : Le gros inconvénient est la mémoire requise, ainsi que le temps pour avoir la liste des voisins, surtout si le graphe contient peu d'arêtes.

II) Comment parcourir une telle structure ?

Motivation 17 : Quand on conçoit une structure de données, il faut penser à son parcours. Ici, on veut parcourir le graphe sans passer plusieurs fois au même endroit.

Motivation 18 : La question la plus basique est : quels sommets sont atteignables à partir de $u \in V$.

Remarque 19 : Ainsi le parcours du graphe sera représenté par un ensemble d'arbres de parcours (issus de sources). Ils peuvent également renvoyer d'autres informations, comme nous le verrons.

a) Parcours en largeur :

Idée 20 : À partir d'une source $s \in V$, on cherche à parcourir les sommets dans l'ordre de leur distance à s. Pour cela, on regarde les voisins de s, puis, pour chacun d'eux, tous leurs voisins, etc... L'algorithme découvre tous les sommets à une distance k avant tout sommet à distance $\geq k+1$.

Remarque 21 : Ce parcours permet d'obtenir la distance de tout sommet à la source, que l'on notera dans un tableau d.

Remarque 22 : Si l'on suit l'idée 3, la meilleure structure pour stocker les sommets parcourus est une file.

Algorithme 23. Parcours_en_largeur

Entrée : $G=(V,E)$ un graphe, $s \in V$ une source.
Sortie : d, distances des sommets à s, π tableau des prédécesseurs durant le parcours

```
Pour u∈V, u≠s faire
 | couleur[u] ← BLANC; d[u] ← +∞, π[u] ← NIL
couleur[s] ← GRIS; d[s] ← 0; π[s] ← NIL; F ← ∅; INSERER(F,s)
Tant que F ≠ ∅
 | u ← EXTRAIRE(F)
 | Pour v ∈ Adj[u] faire
 |  | Si (couleur(v) = BLANC)
 |  |  | couleur[v] ← GRIS;   // Sommet à parcourir, mais déjà vu.
 |  |  | d[v] ← d[u]+1
 |  |  | π[v] ← u
 |  |  | INSERER(F,v)
 | couleur[u] ← NOIR   // Sommet parcouru.
```

Théorème 24 : Durant l'exécution, l'algorithme découvre tous les sommets $v \in V$ accessibles depuis s. De plus, pour tout $v \in V$, $d[v]$ contient la longueur du plus court chemin entre v et s. et un de ces chemins est un plus court chemin de s à $\pi(v)$, suivi de l'arête $(\pi(v), v)$. L'algo s'exécute en $O(|V|+|E|)$.

Exemple 25 L'annexe 2 exécute l'algorithme sur l'ex 5 avec $s=1$.

Applications 26 – Algorithme de Prim pour les arbres couvrants de poids minimal
– Algorithme de Dijkstra pour les plus courts chemins (pondérés).

b) Parcours en profondeur

Idée 27 Contrairement au parcours en largeur, on continue d'explorer l'arbre en profondeur tant qu'on découvre de nouveaux sommets. On peut l'implémenter en utilisant cette fois une pile, ou par des appels récursifs.

Algorithme 28 : Parcours_en_profondeur.

Entrée : $G=(V,E)$
Sortie : d = moment auquel un sommet a été découvert ; f = moment auquel un sommet a fini d'être exploré } ce sont des tableaux à $|V|$ éléments.

```
Pour u ∈ V faire
 | couleur(u) ← BLANC; π[u] ← NIL
temps ← 0;
Pour u ∈ V faire
 | Si (couleur(u) = BLANC)
 |  | VISITER(u)
```

```
VISITER(u)
couleur(u) ← GRIS
temps ← temps + 1
d[u] ← temps
Pour v ∈ Adj(u) faire
 | Si couleur(v) = BLANC
 |  | π[v] ← u; VISITER(v)
couleur(u) ← NOIR;
temps ← temps + 1
f[u] ← temps.
```

Théorème 29 Temps d'exécution en $O(|V|+|E|)$.

Exemple 30 Voir annexe 3 pour l'exécution de l'algorithme sur l'ex 5.

Remarque 31 : Les tableaux couleur et (d,f) permettent d'obtenir des informations sur le graphe, pouvant servir à d'autres algos.

Déf 32 : Une arête arrière (u,v) est une arête où v est un ancêtre de u dans le parcours en profondeur.

Lemme 33 : (u,v) est une arête arrière $\Leftrightarrow$ v est étiqueté gris quand (u,v) est explorée.

Théorème 34 G est acyclique (ne contient pas de cycle) $\Leftrightarrow$ il n'y a pas d'arête arrière dans un parcours en profondeur quelconque de G.

Application 35 : Tri topologique d'un graphe en $O(|V|+|E|)$. $\boxed{\text{DEV 1}}$

III Plus courts chemins dans un graphe pondéré

Remarque 36 L'algo 23 nous donnait le plus court chemin de $s \in V$ à u, $\forall u \in V$ en terme de nombre d'arêtes.

Définition 37 Un graphe pondéré est la donnée d'un graphe orienté $G=(V,E)$ et d'une fonction $w: E \to \mathbb{R}$. Le poids d'un chemin $p=\{v_0, v_1, \dots, v_k\}$ est $w(p) = \sum_{i=0}^{k-1} w(p_i, p_{i+1})$. Le poids du plus court chemin de u à v est :
$\delta(u,v) = \min\{w(p),\ p$ chemin de u à $v\}$ s'il existe un tel chemin, $+\infty$ sinon.
Un plus court chemin de u à v est un chemin p tel que $w(p) = \delta(u,v)$.

Remarque 38 Si G contient un cycle p de poids $w(p)<0$, alors certains $\delta(u,v)$ valent $-\infty$.

a) Une seule source : Algorithme de Bellman-Ford

Idée 39 : On garde une borne supérieure sur le poids du plus court chemin pour chaque sommet. À chaque étape de l'algorithme, on regarde si on peut améliorer cette valeur car d'autres valeurs auront été mises à jour. Comme on peut borner le nombre de mises à jour par la longueur d'un plus court chemin, on sait quand termine l'algorithme et on peut détecter la présence de cycles de poids négatifs. L'étape de mise à jour s'appelle <u>relaxation</u>.

<u>Algorithme 40</u> : Bellman_Ford

<u>Entrée</u> : $G = (V,E)$, $w : E \to \mathbb{R}$, $s \in V$ source.

<u>Sortie</u> : d longueurs des plus courts chemins, π prédecesseur, ou CRASH si cycle de poids négatif.

```
Pour v ∈ V faire
  | d[v] ← ∞; π[v] ← NIL
d[s] ← 0;
Pour i = 1 à |V|-1 faire          // Mises à jour
  | Pour (u,v) ∈ E faire
  L L RELAXE(u,v,w)
Pour (u,v) ∈ E faire              // Détection finale
  | si d[v] > d[u] + w(u,v)
  L L CRASH (cycle de poids négatif)
Renvoyer d, π
```

```
RELAXE(u,v,w) :
  si (d[v] > d[u]+w(u,v))
   | d[v] ← d[u]+w(u,v)
   L π[v] ← u
```

<u>Ex 40.5</u> : L'annexe 4 présente l'exécution de l'algo sur un exemple.

<u>Théorème 41</u> : L'algorithme est correct et s'exécute en temps $O(|V| \times |E|)$.

<u>Application 42</u> : Trouver la route la plus courte de notre point de départ vers une autre ville

b) <u>All-to-All : Algorithme de Floyd Warshall</u>

<u>Motivation 43</u> : Dans une carte / GPS, on veut calculer pour chaque couple de villes d'un ensemble leur distance.

<u>Méthode naïve 44</u> : Appliquer l'algo 40 pour chaque $s \in V \to O(|V|^2 \times |E|)$.

DEV2 <u>Algorithme 45</u> : L'algorithme de Floyd-Warshall résout le problème par programmation dynamique en temps $O(|V|^3)$ si G ne contient pas de cycle de poids négatif.

<u>Application 46</u> : Clôture transitive d'un graphe en $O(|V|^3)$.

IV <u>Difficulté des certains problèmes de graphe</u>

<u>Remarque 47</u> Jusqu'à présent, nous n'avons vu que des problèmes pouvant être résolus en temps polynomial. Est-ce toujours le cas ?

a) <u>Autour des cycles</u>

<u>Déf 48</u> : Soit G un graphe orienté. Un <u>tour eulérien</u> est un cycle qui passe par chaque arête de G exactement une fois. Un <u>cycle hamiltonien</u> est un cycle qui passe par chaque SOMMET de G exactement une fois.

<u>Théorème 49</u> G admet un tour eulérien $\iff \forall v \in V$, $\mathrm{Card}\{u / (u,v) \in E\} = \mathrm{Card}\{u / (v,u) \in E\}$.

On peut alors savoir si G admet un tour eulérien en $O(E)$.

<u>Théorème 50</u> Le problème de décision suivant : " Etant donné G, G admet-il un cycle hamiltonien ? " est NP-Complet.

<u>Remarque 51</u> Surprenant alors que ces problèmes ont l'air très proche.

<u>Corollaire 52</u> : Le problème du voyageur de commerce (TSP) est NP-Complet

<u>Théorème 53</u> : Si la fonction de coût du problème TSP satisfait l'inégalité triangulaire, alors il existe un algorithme de 2-approx au problème TSP.

b) <u>Plus courts et plus longs chemins</u>

<u>Remarque 54</u> On a vu que la recherche du plus court chemin peut être résolue en temps polynomial.

<u>Application 55</u> : Diamètre d'un graphe $D(G) = \max(\delta(u,v), (u,v) \in V^2)$.

<u>Théorème 56</u> : Le problème du plus long chemin est NP-Complet : « Etant donnés $G = (V,E)$, $c : E \to \mathbb{N}$, $k \in \mathbb{N}$, $(s,t) \in V^2$, existe-t-il un chemin simple de s à t de coût $\geq k$? »

c) <u>Isomorphisme de graphes</u> :

<u>Déf 57</u> : On définit le problème GI : " Etant donné G_1, G_2 deux graphes, existe-t-il une bijection $\varphi : V_1 \to V_2$ telle que $\forall (u,v) \in V_1^2$, $(u,v) \in E_1 \iff (\varphi(u), \varphi(v)) \in E_2$? »

<u>Remarque 58</u> : G_1 et G_2 représentent alors le même graphe modulo renommage des sommets.

<u>Théorème 59</u> · • GI $\in$ NP

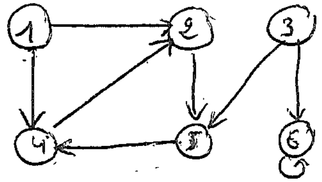
• Si GI est NP-complet, alors $\Sigma_2 = \Pi_2$ (difficile)

<u>Remarque 60</u> : – Savoir si GI est NP-complet est un problème ouvert.

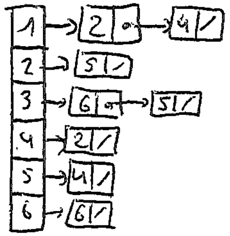
– GI définit à lui seul une classe de complexité, qui porte le même nom…

Annexe 1

a) Exemple de graphe



b) Liste d'adjacence associée

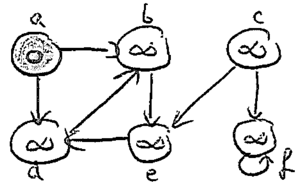


c) Matrice d'adjacence

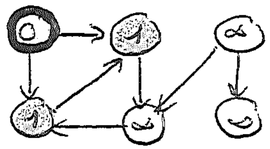
$$
\begin{array}{c|c}
 & 1\ 2\ 3\ 4\ 5\ 6 \\
\begin{matrix}1\\2\\3\\4\\5\\6\end{matrix} &
\begin{pmatrix}
0&1&0&1&0&0\\
0&0&0&0&1&0\\
0&0&0&0&1&1\\
0&1&0&0&0&0\\
0&0&0&1&0&0\\
0&0&0&0&0&1
\end{pmatrix}
\end{array}
$$

Annexe 2 : Parcours en largeur à partir du sommet 1

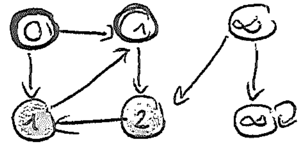
(Dans les sommets on note d[u] à chaque instant).



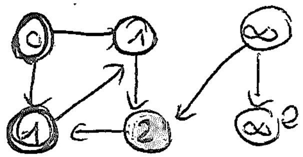
F = {a}



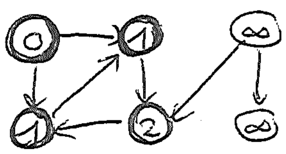
F = {b,d}



F = {d,e}



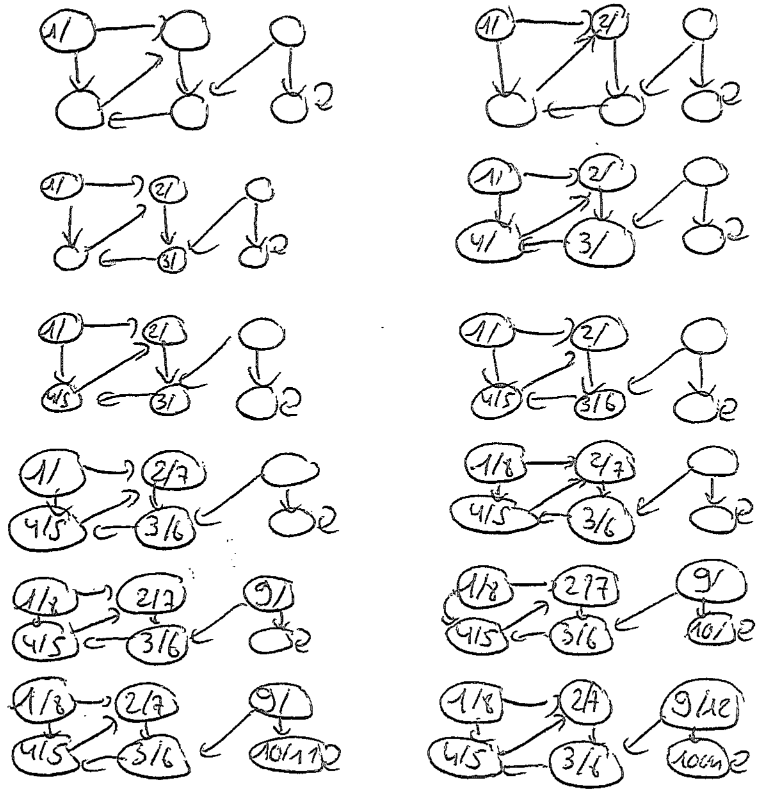
F = {e}



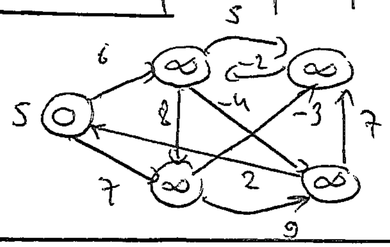
F = ∅ → Fin.

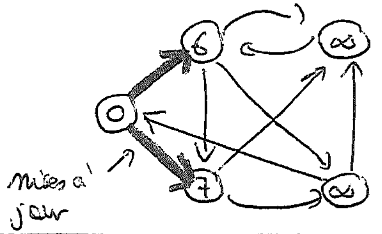
Annexe 3 Parcours en profondeur à partir de 1.

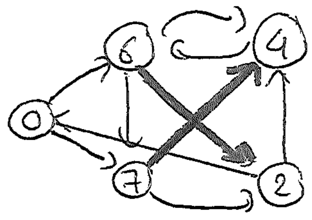
Dans les sommets on note (d(u)/f(u)). On suppose que l'ordre dans V est (1,2,3,4,5,6).

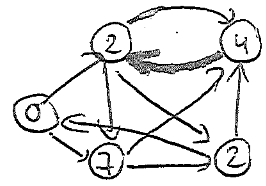


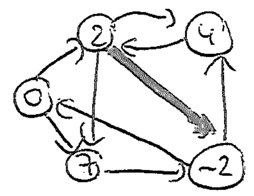
Annexe 4 Exemple pour Bellman-Ford.











Plus de mises à jour.