INF4705 — Analyse et conception d'algorithmes

CHAPITRE 5 : ALGORITHMES VORACES

Exemple 1 *Faire de la monnaie, canadienne*

fonction *monnaie(P :ensemble {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 ne dépasse pas* s ;
        $S \leftarrow S \cup \{p\}$;
        $s \leftarrow s - p$;
    retourner(S)

Pourquoi est-ce que ça fonctionne ?

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

+ simple à concevoir
+ facile à implanter (pas de prévision globale ou de retour en arrière)
+ efficace
− plus ou moins confiné aux problèmes d'optimisation
− et même là, d'application limitée (lorsque donne une solution suboptimale)

Schéma général :

fonction vorace(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"

5.1 Arbre sous-tendant (ou encore "couvrant") minimum

Définitions :

- Soit $G = (N, A)$ un *graphe* composé d'un ensemble de *sommets* N et d'un ensemble A de paires de sommets (appelées *arêtes*).
- Un graphe est *non orienté* si chaque arête n'est pas ordonnée ((a,b) est la même paire que (b,a)).
- Un graphe est *connexe* si $\forall u, v \in N$, $\exists$ 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.
- Un *arbre* est un graphe non orienté, connexe et sans cycle. Un arbre *sous-tendant* de G est un arbre $T = (N', A')$ tel que $N' = N$ et $A' \subseteq A$.
- Associons une longueur à chaque arête de G. Un *arbre sous-tendant minimum* de G a la plus petite somme des longueurs des arêtes utilisées.

Utilité : établir un réseau à moindre coût entre un ensemble de sites, algo. approximatif pour le pb du voyageur de commerce, ...

5.1.1 Algorithme de Kruskal

idée :
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.

fonction Kruskal($G = (N, A)$:graphe, *longueur* $: A \to \mathbb{R}_+$) : ensemble
  trier A par *longueur* croissante ;
  $n \leftarrow |N|$; $T \leftarrow \emptyset$;
  initialiser n ensembles, chacun contenant un élément différent de N ;
  répéter
    $\langle u, v\rangle \leftarrow$ plus courte arête non encore considérée ;
    $ucomp \leftarrow$ trouver(u) ;
    $vcomp \leftarrow$ trouver(v) ;
    si $ucomp \neq vcomp$ **alors**
      fusionner($ucomp$,$vcomp$) ;
      $T \leftarrow T \cup \{\langle u, v\rangle\}$;
  jusque $|T| = n - 1$
  retourner T ;

Analyse :
- tri : $\Theta(a \lg a) = \Theta(a \lg n)$
- $\Theta(1), \Theta(1), \Theta(n)$
- boucle : $\leq a$ itérations de $\Theta(1) + \cdots$
  structure d'ensembles disjoints (cf Brassard,Bratley §5.9 (forêt de faible hauteur)) :
  x trouver() et y fusionner() $\Rightarrow \Theta((x + y) \cdot \alpha(x + y, n))$.
  Dans notre cas, $x \leq 2a$ et $y = n - 1 \Rightarrow \mathcal{O}(a \cdot \alpha(2a + n - 1, n))$. Puisque $\alpha(2a + n - 1, n) \in \mathcal{O}(\lg n)$, le temps pris par la boucle est dans $\mathcal{O}(a \lg n)$.

Donc $\Theta(a \lg n)$.

5.1.2 Algorithme de Prim (mais dû à quelqu'un d'autre)

idée :
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.

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**
$min \leftarrow distmin[j]$;
$k \leftarrow j$;
$T \leftarrow T \cup \{\langle k, \text{voisin}[k]\rangle\}$;
$distmin[k] \leftarrow -1$;
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 :
- 1 ère boucle : $n-1$ itérations d'opérations élémentaires $\Rightarrow \Theta(n)$.
- 2 ème boucle : utilisons "$L[k,j] < distmin[j]$" comme opération baromètre ; $(n-1)\cdot(n-1)\ \Theta(1) \Rightarrow \Theta(n^2)$
(Mais attention à la structure de données utilisée pour "$T \leftarrow T \cup \{\langle k, voisin[k]\rangle\}$".)

Donc $\Theta(n^2)$

5.1.3 Lequel choisir ?

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

5.1.4 Preuves d'optimalité

Qu'est-ce qui nous garantit que ces algorithmes trouvent un arbre sous-tendant qui soit minimum ?
Un ensemble d'arêtes est *prometteur* si on peut le compléter pour obtenir un arbre sous-tendant minimum.
Une arête (x,y) *laisse* un ensemble de sommets B si $x \in B$ et $y \notin B$.

Lemme 1 *Soient $B \subset N$, $T \subseteq A$ prometteur t.q. $\nexists u \in T$ laissant B, v une plus courte arête laissant B. Alors $T \cup \{v\}$ est prometteur.*

Démonstration 1
Soit $U \supseteq T$ un arbre sous-tendant minimum (existe puisque T prometteur).
Si $v \in U$, il n'y a rien à démontrer.
Sinon, $U \cup \{v\}$ crée exactement un cycle et il doit exister $u \in U$ laissant B tout comme v.
Or, u ne peut pas être plus courte que v donc $U \setminus \{u\} \cup \{v\}$ est un arbre sous-tendant minimum complétant $T \cup \{v\}$ puisque $T \subseteq U$ et $u \notin T$.
Donc $T \cup \{v\}$ est prometteur. $\surd$

On prouve l'optimalité des deux algorithmes par induction sur le nombre d'arêtes dans T. On montre que T est toujours prometteur.

Théorème 1 *L'algorithme de Prim produit un arbre sous-tendant minimum.*

Démonstration 2

base : *L'ensemble vide est prometteur.*

pas d'induction : *Supposons T prometteur et soit B l'ensemble de ses sommets.*
- *$B \subset N$ sinon l'algorithme a terminé,*
- *T prometteur et aucune de ses arêtes ne laisse B, par l'hypothèse d´induction et la défn de B,*
- *v est une plus courte arête et laisse B,*

donc le lemme s'applique et le nouveau T ($T \cup \{v\}$) est prometteur.

√

Théorème 2 *L'algorithme de Kruskal produit un arbre sous-tendant minimum.*

Démonstration 3

Semblable à la preuve précédente, en prenant pour B l'ensemble des sommets d'un des deux arbres joints par la nouvelle arête. √

5.2 Plus court chemin

$G = (N, A)$ est maintenant orienté ; A est un ensemble d'*arcs*. Les arcs ont une longueur non négative. Étant donné un sommet *source*, quelle est la longueur du plus court chemin de cette source à chacun des autres sommets ?

Utilité : Lorsqu'on veut connaître la façon la plus économique d'aller de A à B, très fréquent comme sous-problème.

Remarque : Le calcul du plus court chemin entre une paire de sommets en particulier ne semble pas plus facile.

5.2.1 Algorithme de Dijkstra

idée :

Nos candidats sont les sommets.

L'algorithme maintient un ensemble spécial de sommets, S (initialement {source})
ainsi qu'un tableau des longueurs du plus court chemin *spécial* (c.-à-d. dont les sommets intermédiaires sont tous dans S) de la source à chacun des autres sommets.

À chaque itération, on choisit puis ajoute à S le sommet qui offre le plus court chemin spécial de la source. On s'arrête après $n-2$ répétitions.

fonction Dijkstra($L[1..n, 1..n]$) : tableau$[2..n]$
  $C \leftarrow \{2, 3, \ldots, n\}$; {implicitement $S = N \setminus C$}
  pour $i = 2$ **à** n **faire**
    $D[i] \leftarrow L[1, i]$;
  répéter $n-2$ **fois**
    $v \leftarrow$ l´élément de C qui minimise $D[v]$;
    $C \leftarrow C \setminus \{v\}$;
    pour chaque élément w de C **faire**
      $D[w] \leftarrow \min(D[w], D[v] + L[v, w])$;
  retourner D ;

Note :

Pour obtenir les chemins, ajouter un tableau donnant pour chaque sommet le sommet le précédant sur son plus court chemin.

Analyse :
- initialisation : $\Theta(n)$.
- boucle : $(n-1) + (n-2) + \cdots + (2) = \Theta(n^2), (n-2) \cdot \Theta(1), (n-2) + (n-3) + \cdots + (1) = \Theta(n^2)$.

Donc $\Theta(n^2)$.

À noter qu'on peut obtenir une meilleure complexité asymptotique pour G épars si on utilise un tableau de listes pour les longueurs (semblable à une représentation par liste d'adjacences) plutôt qu'un tableau à deux dimensions et un monceau pour les candidats C, avec pour valeur les $D[i]$:

- initialisation : idem.
- boucle : $(n-2)\cdot\Theta(1), \sum_{i=n-1}^{2} \lg i \in \Theta(n \lg n), \Theta(a)\cdot\mathcal{O}(\lg n)$ (serait $\Theta(n^2)\cdot\mathcal{O}(\lg n)$ sans le tableau de listes).
  $\Rightarrow \mathcal{O}((n+a)\cdot\lg n) = \mathcal{O}(a \lg n)$ si G est connexe.

Théorème 3 *L'algorithme de Dijkstra trouve les plus courts chemins d'une source vers tous les autres sommets.*

Démonstration 4

Prouvons par induction sur le nombre de sommets dans S que :
(a) $\forall$sommet $i \in S$ (autre que la source), $D[i]$ contient la longueur du plus court chemin de la source à i ;
(b) $\forall$sommet $i \notin S$, $D[i]$ contient la longueur du plus court chemin spécial de la source à i.

base : *$S = \{source\}$, (a) trivialement vérifié et (b) découle de l'initialisation de $D[]$.*

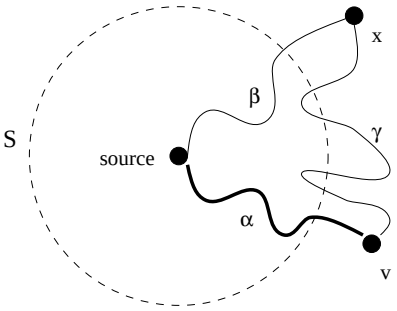
hypothèse d'induction : *(a) et (b) sont vérifiés pour S juste avant l'ajout d'un sommet v.*

pas d'induction pour (a) :

$\forall i$ déjà dans S, $D[i]$ ne change pas alors (a) vérifié par H.I.(a).
Pour v nouvellement dans S, nous devons nous assurer que $D[v]$ donne la longueur du plus court chemin.
Par H.I.(b), $D[v]$ = longueur plus court chemin spécial.
Examinons donc les chemins passant à l'extérieur de S : soit x le premier sommet $\notin S$ sur un tel chemin.

1. *β est un chemin spécial donc $\ell(\beta) = D[x]$ par H.I.(b)*
2. *Puisque v et non pas x a été choisi, $D[x] \geq D[v]$.*
3. *$\ell(\gamma) \geq 0$ puisque les longueurs sont non négatives.*
4. *Donc $\ell(\alpha) = D[v] \leq D[x] = \ell(\beta) \leq \ell(\beta) + \ell(\gamma)$.*

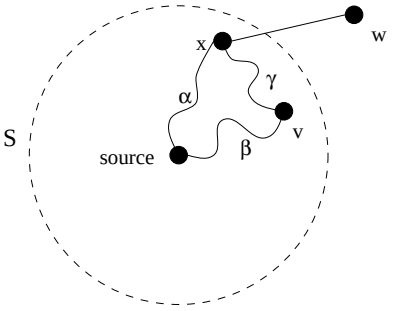
Conséquemment $D[v]$ = longueur plus court chemin.



pas d'induction pour (b) :

Pour un sommet $w \notin S$, la seule façon d'obtenir un chemin spécial plus court est de passer par v : soit x le dernier sommet $\in S$ sur un tel chemin.

1. *Si $x \neq v$, $D[x] = \ell(\alpha) \leq \ell(\beta) + \ell(\gamma)$, par H.I.(a).*
   Donc $\ell(\beta)+\ell(\gamma)+L[x,w] \geq D[x]+L[x,w] \geq D[w]$ puisqu'on a déjà considéré $D[x]+L[x,w]$ au moment de l'ajout de x à S.
   Ainsi passer par v ne donne pas plus court.
2. *Sinon, v est ce dernier sommet et le chemin spécial considéré a pour longueur $D[v]+L[v,w]$, ce que l'algorithme vérifie.*



À la fin de l'algorithme, tous les sommets (sauf un) sont dans S et donc par (a) il fonctionne correctement. De même pour le sommet restant, par (b) (plus court chemin spécial $\equiv$ plus court chemin). $\surd$

5.3 Sac à dos

Étant donné n objets $\{1, \ldots, n\}$, de poids et valeur positifs w_i et v_i resp., et un sac à dos pouvant supporter W, quels objets devrais-je mettre dans mon sac à dos afin de maximiser la valeur totale ?
Le problème n'est intéressant que si $\Sigma_{i=1}^{n} w_i > W$.
VERSION SIMPLIFIÉE :
Nous avons droit aux fractions d'objets $0 \leq x_i \leq 1$.
idée :
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 ? Selon

1. poids croissant
2. valeur décroissante
3. densité de valeur (v_i/w_i) décroissante

Théorème 4 *La stratégie v_i/w_i décroissante mène l'algorithme vorace à une solution optimale.*

Démonstration 5
- *Supposons $v_1/w_1 \geq v_2/w_2 \geq \cdots \geq v_n/w_n$.*
- *Soit $X = (x_1, \ldots, x_n)$ la soln trouvée par l'algorithme et $V(X)$ sa valeur.*
- *Soit j le plus petit indice t.q. $x_j < 1$: on aura donc $X = (1, \ldots, 1, x_j, 0, \ldots, 0)$.*
- *Soit $Y = (y_1, \ldots, y_n)$ une autre soln et $V(Y)$ sa valeur.*

$$\begin{aligned} V(X) - V(Y) &= \Sigma_{i=1}^{n} x_i v_i - \Sigma_{i=1}^{n} y_i v_i \\ &= \Sigma_{i=1}^{n} (x_i - y_i) v_i \\ &= \Sigma_{i=1}^{n} (x_i - y_i) w_i \frac{v_i}{w_i} \end{aligned}$$

3 cas :

$i < j$ **:** $x_i = 1$ *donc* $(x_i - y_i) \geq 0$ *et* $v_i/w_i \geq v_j/w_j$ *ainsi* $(x_i - y_i)v_i/w_i \geq (x_i - y_i)v_j/w_j$.

$i > j$ **:** $x_i = 0$ *donc* $(x_i - y_i) \leq 0$ *et* $v_i/w_i \leq v_j/w_j$ *ainsi* $(x_i - y_i)v_i/w_i \geq (x_i - y_i)v_j/w_j$.

$i = j$ **:** $(x_i - y_i)v_i/w_i = (x_i - y_i)v_j/w_j$.

$$\begin{aligned} V(X) - V(Y) &\geq \frac{v_j}{w_j} \Sigma_{i=1}^{n} (x_i - y_i) w_i \\ &= \frac{v_j}{w_j} (\Sigma_{i=1}^{n} x_i w_i - \Sigma_{i=1}^{n} y_i w_i) \\ &= \frac{v_j}{w_j} (W - \Sigma_{i=1}^{n} y_i w_i) \\ &\geq 0 \text{ , puisque } \Sigma_{i=1}^{n} y_i w_i \leq W \end{aligned}$$

Donc $\nexists$ soln meilleure que X. $\surd$

Analyse :
- tri : $\Theta(n \lg n)$.
- boucle : $\Theta(n)$.

Donc $\Theta(n \lg n)$.
Note :
Ici aussi, l'utilisation d'un monceau ne change pas le comportement en pire cas mais est avantageux si peu d'objets sont requis pour remplir le sac.

VERSION ORIGINELLE :
Beaucoup plus difficile! Une approche vorace ne garantit plus la solution optimale.

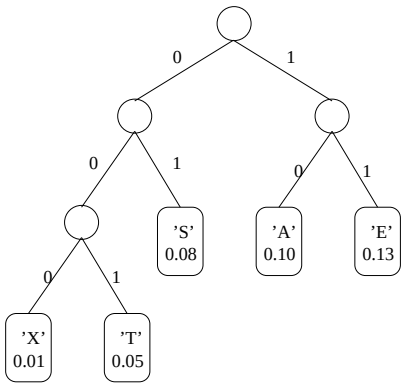
5.4 Codes de Huffman

Étant donné un ensemble de messages $M = \{m_1, m_2, \ldots, m_n\}$, chacun avec une probabilité de transmission p_i, déterminer un code binaire distinct pour chaque message tel que la longueur espérée d´un message codé est minimale.

Utilité : codage, compression de données
idée :
Nos candidats sont des arbres binaires, initialement les messages.
À chaque itération, on choisit les deux arbres de plus faible probabilité et on les fusionne en leur donnant une racine commune.
On s'arrête une fois qu´on a un seul arbre.
Les codes sont obtenus en étiquettant les branches de gauche dans l´arbre avec un ¨0¨ et celles de droite avec un ¨1¨. Il suffit de lire les bits de la racine à la feuille contenant le message à coder.

fonction Huffman(M) : arbre binaire
  $L \leftarrow \{\langle m_1, p_1\rangle, \ldots, \langle m_n, p_n\rangle\}$; {liste initiale d´arbres binaires avec leur probabilité}
  répéter $n-1$ **fois**
    $a_i \leftarrow$ l´élément de L qui minimise p_i ;
    $L \leftarrow L \setminus \{\langle a_i, p_i\rangle\}$;
    $a_j \leftarrow$ l´élément de L qui minimise p_j ;
    $L \leftarrow L \setminus \{\langle a_j, p_j\rangle\}$;
    $a \leftarrow$ l´arbre dont la racine a pour enfants a_i et a_j ;
    $L \leftarrow L \cup \{\langle a, p_i + p_j\rangle\}$;
  retourner L ;

Exemple 2



AS : 1001

TEXAS : 001110001001

TASSE : 00110010111

Comment décode-t-on une suite de bits ?

Analyse et preuve d´optimalité laissées en exercice.