IFT436 – Algorithmes et structures de données

Université de Sherbrooke

Examen périodique

Enseignant: Michael Blondin
Date: jeudi 20 octobre 2022
Durée: 110 minutes

Directives:

— Vous devez répondre aux questions dans le **cahier de réponses**, *pas* sur ce questionnaire;
— **Une seule feuille** de notes au format 8½" × 11" est permise;
— Les **fiches récapitulatives** des chapitres 1 à 4 se trouvent à la fin de ce questionnaire;
— **Aucun matériel additionnel** n'est permis;
— **Aucun appareil électronique** (calculatrice, téléphone, montre intelligente, etc.) n'est permis;
— Vous devez donner **une seule réponse** par sous-question;
— L'examen comporte **4 questions** sur **5 pages** valant un total de **50 points**;
— La correction se base notamment sur la **clarté**, l'**exactitude** et la **concision** de vos réponses, ainsi que sur la **justification** pour les questions qui en requièrent une;
— Si la **base d'un logarithme** n'est pas spécifiée, il s'agit de la base 2: « log » dénote « $\log_2$ »;
— Les indices d'une séquence **débutent à 1**; autrement dit, $s = [s[1], s[2], \ldots, s[n]]$ si $n = |s|$.

Question 1: analyse des algorithmes

(a) Ordonnez ces fonctions f_1, f_2, f_3, f_4 et f_5 selon la notation $\mathcal{O}$ (de la plus faible vers la plus haute complexité): 7 pts

$$\underbrace{\sum_{i=0}^{n-1}(2^i - 5)}_{f_1}, \quad \underbrace{100 + (2/n)}_{f_2}, \quad \underbrace{(n^2/3 + 10n) \cdot \log(n^{42})}_{f_3}, \quad \underbrace{\log(2n/5)}_{f_4}, \quad \underbrace{\sum_{i=1}^{n}(n - i)}_{f_5}.$$

$$\underbrace{\mathcal{O}(f_2)}_{\mathcal{O}(1)} \subset \underbrace{\mathcal{O}(f_4)}_{\mathcal{O}(\log n)} \subset \underbrace{\mathcal{O}(f_5)}_{\mathcal{O}(n^2)} \subset \underbrace{\mathcal{O}(f_3)}_{\mathcal{O}(n^2 \log n)} \subset \underbrace{\mathcal{O}(f_1)}_{\mathcal{O}(2^n)}$$

(b) Considérons un algorithme $\mathcal{A}$ qui reçoit en entrée un graphe non dirigé $\mathcal{G}$ de n sommets et m arêtes, avec comme pré-condition que $\mathcal{G}$ est un arbre. Après analyse, nous obtenons $t_{\max} \in \mathcal{O}(3m + 5n \log n + 2)$, où $t_{\max}$ dénote le temps d'exécution dans le pire cas de $\mathcal{A}$. Pouvons-nous affirmer que $t_{\max} \in \mathcal{O}(n \log n)$? Justifiez. 2 pts

Oui. On a $m = n - 1$ car $\mathcal{G}$ est un arbre. Ainsi, $t_{\max} \in \mathcal{O}(m + n \log n) = \mathcal{O}(n + n \log n) = \mathcal{O}(n \log n)$.

(c) Soit $f(n) \coloneqq n^2 - n + 42$. Montrez que $f \in \Omega(n^2)$ en identifiant explicitement une constante multiplicative c et un seuil n_0. 2 pts

On a $f \in \Omega(n^2)$ avec $c \coloneqq 1/2$ et $n_0 \coloneqq 2$ car:

$$\begin{aligned} f(n) &= n^2 - n + 42 \\ &\geq n^2 - n && \forall n \geq 0 \\ &\geq n^2 - (n/2) \cdot n && \forall n \geq 2 \\ &= (1/2) \cdot n^2. \end{aligned}$$

(d) Dites auxquels de ces ensembles appartient la fonction $n \log n$: 2 pts

$$\mathcal{O}(n), \quad \mathcal{O}((3/2)^n), \quad \mathcal{O}(1/5 \cdot n \log n - 2), \quad \Omega(\log n), \quad \Omega(n^2), \quad \mathcal{O}(n^2), \quad \Theta(1).$$

$$\mathcal{O}((3/2)^n),\ \mathcal{O}(1/5 \cdot n \log n - 2),\ \Omega(\log n),\ \mathcal{O}(n^2)$$

Question 2: tri

Considérons cet algorithme de tri:

Entrée: séquence s de $n \in \mathbb{N}$ éléments comparables
Sortie: s triée

```
1  terminé ← faux
2  tant que ¬terminé
       // Phase 1: corrections
3      pour i ∈ [1..n]
4          pour j ∈ [n..i]                    // j est décrémenté de n jusqu'à i
5              si s[i] > s[j] alors
6                  s[i] ↔ s[j]                // inverser le contenu de s[i] et s[j]
7                  quitter la boucle « pour j »   // passer directement au prochain i
       // Phase 2: vérifier si séquence maintenant triée
8      terminé ← vrai
9      pour k ∈ [1..n − 1]
10         si s[k] > s[k + 1] alors
11             terminé ← faux
12 retourner s
```

(a) Exécutez l'algorithme sur l'entrée $s = [33, 11, 22]$. Donnez une trace qui illustre clairement le contenu de la séquence s et du compteur i à la fin de chaque tour de la boucle « **pour** i ». 3 pts

s (i **en gras**)
$[33, 11, 22]$
$[\mathbf{22}, 11, 33]$
$[22, \mathbf{11}, 33]$
$[22, 11, \mathbf{33}]$
$[\mathbf{11}, 22, 33]$
$[11, \mathbf{22}, 33]$
$[11, 22, \mathbf{33}]$

(b) Les affirmations suivantes sont toutes vraies. Choisissez-en *deux* (pas plus!) et montrez leur véracité. 8 pts

(i) L'algorithme fonctionne en temps $\Omega(n^2)$ dans le pire cas.

Considérons la famille d'entrées $[1, 2, \ldots, n]$. Comme la condition du « **si** » n'est jamais satisfaite, la ligne 5 est atteinte $n + (n-1) + \ldots + 1 = n(n+1)/2$ fois. Ainsi, $t_{\text{max}} \in \Omega(n^2)$.

(ii) L'algorithme fonctionne en temps $\mathcal{O}(n^3)$ dans le pire cas.

Après l'itération ℓ de la boucle « **tant que** », les ℓ plus grands éléments sont bien ordonnés à la fin de s. Il y a donc au plus n itérations de cette boucle. Le temps d'exécution de son corps appartient à $\mathcal{O}(n \cdot n + n) = \mathcal{O}(n^2)$. Ainsi, $t_{\text{max}} \in \mathcal{O}(n \cdot n^2) = \mathcal{O}(n^3)$.

(iii) L'algorithme fonctionne en temps $\Omega(n)$ dans le meilleur cas.

La boucle « **pour** i » est toujours exécutée n fois. Peu importe l'entrée, il y a donc au moins n opérations. Ainsi, $t_{\text{min}} \in \Omega(n)$.

(iv) L'algorithme fonctionne en temps $\mathcal{O}(n)$ dans le meilleur cas.

Sur la famille $[2, 3, \ldots, n, 1]$, la boucle « **pour** j » effectue toujours une seule itération, et la séquence devient triée après une seule itération de la boucle « **tant que** ». Ainsi, $t_{\text{min}} \in \Omega(n + n) = \Omega(n)$.

(v) L'algorithme termine et est correct.

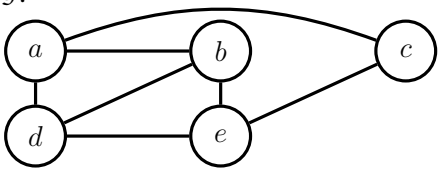
Si la phase 2 échoue, alors il existe au moins une inversion dans s. La phase 1 va donc forcément en corriger une. Puisque le nombre d'inversions décroît strictement, il n'y en a éventuellement plus. À ce moment, la phase 2 confirme que la séquence est triée, et l'algorithme se termine.

(c) Dites si l'algorithme est stable. Justifiez. 3 pts

Non, sur $s = [②, 2, 1]$, l'algorithme retourne $[1, 2, ②]$ et inverse donc l'ordre relatif de ② et 2.

Question 3: graphes

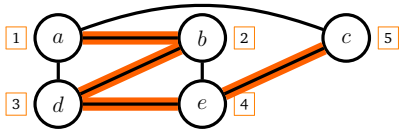
(a) Considérons ce graphe non dirigé $\mathcal{G}$:



(i) Donnez le contenu de $adj[a]$ et $adj[b]$, où adj est la liste d'adjacence de $\mathcal{G}$. 2 pts

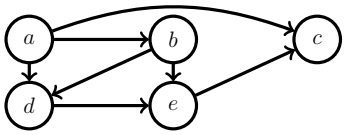
$adj[a] = [b, c, d]$ et $adj[b] = [a, d, e]$.

(ii) Parcourez $\mathcal{G}$ en profondeur à partir du sommet a. Laissez une trace de l'ordre dans lequel les sommets sont marqués, et de l'arbre couvrant formé des arêtes qui contribuent à la découverte d'un sommet. Lors de l'exploration d'un sommet, considérez ses voisins en ordre alphabétique. 2 pts



(iii) Ajoutez des directions aux arêtes de $\mathcal{G}$ de telle sorte à ce que le graphe dirigé résultant possède un, et un seul, ordre topologique. 2 pts

Voici une solution parmi d'autres:



(b) Soit $\mathcal{G} = (V, E)$ un graphe dirigé. Rappelons qu'une composante fortement connexe de $\mathcal{G}$ est formée d'un ensemble maximal de sommets mutuellement accessibles. En particulier, la composante fortement connexe qui contient le sommet u est formée des sommets $\{v \in V : u \xrightarrow{*} v \text{ et } v \xrightarrow{*} u\}$. 7 pts

Donnez un algorithme, sous forme de pseudocode, qui résout ce problème:

ENTRÉE: graphe dirigé $\mathcal{G} = (V, E)$ (sous forme de liste d'adjacence) et sommet $u \in V$
SORTIE: sommets de la composante fortement connexe qui contient u

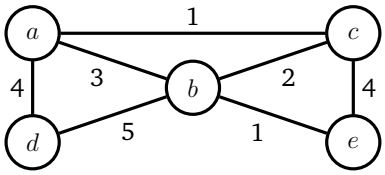
Analysez son temps d'exécution dans le pire cas. Pour obtenir tous les points, il doit appartenir à $\mathcal{O}(|V|+|E|)$.

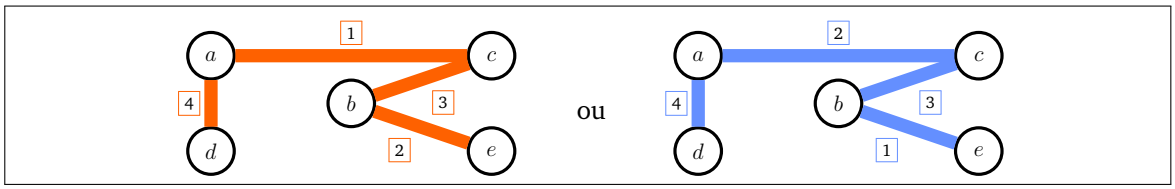
Précision: au besoin, vous pouvez invoquer des algorithmes couverts en classe sans les réimplémenter.

```
// Construire représentation du graphe renversé
adj' ← [x ↦ [] : x ∈ V]                                   // O(|V|)
pour x ∈ V                                                // O(Σ_{x∈V}(1 + deg(x))) = O(|V| + |E|)
    pour y ∈ adj[x]
        ajouter x à adj'[y]
// Trouver sommets que u peut atteindre et vice-versa
X  ← parcours-en-profondeur(adj, u)                       // O(|V| + |E|)
X' ← parcours-en-profondeur(adj', u)                      // O(|V| + |E|)
// Calculer intersection de X et X'
accessible  ← [x ↦ faux : x ∈ V]                          // O(|V|)
composante ← []
pour x ∈ X                                                // O(|V|)
    accessible[x] ← vrai
pour x ∈ X'                                               // O(|V|)
    si accessible[x] alors ajouter x à composante
retourner composante
```

Question 4: algorithmes gloutons

(a) Identifiez un arbre couvrant minimal de ce graphe pondéré avec l'algorithme de Kruskal (celui qui fusionne des arbres dans une forêt). Laissez une trace de l'ordre dans lequel les arêtes de l'arbre couvrant sont sélectionnées. 3 pts





(b) Une personne prétend avoir découvert un graphe (non dirigé, pondéré et connexe) qui a deux arbres couvrants de poids minimal: un constitué de trois arêtes de poids respectivement $[2, 4, 6]$, et un autre constitué de trois arêtes respectivement de poids $[2, 5, 5]$. Cela est impossible. Pourquoi? 3 pts

Sol. 1: Soient e et f les arêtes de poids 2. Exécutons l'algorithme de Kruskal. La première arête choisie a un poids de 2. La deuxième arête sera la deuxième plus petite, car deux arêtes ne peuvent pas former un cycle simple. Si $e = f$, alors la deuxième arête choisie a un poids de 4. Si $e \neq f$, alors la deuxième arête choisie a un poids de 2. Dans les deux cas, $[2, 5, 5]$ est impossible.

Sol. 2: Considérons le premier arbre avec les arêtes de poids $[2, 4, 6]$. Le deuxième arbre nous apprend qu'il y a au moins deux arêtes g et h de poids 5. On peut substituer l'arête de poids 6 du premier arbre par g ou h, car au plus une des deux peut créer un cycle simple. Cela donne un meilleur poids, ce qui contredit la minimalité.

(c) Rappelons qu'un sommet d'un arbre est une *feuille* ssi son degré est 1. Le *feuillage* d'un arbre est sa quantité de feuilles. Par exemple, ces deux arbres ont un feuillage respectif de 3 et 2:

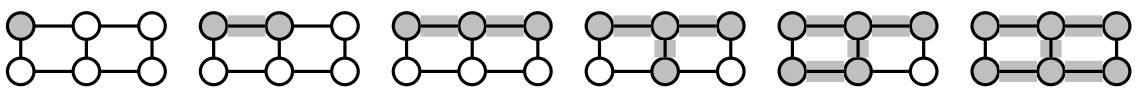
Considérons le problème qui consiste à identifier un arbre couvrant à *feuillage maximal*. Pour le résoudre, nous proposons l'approche gloutonne suivante (inspirée de l'algorithme de Prim–Jarník):

— on débute par un sommet arbitraire, afin de former un arbre trivial;

— on considère les arêtes adjacentes à l'arbre actuel; et on en choisit une dont l'ajout ne forme pas de cycle simple, et qui maximise le nombre de feuilles dans l'arbre résultant de son ajout (s'il y a plusieurs choix, on en fait un arbitraire).

— on répète cette dernière étape jusqu'à l'obtention d'un arbre couvrant.

Voici une exécution possible de cette procédure:

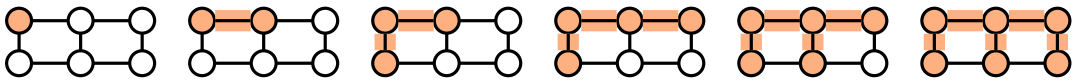


(i) L'arbre couvrant obtenu ci-dessus a un feuillage maximal. Pourquoi? 2 pts

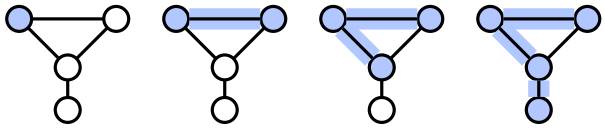
Un arbre de n sommets possède au plus $n-1$ feuilles. Le graphe n'a pas d'arbre couvrant de $6-1=5$ feuilles, car il n'a aucun sommet de degré 5. Le feuillage maximal est donc bien 4.

(ii) Montrez que la procédure gloutonne n'est pas correcte (en général). 2 pts

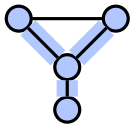
Sol. 1: En exécutant l'algorithme avec des choix arbitraires différents (mais valides), nous obtenons un arbre à feuillage 3, alors que le feuillage maximal est pourtant 4:



Sol. 2: Une exécution valide sur ce graphe donne un feuillage de 2:



Il existe pourtant un arbre couvrant de feuillage 3:



Annexe:

Fiches récapitulatives

1. Analyse des algorithmes

Temps d'exécution

- *Opérations élémentaires*: dépend du contexte, souvent comparaisons, affectations, arithmétique, accès, etc.
- *Pire cas* $t_{\max}(n)$: nombre maximum d'opérations élémentaires exécutées parmi les entrées de taille n
- *Meilleur cas* $t_{\min}(n)$: même chose avec « minimum »
- $t_{\max}(m,n)$, $t_{\min}(m,n)$: même chose par rapport à m et n

Notation asymptotique

- *Déf.*: $f \in \mathcal{O}(g)$ si $n \geq n_0 \to f(n) \leq c \cdot g(n)$ pour certains c, n_0
- *Signifie*: f croît moins ou aussi rapid. que g pour $n \to \infty$
- *Transitivité*: $f \in \mathcal{O}(g)$ et $g \in \mathcal{O}(h) \to f \in \mathcal{O}(h)$
- *Règle des coeff.*: $f_1 + \ldots + f_k \in \mathcal{O}(c_1 \cdot f_1 + \ldots + c_k \cdot f_k)$
- *Règle du max.*: $f_1 + \ldots + f_k \in \mathcal{O}(\max(f_1, \ldots, f_k))$
- *Déf.*: $f \in \Omega(g) \leftrightarrow g \in \mathcal{O}(f)$; $f \in \Theta(g) \leftrightarrow f \in \mathcal{O}(g) \cap \Omega(g)$
- *Règle des poly.*: f polynôme de degré $d \to f \in \Theta(n^d)$

Notation asymptotique (suite)

- *Simplification*: lignes élem. comptées comme une seule opér.
- *Règle de la limite*:
$$\lim_{n\to+\infty} \frac{f(n)}{g(n)} = \begin{cases} 0 & f \in \mathcal{O}(g) \text{ et } g \notin \mathcal{O}(f) \\ +\infty & f \notin \mathcal{O}(g) \text{ et } g \in \mathcal{O}(f) \\ \text{const.} & \Theta(f) = \Theta(g) \end{cases}$$
- *Multi-params.*: $\mathcal{O}, \Omega, \Theta$ étendues avec plusieurs seuils

Correction et terminaison

- *Correct*: sur *toute* entrée x qui satisfait la pré-condition, x et sa sortie y satisfont la post-condition
- *Termine*: atteint instruction **retourner** sur *toute* entrée
- *Invariant*: propriété qui demeure vraie à chaque fois qu'une ou certaines lignes de code sont atteintes

Exemples de complexité

$$\mathcal{O}(1) \subset \mathcal{O}(\log n) \subset \mathcal{O}(n) \subset \mathcal{O}(n \log n) \subset \mathcal{O}(n^2) \subset \mathcal{O}(n^2 \log n) \subset \mathcal{O}(n^3) \subset \mathcal{O}(n^d) \subset \mathcal{O}(2^n) \subset \mathcal{O}(3^n) \subset \mathcal{O}(b^n) \subset \mathcal{O}(n!)$$

2. Tri

Approche générique

- *Inversion*: indices (i,j) t.q. $i < j$ et $s[i] > s[j]$
- *Progrès*: corriger une inversion en diminue la quantité
- *Procédure*: sélectionner et corriger une inversion, jusqu'à ce qu'il n'en reste plus

Algorithmes (par comparaison)

- *Insertion*: considérer $s[1:i-1]$ triée et insérer $s[i]$ dans $s[1:i]$
- *Monceau*: transformer s en monceau et retirer ses éléments
- *Fusion*: découper s en deux, trier chaque côté et fusionner
- *Rapide*: réordonner autour d'un pivot et trier chaque côté

Propriétés

- *Sur place*: n'utilise pas de séquence auxiliaire
- *Stable*: l'ordre relatif des éléments égaux est préservé

Sommaire

Algorithme	Complexité (par cas) meilleur	moyen	pire	Sur place	Stable
insertion	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	✓	✓
monceau	$\Theta(n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	✓	✗
fusion	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	✗	✓
rapide	$\Theta(n)$	$\Theta(n \log n)$	$\Theta(n^2)$	✓	✗

Usage

- *Petite taille*: tri par insertion
- *Grande taille*: tri par monceau ou tri rapide
- *Grande taille + stabilité*: tri par fusion

Tri sans comparaison

- *Par comparaison*: barrière théorique de $\Omega(n \log n)$
- *Sans comparaison*: possible de faire mieux pour certains cas
- *Représentation binaire*: trier (de façon stable) en ordonnant du bit de poids faible vers le bit de poids fort
- *Complexité*: $\Theta(mn)$ où $m =$ nombre de bits et $n = |s|$

3. Graphes

Graphes

- *Graphe*: $\mathcal{G} = (V, E)$ où $V =$ sommets et $E =$ arêtes
- *Dirigé vs. non dirigé*: $\{u, v\} \in E$ vs. $(u, v) \in E$
- *Degré (cas non dirigé)*: $\deg(u) = \#$ de voisins
- *Degré (cas dirigé)*: $\deg^-(u) = \#$ préd., $\deg^+(u) = \#$ succ.
- *Taille*: $|E| \in \Theta($somme des degrés$)$ et $|E| \in \mathcal{O}(|V|^2)$
- *Chemin*: séq. $u_0 \to \cdots \to u_k$ (taille $= k$, simple si sans rép.)
- *Cycle*: chemin de u vers u (simple si sans rép. sauf début/fin)
- *Sous-graphe*: obtenu en retirant sommets et/ou arêtes
- *Composante*: sous-graphe max. où sommets access. entre eux

Parcours

- *Profondeur*: explorer le plus loin possible, puis retour (pile)
- *Largeur*: explorer successeurs, puis leurs succ., etc. (file)
- *Temps d'exécution*: $\mathcal{O}(|V| + |E|)$

Représentation

$$\begin{array}{c|ccc} & a & b & c \\ a & 0 & 1 & 1 \\ b & 1 & 0 & 0 \\ c & 0 & 1 & 0 \end{array} \quad \begin{array}{l} [a \mapsto [b,c], \\ b \mapsto [a], \\ c \mapsto [b]] \end{array}$$

	Mat.	Liste (non dirigé)	Liste (dir.)
$u \to v$?	$\Theta(1)$	$\mathcal{O}(\min(\deg(u), \deg(v)))$	$\mathcal{O}(\deg^+(u))$
$\{v : u \to v\}$	$\Theta(\lvert V\rvert)$	$\mathcal{O}(\deg(u))$	$\mathcal{O}(\deg^+(u))$
$\{u : u \to v\}$	$\Theta(\lvert V\rvert)$	$\mathcal{O}(\deg(v))$	$\mathcal{O}(\lvert V\rvert + \lvert E\rvert)$
Modif. $u \to v$	$\Theta(1)$	$\mathcal{O}(\deg(u) + \deg(v))$	$\mathcal{O}(\deg^+(u))$
Mémoire	$\Theta(\lvert V\rvert^2)$	$\Theta(\lvert V\rvert + \lvert E\rvert)$	

Propriétés et algorithmes

- *Plus court chemin*: parcours en largeur + stocker préd.
- *Ordre topologique*: $u_1 \preceq \cdots \preceq u_n$ où $i < j \implies (u_j, u_i) \notin E$
- *Tri topologique*: mettre sommets de degré 0 en file, retirer en mettant les degrés à jour, répéter tant que possible
- *Détec. de cycle*: tri topo. + vérifier si contient tous sommets
- *Temps d'exécution*: tous linéaires

Arbres

- *Arbre*: graphe connexe et acyclique (ou prop. équivalentes)
- *Forêt*: graphe constitué de plusieurs arbres
- *Arbre couv.*: arbre avec tous sommets d'un graphe $\mathcal{G}$ non dirigé; possible ssi $\mathcal{G}$ connexe; se trouve avec parcours en prof.

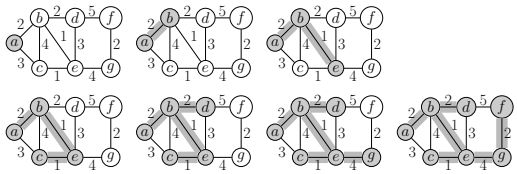
4. Algorithmes gloutons

Arbres couvrants minimaux

- *Graphe pondéré*: $\mathcal{G} = (V, E)$ où $p[e]$ est le poids de l'arête e
- *Poids d'un graphe*: $p(\mathcal{G}) = \sum_{e \in E} p[e]$
- *Arbre couv. min.*: arbre couvrant de $\mathcal{G}$ de poids minimal

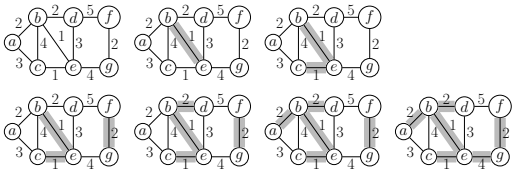
Algorithme de Prim–Jarník

- *Approche*: faire grandir un arbre en prenant l'arête min.
- *Complexité*: $\mathcal{O}(|E| \log |V|)$ avec monceau



Algorithme de Kruskal

- *Approche*: connecter forêt avec l'arête min. jusqu'à un arbre
- *Complexité*: $\mathcal{O}(|E| \log |V|)$ avec ensembles disjoints



Ensembles disjoints

- *But*: manipuler une partition d'un ensemble V
- *Représentation*: chaque ensemble sous une arborescence

Algorithme glouton

1) *Choisir* un candidat c itérativement (*sans reconsidérer*)
2) Ajouter c à solution partielle S si *admissible*
3) Retourner S si *solution (complète)*, « impossible » sinon

Problème du sac à dos

- *But*: choisir objets pour maxim. valeur sans excéder capacité
- *Algo. glouton*: trier en ordre décroissant par $val[i]/poids[i]$
- *Fonctionne* si on peut découper objets (sinon approx. à ½)