Chapitre 5 : Algorithmes gloutons

INF4705 - Analyse et conception d'algorithmes

Gilles Pesant Simon Brockbank

École Polytechnique Montréal
gilles.pesant@polymtl.ca, simon.brockbank@polymtl.ca

Hiver 2017

Plan

Faire de la monnaie, canadienne

Notre algorithme intuitif

fonction monnaie(P :ensemble {billets et pièces},
s :entier {somme à rendre}) :ensemble avec répétitions

    $S \leftarrow \emptyset$; {solution}
    tant que $s > 0$ **faire**
        $p \leftarrow$ plus grande pièce de P dont la valeur $\leq s$;
        $S \leftarrow S \cup \{p\}$;
        $s \leftarrow s - p$;
    retourner(S)

Et ça fonctionne ?

56$ à rendre, avec des 20$, 10$, 5$, 2$ et 1$

56$ à rendre, avec des 20$, 10$, 5$, 2$ et 1$

56$ à rendre, avec des 20$, 10$, 5$, 2$ et 1$

56$ à rendre, avec des 20$, 10$, 5$, 2$ et 1$

56$ à rendre, avec des 20$, 10$, 5$, 2$ et 1$

56$ à rendre, avec des 20$, 10$, 5$, 2$ et 1$

$$2 \times 20\$ \ + \ 1 \times 10\$ \ + \ 1 \times 5\$ \ + \ 1 \times 1\$ = 56\$$$

Un total de 5 billets/pièces, le plus petit nombre possible (optimal).

56$ à rendre, avec des 25$, 10$ et 1$

$$2 \times 25\$ \ + \ 6 \times 1\$ = 56\$$$

Un total de 8 billets et pièces ; mais aurait-on pu faire mieux ?

56$ à rendre, avec des 25$, 10$ et 1$

On aurait dû rendre $1 \times 25\$ + 3 \times 10\$ + 1 \times 1\$ = 56\$$
pour un total de 5 billets et pièces.

Ici notre algorithme n'est donc pas optimal.
Mais il y a encore pire !

56$ à rendre, avec des 20$, 10$, 5$ et 2$

$$2 \times 20\$ \ + \ 1 \times 10\$ \ + \ 1 \times 5\$ + \ ???$$

L'algorithme échoue !

56$ à rendre, avec des 20$, 10$, 5$ et 2$

Alors qu'on aurait pu rendre

$$2 \times 20\$ \ + \ 1 \times 10\$ \ + \ 3 \times 2\$ = 56\$$$

Principe Général des algorithmes gloutons

On procède par une succession de choix,
prenant toujours l'alternative semblant la meilleure à ce moment-là,
sans jamais revenir sur nos décisions.

Avantages

- simple à concevoir
- facile à implanter
  (pas de retour en arrière)
- rapide

Inconvénients

- plus ou moins confiné aux problèmes d'optimisation
- pas nécessairement optimal
- pourrait même ne pas retourner de solution

Patron de conception des algorithmes gloutons (voraces, «greedy»)

fonction glouton(C :ensemble {candidats}) : ensemble
    $S \leftarrow \emptyset$; {solution}
    tant que $C \neq \emptyset$ **et non** solution(S) **faire**
        $x \leftarrow$ choix(C) ;
        $C \leftarrow C \setminus \{x\}$;
        si non illégal($S \cup \{x\}$)
            alors $S \leftarrow S \cup \{x\}$;
    si solution(S)
        alors retourner(S)
        sinon retourner "aucune solution trouvée"

Patron de conception des algorithmes gloutons (voraces, «greedy»)

Patron

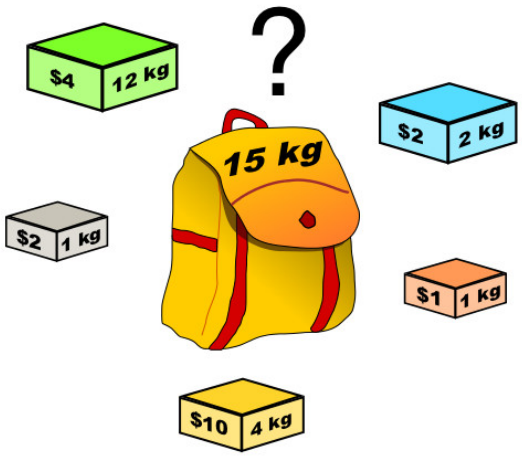
fonction glouton(C :ensemble {candidats}) : ensemble
    $S \leftarrow \emptyset$; {solution}
    tant que $C \neq \emptyset$ **et non** solution(S) **faire**
        $x \leftarrow$ choix(C) ;
        $C \leftarrow C \setminus \{x\}$;
        si non illégal($S \cup \{x\}$)
            alors $S \leftarrow S \cup \{x\}$;
    si solution(S)
        alors retourner(S)
        sinon retourner "aucune solution trouvée"

Faire de la monnaie

fonction monnaie(P :ensemble {billets et pièces}, s :entier {somme à rendre}) :ensemble avec répétitions
    $S \leftarrow \emptyset$; {solution}
    tant que $s > 0$ **faire**
        $p \leftarrow$ plus grande pièce de P dont la valeur $\leq s$;
        $S \leftarrow S \cup \{p\}$;
        $s \leftarrow s - p$;
    retourner(S)

Plan

Problème du sac à dos («knapsack»)



source : Wikipedia

Données

- n objets de poids et valeur positifs w_i et v_i, $1 \leq i \leq n$.
- un sac à dos pouvant supporter un poids total W
- $\sum_{i=1}^{n} w_i > W$, donc on ne peut pas y mettre tous les objets

Problème

Quels objets devrais-je mettre dans mon sac à dos afin de maximiser la valeur totale de son contenu ?

Formulation mathématique

Version originale

$$\begin{aligned} \max \sum_{i=1}^{n} v_i x_i \qquad & \text{tel que} \\ \sum_{i=1}^{n} w_i x_i \leq W & \\ x_i \in \{0,1\} \qquad & 1 \leq i \leq n \end{aligned}$$

- x_i indique si le $i^{\text{ème}}$ objet est choisi.
- «difficile» à résoudre
- On ne connaît pas (encore) d'algorithme pour le résoudre en temps polynomial.
- exemple de *programmation linéaire en nombres entiers*

Formulation mathématique

Version avec fractionnement

$$\begin{aligned} \max \sum_{i=1}^{n} v_i x_i \qquad & \text{tel que} \\ \sum_{i=1}^{n} w_i x_i \leq W & \\ 0 \leq x_i \leq 1 \qquad & 1 \leq i \leq n \end{aligned}$$

- Nous avons droit aux fractions d'objets : $0 \leq x_i \leq 1$.
- On parle de *relaxation continue* du problème.
- Cette version est facile à résoudre, notamment à l'aide d'un algorithme glouton.
- exemple de *programmation linéaire*

Pourquoi s'y intéresser ?

Quelques applications

- portefeuille d'investissements
- chargement de conteneurs
- découpe de rouleaux de matériel
- cryptographie à clef publique
- . . . une composante importante de plusieurs autres problèmes

Généralisations

- plusieurs dimensions
- plusieurs sacs
- . . .

Algorithme glouton pour le sac à dos, version avec fractionnement

- Nos candidats sont les objets.
- L'algorithme maintient le poids accumulé du sac.
- À chaque itération, on choisit le "meilleur" objet et on met la plus grande fraction possible de cet objet dans le sac.
- On s'arrête lorsque le sac est plein, ce qui arrivera nécessairement.

Comment choisir le meilleur objet (choix glouton) ?

Choix : plus léger d'abord (w_i croissant)

Objets :

w_i	10	20	30	40	50
v_i	20	35	50	36	50

$W = 90$

$20 + 35 + 50 + 3/4 \times 36 = 132\$$

Choix : plus précieux d'abord (v_i décroissant)

Objets :

w_i	10	20	30	40	50
v_i	20	35	50	36	50

$W = 90$

$50 + 50 + 1/4 \times 36 = 109\$$

Choix : densité de valeur décroissante

Objets :

w_i	10	20	30	40	50
v_i	20	35	50	36	50
v_i/w_i	2	1.75	1.67	0.9	1

$W = 90$

$20 + 35 + 50 + 3/5 \times 50 = 135\$$

Choix : densité de valeur décroissante

Objets :

w_i	10	20	30	40	50
v_i	20	35	50	36	50
v_i/w_i	2	1.75	1.67	0.9	1

$W = 90$

$20 + 35 + 50 + 3/5 \times 50 = 135\$$

Ce critère de choix est optimal
(pour la version avec fractionnement).

Choix : densité de valeur décroissante

Objets :

w_i	10	20	30	40	50
v_i	20	35	50	36	50
v_i/w_i	2	1.75	1.67	0.9	1

$W = 90$

$20 + 35 + 50 + 3/5 \times 50 = 135\$$

Ce critère de choix est optimal
(pour la version avec fractionnement).

Solution optimale sans fractionnement : 121\$ (objets 2, 3 et 4)

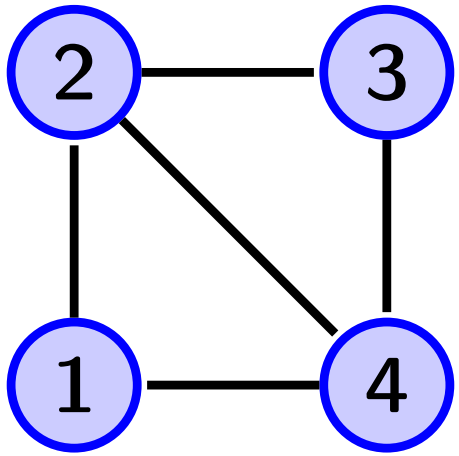
Analyse asymptotique (ici version originale)

fonction knapsack(C, v, w, W) : ensemble
    $S \leftarrow \emptyset$;
    ordonner C selon v_i/w_i décroissant
    tant que $C \neq \emptyset$ **et** $W > 0$ **faire**
        $x \leftarrow$ prochain élément de C ;
        $C \leftarrow C \setminus \{x\}$;
        si $w_x \leq W$ **alors**
            $S \leftarrow S \cup \{x\}$;
            $W \leftarrow W - w_x$;
    retourner(S)

Plan

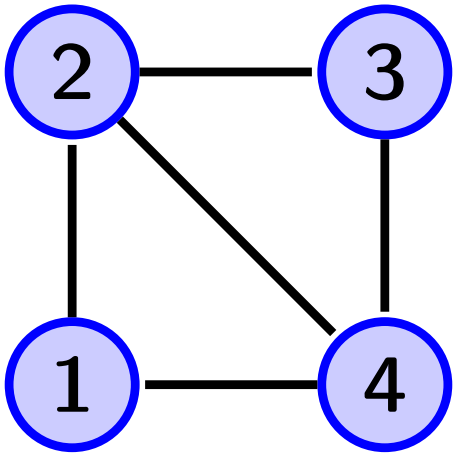
Rappels

- Soit $G = (S, A)$ un graphe composé d'un ensemble de *sommets* S et d'un ensemble A de paires de sommets (appelées *arêtes*).
- Un graphe est non orienté si ses arêtes ne sont pas ordonnées.



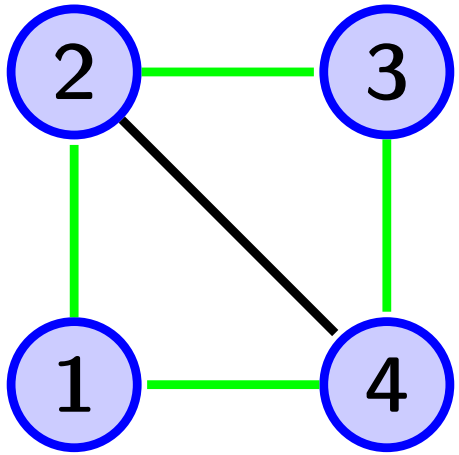
Rappels

- Un graphe est connexe si $\forall u, v \in S$, $\exists$ chaîne d'arêtes dans A joignant u et v.



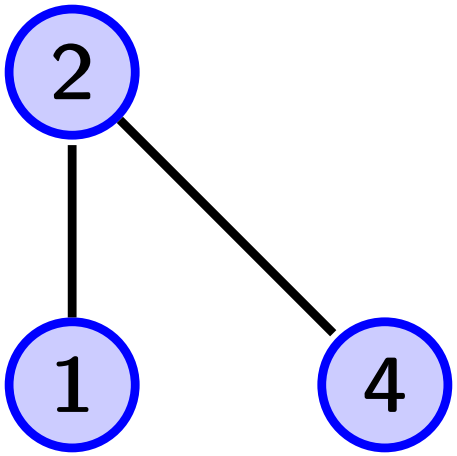
Rappels

- Un graphe est connexe si $\forall u, v \in S$, $\exists$ chaîne d'arêtes dans A joignant u et v.
- Un cycle est une chaîne d'au moins 3 arêtes joignant un sommet à lui-même.



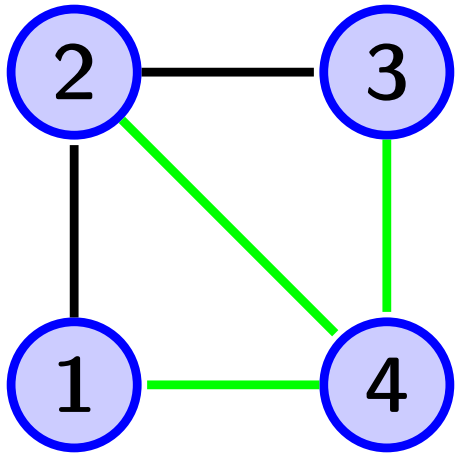
Rappels

- Un arbre est un graphe non orienté, connexe et sans cycle.



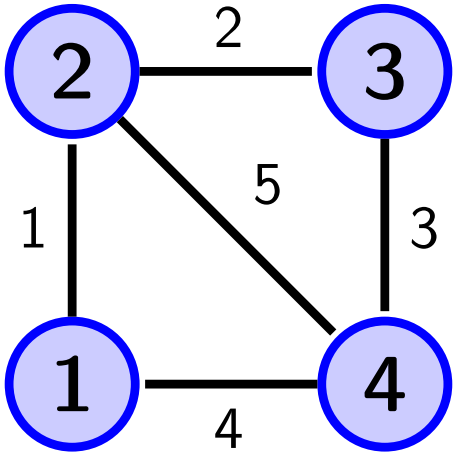
Rappels

- Un arbre est un graphe non orienté, connexe et sans cycle.
- Un arbre couvrant (ou *sous-tendant*) de G est un arbre $T = (S', A')$ tel que $S' = S$ et $A' \subseteq A$.



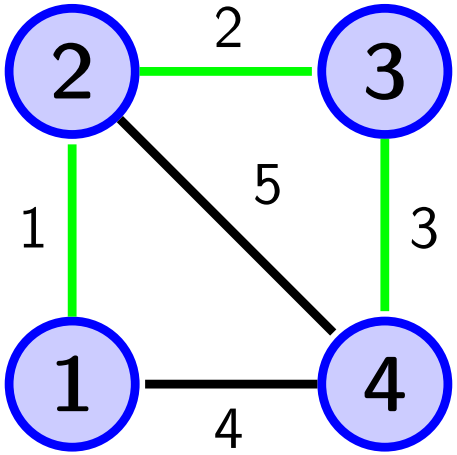
Rappels

- Un graphe pondéré associe une longueur à chaque arête.



Rappels

- Un graphe pondéré associe une longueur à chaque arête.
- Un arbre couvrant de poids minimum de G minimise la somme des longueurs des arêtes utilisées.



Pourquoi s'y intéresser ?

Quelques applications

- Établir un réseau (routier, informatique) à moindre coût entre un ensemble de sites
- Variantes beaucoup plus difficiles à résoudre : à degré limité, à diamètre limité, à consommation d'énergie équilibrée, ...
- une composante importante de plusieurs algorithmes (ex : algorithme d'approximation pour le TSP)

Pour un même problème,

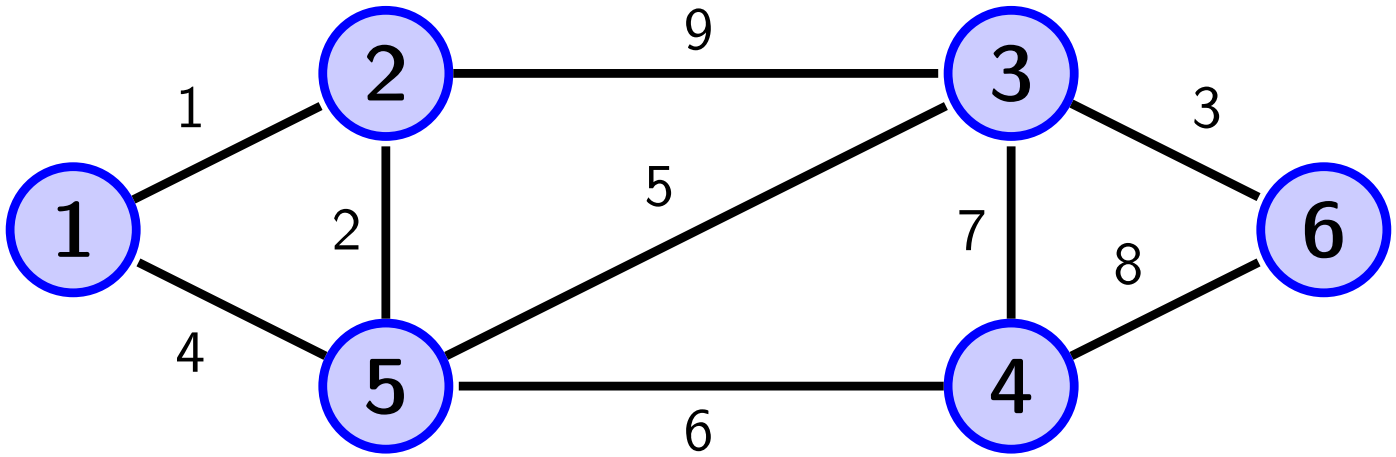
en utilisant le même patron de conception,

nous verrons deux algorithmes distincts.

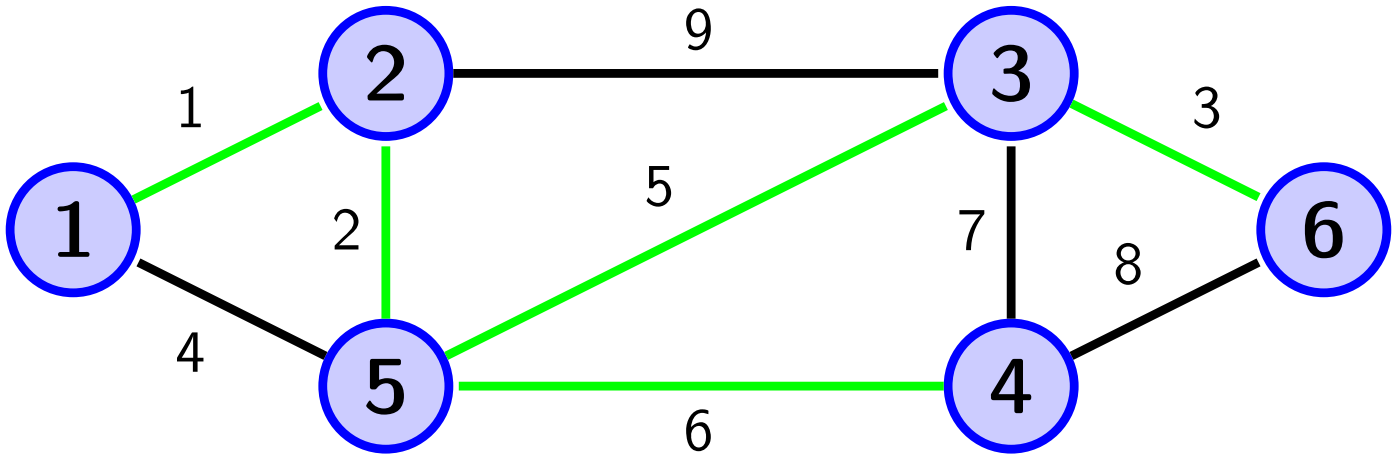
Algorithme de Kruskal

- Nos candidats sont les arêtes.
- L'algorithme maintient une *forêt* (un ensemble d'arbres).
- À chaque itération, on choisit une arête la plus courte :
  si elle joint deux arbres, on l'ajoute ;
  sinon on la rejette car elle créerait un cycle.
- On s'arrête lorsqu'on a plus qu'un seul arbre.

Exemple



Exemple



Description

fonction Kruskal($G = (S, A)$:graphe, $\ell : A \to \mathbb{R}_+$) : ensemble
    ordonner A selon ℓ croissant ;
    $n \leftarrow |S|$; $T \leftarrow \emptyset$;
    initialiser n ensembles, pour chaque élément de S ;
    répéter
        $(u, v) \leftarrow$ prochain élément de A ;
        $ucomp \leftarrow$ trouver(u) ;
        $vcomp \leftarrow$ trouver(v) ;
        si $ucomp \neq vcomp$ **alors**
            fusionner($ucomp$,$vcomp$) ;
            $T \leftarrow T \cup \{(u, v)\}$;
    jusque $|T| = n - 1$
    retourner T ;

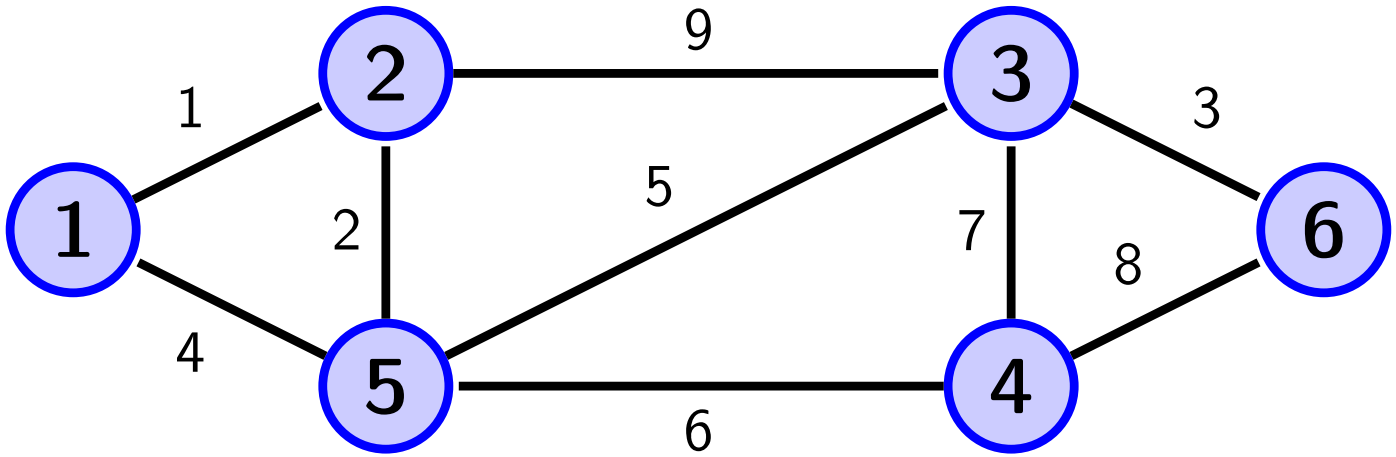
Analyse asymptotique

fonction Kruskal($G = (S, A)$:graphe, $\ell : A \to \mathbb{R}_+$) : ensemble
    ordonner A selon ℓ croissant ;
    $n \leftarrow |S|$; $T \leftarrow \emptyset$;
    initialiser n ensembles, pour chaque élément de S ;
    répéter
        $(u, v) \leftarrow$ prochain élément de A ;
        $ucomp \leftarrow$ trouver(u) ;
        $vcomp \leftarrow$ trouver(v) ;
        si $ucomp \neq vcomp$ **alors**
            fusionner($ucomp$,$vcomp$) ;
            $T \leftarrow T \cup \{(u, v)\}$;
    jusque $|T| = n - 1$
    retourner T ;

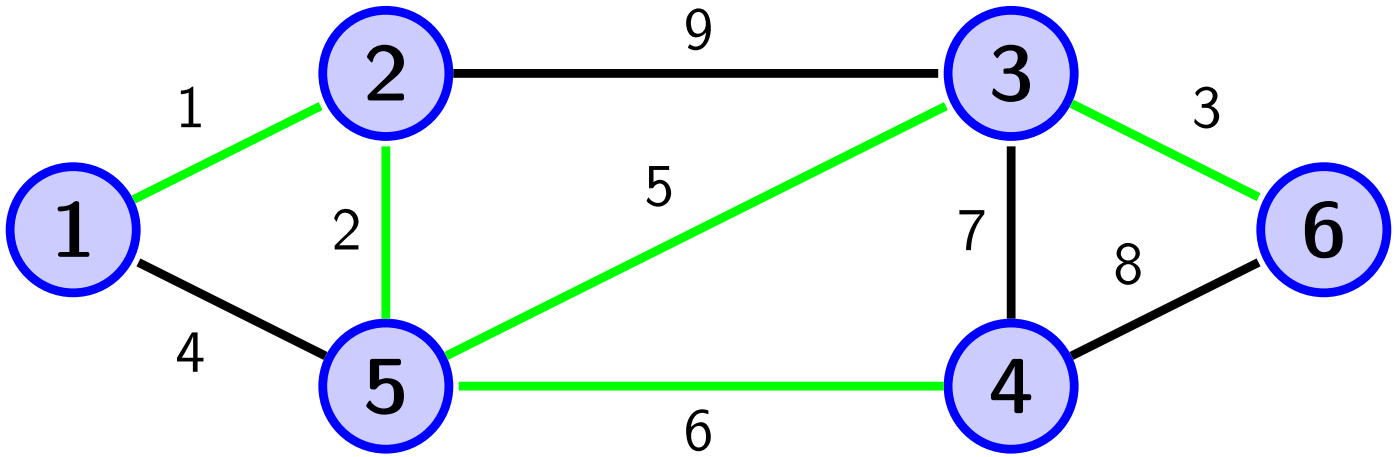
Algorithme de Prim

- Nos candidats sont les arêtes.
- L'algorithme maintient un seul arbre.
- À chaque itération, on choisit une nouvelle branche (arête) la plus courte.
- On s'arrête après $n - 1$ répétitions.

Exemple



Exemple



Description

fonction Prim($L[1..n, 1..n]$) : ensemble
    $T \leftarrow \emptyset$;
    pour $i = 2$ **à** n **faire**
        $voisin[i] \leftarrow 1$;
        $distmin[i] \leftarrow L[i, 1]$;
    répéter $n - 1$ **fois**
        $min \leftarrow \infty$;
        pour $j = 2$ **à** n **faire**
            si $0 \leq distmin[j] < min$ **alors**
                $min \leftarrow distmin[j]$;
                $k \leftarrow j$;
        $T \leftarrow T \cup \{(k, \text{voisin}[k])\}$;
        $distmin[k] \leftarrow -1$;
        pour $j = 2$ **à** n **faire**
            si $L[k, j] < distmin[j]$ **alors**
                $distmin[j] \leftarrow L[k, j]$;
                $voisin[j] \leftarrow k$;
    retourner T ;

Analyse asymptotique

fonction Prim($L[1..n, 1..n]$) : ensemble
    $T \leftarrow \emptyset$;
    pour $i = 2$ **à** n **faire**
        $voisin[i] \leftarrow 1$;
        $distmin[i] \leftarrow L[i, 1]$;
    répéter $n - 1$ **fois**
        $min \leftarrow \infty$;
        pour $j = 2$ **à** n **faire**
            si $0 \leq distmin[j] < min$ **alors**
                $min \leftarrow distmin[j]$;
                $k \leftarrow j$;
        $T \leftarrow T \cup \{(k, \text{voisin}[k])\}$;
        $distmin[k] \leftarrow -1$;
        pour $j = 2$ **à** n **faire**
            si $L[k, j] < distmin[j]$ **alors**
                $distmin[j] \leftarrow L[k, j]$;
                $voisin[j] \leftarrow k$;
    retourner T ;

Lequel Choisir ?

- Si le graphe est épars, $a \to n$ (connexe) : $\Theta(n^2)$ vs $\Theta(n \lg n)$.
- Si le graphe est dense, $a \to n^2(\binom{n}{2})$: $\Theta(n^2)$ vs $\Theta(n^2 \lg n)$.

Notes

1. On peut implanter l'algorithme de Kruskal à l'aide d'un monceau et éliminer ainsi le tri initial : ceci ne change pas le comportement en pire cas mais est avantageux si peu d'arêtes sont considérées.
2. On peut également implanter l'algorithme de Prim à l'aide d'un monceau $\Rightarrow \Theta(a \lg n)$.
3. Il existe des algorithmes plus performants.

Plan

1ère technique : Sous-structure optimale + choix glouton optimal

Propriété de sous-structure optimale

Une solution optimale à un exemplaire est composée de solutions optimales à des sous-exemplaires.

Propriété du choix glouton optimal

Il existe toujours une solution optimale qui contient le choix glouton qu'on s'apprête à faire.

Application : faire de la monnaie avec des 20$,10$,5$,2$,1$

Démonstration de la sous-structure optimale

Soit une solution optimale $\langle p_1, \ldots, p_k, p_{k+1}, \ldots, p_n \rangle$.

Considérons $\langle p_1, \ldots, p_k \rangle$ et $\langle p_{k+1}, \ldots, p_n \rangle$: chacune doit être une solution optimale pour la sous-somme correspondante, sinon on obtiendrait une meilleure solution en leur substituant une (sous-) solution utilisant moins de pièces. □

Application : faire de la monnaie avec des 20$,10$,5$,2$,1$

Observation

Dans toute solution optimale, on utilisera au plus :
1 pièce de 1$, 2 pièces de 2$, 1 billet de 5$ et 1 billet de 10$.

Démonstration du choix glouton optimal

- $S \geq 21$: choix $= 20\$$; pourrait-on avoir une solution optimale sans un tel billet ? Non car selon l'observation précédente, le total des autres billets et pièces ne dépasse pas 20.
- $S = 20$: choix $= 20\$$; clairement optimal.
- $19 \geq S \geq 11$: choix $= 10\$$; pourrait-on avoir une solution optimale sans un tel billet ? Non car . . . ne dépasse pas 10.
- . . .

Application : faire de la monnaie avec des 25$,10$,1$

Observation

Dans toute solution optimale, on utilisera au plus :
9 pièces de 1$ et 4 billets de 10$.

Choix glouton optimal ?

- $S \geq 26$: choix $= 25\$$; pourrait-on avoir une solution optimale sans un tel billet ? **Oui !** Par exemple pour $S = 30$, remettre trois billets de 10$ est la solution optimale.

2e technique : Matroïdes

Définition

Un matroïde M est un couple $(E, \mathcal{I})$ qui vérifie ces conditions :

1. E est un ensemble fini ;
2. $\mathcal{I}$ est une famille non vide de sous-ensembles de E telle que si $B \in \mathcal{I}$ et $A \subseteq B$ alors $A \in \mathcal{I}$ ($\mathcal{I}$ est *héréditaire*) ;
3. Si $A \in \mathcal{I}$, $B \in \mathcal{I}$ et $|A| < |B|$ alors $\exists x \in B \setminus A$ tel que $A \cup \{x\} \in \mathcal{I}$ (M vérifie la *propriété d'échange*).

Définition

Un matroïde pondéré s'adjoint une fonction $w : E \to \mathbb{R}_+$
qui associe un poids positif à chaque élément de E.
Par extension, $w(A) = \sum_{x \in A} w(x)$ pour $A \in \mathcal{I}$.

Algorithmes gloutons appliqués aux matroïdes

Patron glouton spécialisé pour matroïdes

fonction glouton(M, w) : ensemble
    $S \leftarrow \emptyset$; {solution}
    tant que $E \neq \emptyset$ **faire**
        $x \leftarrow \arg\max_{e \in E} w(e)$;
        $E \leftarrow E \setminus \{x\}$;
        si $S \cup \{x\} \in \mathcal{I}$
            alors $S \leftarrow S \cup \{x\}$;
    retourner(S)

Théorème

Si $\langle M, w \rangle$ est un matroïde pondéré alors glouton(M, w) retourne un sous-ensemble optimal (c.-à-d. de poids maximum).

Un matroïde sur les graphes

Soit $G = (S, A)$ un graphe non orienté. Définissons $M_G = (E, \mathcal{I})$:

- $E = A$
- $F \in \mathcal{I}$ ssi F est acyclique (c.-à-d. une forêt)

M_G est un matroïde

1. E est clairement fini.
2. $\mathcal{I}$ est héréditaire : le sous-ensemble d'une forêt est une forêt.
3. M_G vérifie la propriété d'échange :
   Une forêt $F = (S_F, A_F)$ contient $|S_F| - |A_F|$ arbres. (voir suivante)
   Soient $F_1, F_2 \in \mathcal{I}$ t.q. $|F_1| < |F_2|$: F_1 a plus d'arbres que F_2.
   $\exists$ arbre $T \subseteq F_2$ avec sommets dans arbres distincts de F_1.
   $\exists (u, v) \in T$ avec de tels sommets.
   $F_1 \cup \{(u, v)\} \in \mathcal{I}$ puisqu'aucun cycle ne sera créé. □

Toute forêt de a arêtes sur n sommets contient $n - a$ arbres.

Démonstration

Appelons t ce nombre d'arbres et dénotons respectivement par n_i et a_i le nombre de sommets et d'arêtes du $i^{\text{ème}}$ arbre.

$$a = \sum_{i=1}^{t} a_i = \sum_{i=1}^{t} (n_i - 1) = \sum_{i=1}^{t} n_i - t = n - t$$

Et donc $t = n - a$. □

Application : algorithme de Kruskal

Soit $w(e) = (\max_{a \in A} \ell(a) + 1) - \ell(e), \quad \forall e \in E.$

Kruskal, version glouton sur matroïde

fonction Kruskal(M_G, w) : ensemble
    $S \leftarrow \emptyset$; {solution}
    tant que $E \neq \emptyset$ **faire**
        $x \leftarrow \arg\max_{e \in E} w(e)$;
        $E \leftarrow E \setminus \{x\}$;
        si $S \cup \{x\}$ n'introduit pas un cycle
            alors $S \leftarrow S \cup \{x\}$;
    retourner(S)

Donc Kruskal retourne un arbre couvrant optimal (de poids min.).