NOM : BOLLE **Prénom :** Quentin **Jury :**

Algèbre ← Entourez l'épreuve → **Analyse**

Sujet choisi : 925 . Graphes : représentations et algorithmes

Autre sujet : Réf : • Intro to graph theory, D. • Algo. graph theory, J.A. Mc Hugh + Cormen

1

Motivation : omniprésence des graphes en informatique
- Internet : sommets (ordinateurs) + arêtes (cable, fibre)
- Web : sommets (pages Web) + arêtes (liens cliquables)

I) Représentations

0) Vocabulaire

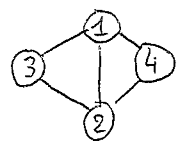
Def. • graphe / sommets / arêtes
• orienté / non orienté

Def : • chemin / chemin simple
• cycle
• poids d'une arête / poids d'un chemin
• longueur d'un chemin / distance entre sommets
• graphe connecté

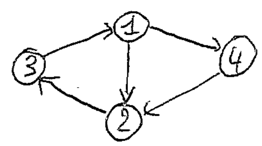
Notation : • $G=(S,A)$ avec $n=\#S$ et $m=\#A$
• $i \mapsto j$: $(i,j) \in A$; $i \rightsquigarrow j$: chemin de i vers j

1) Deux exemples

Cas non-orienté



Cas orienté



2) Représentation statique : matrice d'adjacence

$\mathcal{A} = (a_{ij})_{(i,j)\in[\![1,n]\!]^2}$ avec $a_{ij} = \begin{cases} 0 & \text{si } i \not\mapsto j \\ 1 & \text{si } i \mapsto j \end{cases}$

Exemples :

Cas non-orienté

$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

Cas Orienté

$$\begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

Modification éventuelle : - remplacer 0 par ∞
- remplacer 1 par poids arête

Avantages : - structure simple
- détermination de $i \mapsto j$ en coût constant

Inconvénients : - cas $m \ll n^2$ (graphe creux) beaucoup de 0 dans la matrice
- accès à l'ensemble des voisins d'un sommet : coût $O(n)$.

3) Représentation dynamique : liste d'adjacence

On donne la liste des sommets atteignables par $v \in S$. Ces listes réunies forment la liste d'adjacence.

Exemples :

Cas non-orienté

$\{1:[2,3,4] ; 2:[1,3,4] ; 3:[1,2] ; 4:[1,2]\}$

Cas orienté

$\{1:[2,4] ; 2:[3] ; 3:[1] ; 4:[2]\}$

Modification éventuelle : - ajouter le poids des arêtes
- graphe orienté : liste des prédécesseurs

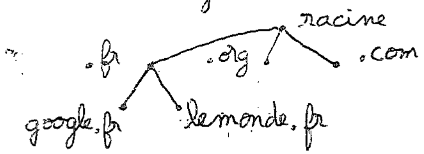
Avantages : - cas $m \ll n^2$
- accès à l'ensemble des voisins atteints d'un sommet
- opération ajout / délétion

Inconvénient : - moins lisible
- détermination de $i \mapsto j$: coût en $O(n)$ dans le pire cas (cette borne peut être améliorée en modifiant l'organisation des listes

4) Cas particulier : les arbres

Déf : Un arbre est un graphe connecté acyclique.
Arbre enraciné / racine. Arbre binaire

Exemple : Domain Name System



Structures particulières : arbre binaire de recherche, AVL, etc...

→ **DEVPMT1 : Codage de Huffman**

II) Parcours de graphes, chemin optimal

1) Parcourir un graphe

a) Algorithme général

On dispose d'une structure de donnée D avec les opérations AJOUT(D, x) et ENLEVE(D) (fournit un élément de D).

PARCOURS :

Entrée : graphe $G = (S, A)$ (donné par liste d'adjacence) et $s \in S$

Code :
```
1. soit C : S ↦ {Blanc, Gris, Noir}, initialisée à Blanc
2. C(s) := Gris
3. D := ∅
4. AJOUT(D, s)
5. Tant que D ≠ ∅, faire
6.    | u = ENLEVE(D)
7.    | Pour tout v ∈ S de la liste d'adjacence de u
8.    |    | Si C(v) = Blanc
9.    |    |    | C(v) := Gris
10.   |    |    | AJOUT(D, v)
11.   | C(u) := Noir
```

b) Parcours en profondeur

Déf : Si D est une pile, PARCOURS réalise un parcours en profondeur.

Prop : Un sommet n'est noir que si l'ensemble des sommets atteignables sont noirs.

Complexité : $O(n+m)$

Application : Tri topologique d'un graphe, détection de cycle

c) Parcours en largeur

Déf : Si D est une file, PARCOURS réalise un parcours en largeur.

Prop : Un sommet dont la distance à s est de k n'est parcouru qu'après tous les sommets de distance au plus $k-1$ de s.

Complexité : $O(n+m)$

Application : Algorithme de Dijkstra, Algorithme de Prim, calcul distance

2) Chemin de poids minimal

Motivation : recherche du plus court itinéraire

Hypothèse : poids des arêtes $\geqslant 0$, poids infini pour $i \not\to j$, $\exists s \leadsto f$

DIJKSTRA : **Entrée** : Liste d'adjacence d'un graphe (S, A), $s \in S$ et $f \in S$

Sortie : poids minimal d'un chemin de s vers f.

Code :
```
1. Soit D := {s} et t : S ↦ ℝ⁺, initialisée par t(s) = 0 et t(u) = poids(su) (u ≠ s)
2. Tant que t(f) = ∞, faire
3    | Sélectionne u ∉ D avec t(u) = min t(z)   (z ∉ D)
4    | D := D ∪ {u}
5.   | Pour tout uz ∈ A avec z ∉ D, faire
6    |    t(z) := min (t(z), t(u) + poids(uz))
7  Renvoie t(f).
```

Complexité : $O(n^2)$

3) <u>Algorithme de Floyd-Warshall</u>

<u>Objectif</u> : calculer le poids minimal d'un chemin entre u et v, pour tous les couples $(u,v) \in S^2$ en même temps.

<u>Objectif secondaire</u> : calculer la fermeture transitive ($i \overset{*}{\mapsto} j$ ssi $i \rightsquigarrow j$)

→ <u>DEVPMT2. Algorithme de Floyd-Warshall, fermeture transitive</u>

<u>Application de la fermeture transitive</u> : précalculer l'ensemble des dépendances dans un compilateur.

III) <u>Arbres couvrants</u>

Motivation : - extraire le "squelette" d'un graphe
- optimisation d'un réseau (autoroutier, pétrole, etc.)

a) <u>Existence</u>

<u>Déf</u> : Un arbre couvrant d'un graphe G est un sous-graphe de G, avec tous ses sommets, qui est un arbre

<u>Th</u> : L'algorithme PARCOURS crée un arbre couvrant de G.

b) <u>Minimisation d'un arbre couvrant</u>

<u>Objectif</u> : trouver l'arbre couvrant T qui minimise $\sum_{(i,j) \in T} \text{poids}(i,j)$

<u>Hypothèse</u> : on suppose que poids > 0

<u>KRUSKAL</u> :

<u>Entrée</u> : Liste d'adjacence (avec poids) d'un graphe connecté $G = (S,A)$

<u>Sortie</u> : Arbre couvrant de poids minimal

<u>Code</u> :
```
1. Soit H un graphe avec S(H) = S et (H) = ∅
2. Tant que H n'est pas connecté, faire
3     | Soit u ∈ A \ A(H) de poids minimal
4     | Si u connecte deux composantes de H, faire
5     |     | A(H) := A(H) ∪ {u}
6     | Sinon, A := A \ {u}
7. Renvoie H.
```

<u>Complexité</u> : $O(m \log n)$

<u>Observation</u> : c'est un algorithme glouton.

<u>PRIM</u>

<u>Entrée</u> : Liste d'adjacence (avec poids) d'un graphe connecté $G = (S,A)$ et $s \in S$

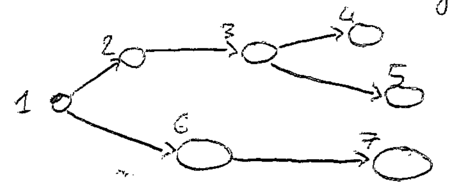
<u>Sortie</u> : Arbre couvrant de poids minimal

<u>Code</u> :
```
1. Soit H un graphe avec S(H) = {s} et A(H) = ∅
2. Soit t : S ↦ ℝ+, initialisée par t(s) = 0 et t(u) = ∞ (u ≠ s)
3.    Soit Q = S
4  Tant que Q ≠ ∅, faire
5     | Soit u ∈ Q avec t(u) = min t(z)   (z ∈ Q)
6     | Pour tout v tel que u ↦ v
7     |     | Si v ∈ Q et poids(u,v) < t(v)
8     |     | t(v) := poids(u,v)
9     | Q := Q \ {u}
10    | S(H) := S(H) ∪ {u}
11    | A(H) := A(H) ∪ {w u}        (où w ∈ S(H) et poids(w,u) = t(u))
12 Renvoie H
```

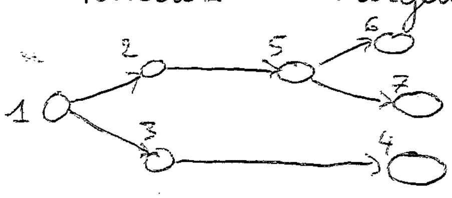
<u>Complexité</u> : * $O(m \log n)$
* en utilisant des tas de Fibonacci : $O(n \log n)$

4

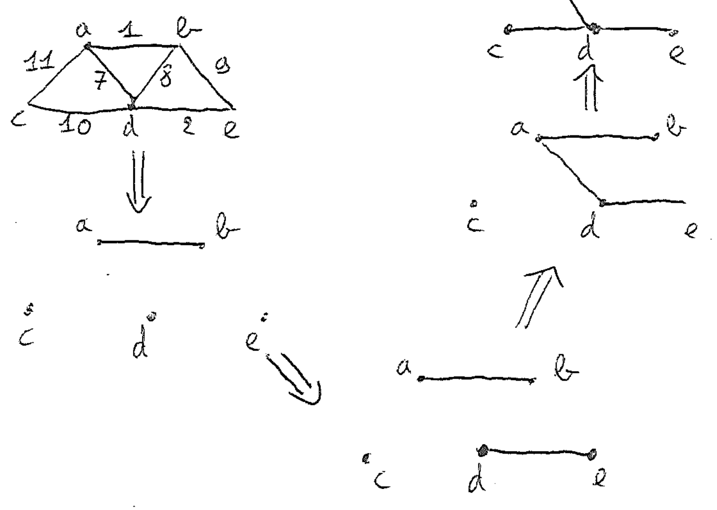
Parcours en longueur



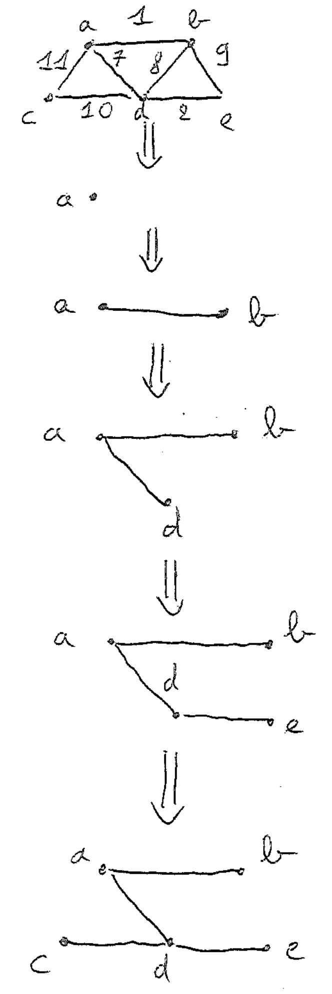
Parcours en largeur



Kruskal



Prim



Dijkstra