Mines-Ponts 2002
Corrigé de l'épreuve d'informatique
(durée de l'épreuve : 3 heures)

1 Exercice de logique des propositions

1 – Notons F_1, F_2 et F_3 les trois formules étudiées. Alors :

- F_1 est satisfiable, en choisissant par exemple la valuation qui associe *vrai* aux trois variables p_i.
- F_2 n'est pas satisfiable car pour que la formule soit vraie, il faudrait que p_2 et $\neg p_2$ soient simultanément vérifiées.
- F_3 n'est également pas satisfiable. En effet, $(\neg p_1 \vee \neg p_2) \wedge (p_1 \vee \neg p_2)$ est logiquement équivalente à $\neg p_2$, et donc F_3 est logiquement équivalente à $p_2 \wedge (\neg p_2) \wedge (p_1 \vee \neg p_2 \vee \neg p_3)$ qui n'est clairement pas satisfiable.

2 – Soit H une formule de Horn et considérons la valuation v associant la valeur $faux$ à toutes les variables p_i. Nous pouvons écrire $H = C_1 \wedge \ldots \wedge C_m$ où $m \geq 0$ et où les C_i sont des clauses comportant au moins un littéral négatif. Pour tout i, nous avons donc $v(C_i) = vrai$, puis $v(H) = vrai$ (en remarquant que cela marche, par convention, même si $m = 0$). H est donc satisfiable.

3 – Soit H une formule de Horn de la forme $H = p_k \wedge C_2 \ldots \wedge C_m$, avec $C_i \neq \neg p_k$ pour tout i compris entre 2 et m. Pour chacun de ces i, soit C'_i définie de la façon suivante :

- si p_k est un littéral de C_i, alors $C'_i = T$;
- sinon, C'_i est la clause obtenue en supprimant de C_i les éventuels littéraux de la forme $\neg p_k$.

Soit enfin H' la formule de Horn conjonction des C'_i distincts de T : $H' = \displaystyle\bigwedge_{i,\ C'_i \neq T} C'_i$. Par construction, nous avons bien $\mathcal{V}_{H'} \subset \mathcal{V}_H$. Montrons que H et H' sont simultanément satisfiable.

- si H est satisfiable, soit v une valuation telle que $v(H) = vrai$. Nous avons alors $v(p_k) = vrai$, et $v(C_i) = vrai$ pour tout i. Si i est tel que $C'_i \neq T$, C_i est égal (à l'ordre près des littéraux) ou à $C'_i \wedge (\neg p_k)$, ou à C'_i. Comme $v(\neg p_k) = faux$, nous avons dans les deux cas $v(C'_i) = v(C_i) = vrai$, puis $v(H') = vrai$: H' est satisfiable.
- si H' est satisfiable, soit v une valuation telle que $v(H') = vrai$. Comme p_k n'est pas une variable de H', nous pouvons (quitte à changer $v(p_k)$) supposer que $v(p_k) = vrai$. Pour i compris entre 2 et m, deux cas se présentent :
  - ou bien C_i contient le littéral p_k, et alors $v(C_i) = vrai$;
  - ou bien C_i ne contient pas le littéral p_k, et alors C_i est de la forme $C'_i \wedge (\neg p_k)$ ou C'_i, puis $v(C_i) = v(C'_i) = vrai$.

Nous obtenons donc $v(H) = vrai$ et H est satisfiable.

4 – Soit $H = C_1 \wedge \ldots \wedge C_m$ une formule de Horn. Pour déterminer la satisfiabilité de H, on utilise l'algorithme récursif suivant :

- si $H = T$, alors H est satisfiable ;
- sinon, on cherche i tel que la clause C_i ne contienne pas de littéraux négatifs. Si un tel i n'existe pas, H est satifiable d'après la question (2). Sinon, la clause C_i est de la forme p_k où $1 \leq k \leq n$, puisque les clauses d'une formule de Horn contiennent au plus un littéral positif. Si l'une des autres clauses de H est réduite à $(\neg p_k)$, la formule H n'est pas satisfiable. Sinon, on applique récursivement l'algorithme à la formule de Horn H' construite en appliquant la question (3) (on sait que H est satisfiable si et seulement si H' l'est).

Cet algorithme se termine car, en cas d'appel récursif, l'ensemble $\mathcal{V}_{H'}$ est strictement contenu dans $\mathcal{V}_H$. Dans le pire des cas, au bout de n appel récursif, on appliquera l'algorithme à la formule T.

Il est difficile de majorer le temps de calcul sans préciser le type de donnée utilisé pour stocker les formules. Nous supposerons dans cette question qu'un littéral est un couple (i, b) où i est un entier et b un booléen, $(i, vrai)$ et $(i, faux)$ représentant respectivement p_i et $\neg p_i$. Une clause sera alors une liste de littéraux, et une formule sous forme normale conjonctive sera une liste de clauses. Pour traiter le problème, on commence à tester si l'une des clauses est de la forme p_k, ce qui demande un nombre d'opérations de l'orde de m (on teste si une des m listes est de taille 1, et associée à un littéral de la forme $(k, vrai)$). Si la recherche échoue, le calcul est terminé. Sinon, il faut encore de l'ordre de m opérations pour vérifier si l'une des autres clauses est de la forme $\neg p_k$. Si c'est le cas, le calcul est terminé en un temps $\Theta(m)$. Sinon, il faut définir la formule H', ce qui demande au plus de l'ordre de nm opérations (on parcourt les m clauses, qui sont des listes de longueurs au plus $2n$). Comme $n(H') \leq n(H) - 1$, nous obtenons en notant $T(n, m)$ le temps de calcul dans les pire des cas pour traiter une formule de Horn contenant n variables et m clauses :

$$T(n, m) \leq Knm + T(n-1, m)$$ [1]

Ainsi, nous avons :

$$T(n,m) \leq Knm + T(n-1,m) \leq Km\big(n+n-1\big) + T(n-2,m) \leq \ldots \leq Km\big(n+n-1+n-2+\cdots+1\big) + T(0,m)$$

puis $T(n, m) = O(mn^2)$ puisque $T(0, m) = constante$ (c'est le temps de vérifier que la formule est associée à la liste vide).

5 – On remarque que la clause C_1 est réduite à p_2 et que ni C_2, ni C_3 ne sont réduites à $(\neg p_2)$. On pose donc $H' = (\neg p_1) \wedge (p_1) \wedge (p_1 \vee \neg p_3) = C'_1 \wedge C'_2 \wedge C'_3$. Comme $C'_2 = p_1$ et $C'_1 = (\neg p_1)$, H' n'est pas satisfiable et H ne l'est également pas.

2 Problème d'algorithmique

1 – a Le problème commence par une question très délicate, en tout cas si l'on veut donner une preuve précise. La situation est plus facile à visualiser quand elle est proposée sous la forme équivalente suivante : un voleur s'est introduit dans l'arrière boutique d'un joaillier. Il est en présence de certaines quantités de métaux précieux :

1. On pourrait aussi remplacer $T(n-1, m)$ par $T(n-1, m-1)$ puisque H' contient au moins une clause de moins que H, mais cela complique le calcul.

p_0 grammes du métal O_0, p_1 grammes du métal O_1, ..., p_{n-1} grammes du métal O_{n-1}. On suppose ici que les métaux sont classés par valeurs décroissantes, la valeur étant le prix du gramme de métal. Il ne peut (malheureusement ?) dérober qu'une masse maximale Q, et il cherche donc un remplissage optimal de son sac à dos. Sous cette forme, le commun des mortel est convaincu que le gain maximal s'obtient en utilisant un algorithme *glouton* : on prend en priorité le métal O_0 (dans sa totalité si $p_0 \leq Q$), puis le métal O_1 (dans sa totalité si $p_0 + p_1 \leq Q$) et ainsi de suite, en s'arrêtant quand le sac à dos est plein (cas où $i^* < n$), ou bien quand tous les métaux ont été pris dans leur totalité (cas où $i^* = n$). Est-ce que cette façon de traduire le problème (même si elle est peu civique) suffira à convaincre le correcteur ? J'aurais plutôt tendance à croire que non, mais c'est peut-être une déformation de mathématicien. Nous allons donc donner une preuve “précise” de ce résultat.

Première méthode

L'idée naturelle consiste à démontrer que si $(y_0, \ldots, y_{n-1})$ est une solution non nulle du problème, il existe une autre solution $(z_0, \ldots, z_{n-1})$ vérifiant :

- il existe $j^* \in \{0, \ldots, n\}$ tel que $z_i = 1$ pour $i < j^*$, $z_i = 0$ pour $i > j^*$ et $z_{j^*} \neq 0$;
- $\displaystyle\sum_{i=0}^{n-1} u_i z_i \geq \sum_{i=0}^{n-1} u_i y_i$.

Une fois cette propriété démontrée, nous aurons :

- $\sum_{i=0}^{j^*-1} p_i = \sum_{i=0}^{j^*-1} p_i z_i < \sum_{i=0}^{j^*} p_i z_i \leq Q$ donc $j^* \leq i^*$;
- si $j^* < i^*$, $\sum_{i=0}^{n-1} u_i z_i = \sum_{i=0}^{j^*} u_i z_i \leq \sum_{i=0}^{j^*} u_i \leq \sum_{i=0}^{i^*-1} u_i \leq \sum_{i=0}^{n-1} u_i x_i$;
- si $j^* = i^*$, $\sum_{i=0}^{n-1} p_i z_i = \sum_{i=0}^{i^*-1} p_i + p_{i^*} z_{i^*} \leq Q$, donc $z_{i^*} \leq \dfrac{Q - \sum_{i=0}^{i^*-1} p_i}{p_{i^*}} = x_{i^*}$. On en déduit :

$$\sum_{i=0}^{n-1} u_i z_i = \sum_{i=0}^{i^*-1} u_i + u_{i^*} z_{i^*} \leq \sum_{i=0}^{i^*-1} u_i + u_{i^*} x_{i^*} = \sum_{i=0}^{n-1} u_i x_i.$$

Ainsi, nous aurons $\sum_{i=0}^{n-1} u_i y_i \leq \sum_{i=0}^{n-1} u_i z_i \leq \sum_{i=0}^{n-1} u_i x_i$, ce qui achèvera de démontrer que l'algorithme fournit une solution optimale au problème du sac à dos fractionnaire.

La construction de cette solution (z_i) se fait récursivement. Supposons en effet que (y_i) ne soit pas de la forme $(1, \ldots, 1, y_{j^*}, 0, \ldots, 0)$. C'est donc qu'il existe deux indices i_0 et i_1 tels que :

- $0 \leq i_0 < j_0 \leq n-1$;
- $y = (1, \ldots, 1, y_{i_0}, 0, \ldots, 0, y_{i_1}, \ldots, y_{n-1})$;
- $y_{i_0} < 1$ et $y_{i_1} > 0$.

Nous allons alors “échanger” une partie de l'objet i_1 contre une partie de l'objet i_0, de sorte à remplacer ou bien y_{i_0} par 1, ou bien y_{i_1} par 0. Pour cela, posons $\widetilde{y}_i = y_i$ pour tout i élément de $\{0, 1, \ldots, n-1\} \setminus \{i_0, i_1\}$, $\widetilde{y}_{i_0} = y_{i_0} + \varepsilon_0$ et $\widetilde{y}_{i_1} = y_{i_1} - \varepsilon_1$, où ε_0 et ε_1 sont deux réels vérifiant les contraintes :

- $0 \leq \varepsilon_0 \leq 1 - y_{i_0}$;
- $0 \leq \varepsilon_1 \leq y_{i_1}$;
- $p_{i_0} \varepsilon_0 = p_{i_1} \varepsilon_1$.

Les deux premières contraintes assurent que l'on a bien $0 \leq \widetilde{y}_i \leq 1$ pour tout i. La troisième permet de conserver l'égalité des poids : $\sum_{i=0}^{n-1} p_i \widetilde{y}_i = \sum_{i=0}^{n-1} p_i y_i$. Ainsi, il faut choisir ε_0 et ε_1 vérifiant :

- $0 \leq \varepsilon_0 \leq 1 - y_{i_0}$;
- $0 \leq \varepsilon_0 \leq \dfrac{p_{i_1}}{p_{i_0}} y_{i_1}$;
- $\varepsilon_1 = \dfrac{p_{i_0}}{p_{i_1}} \varepsilon_1$.

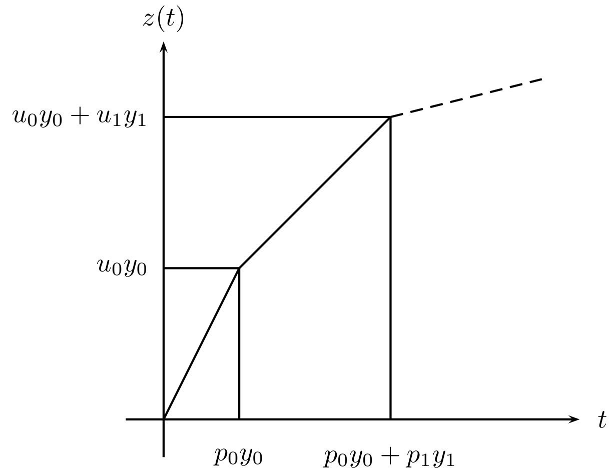
En choisissant $\varepsilon_0 = \min\left(1 - y_{i_0}, \dfrac{p_{i_1}}{p_{i_0}} y_{i_1}\right)$ puis $\varepsilon_1 = \dfrac{p_{i_0}}{p_{i_1}} \varepsilon_0$, nous aurons ainsi construit une solution $(\widetilde{y}_i)$ répondant à la question posée. En effet, selon la valeur du minimum définissant ε_0, on a bien $\widetilde{y}_{i_1} = 1$ ou $\widetilde{y}_{i_1} = 0$. D'autre part, $(\widetilde{y}_i)$ est plus efficace que (y_i) :

$$
\begin{aligned}
\sum_{i=0}^{n-1} u_i \widetilde{y}_i &= \sum_{i=0}^{n-1} u_i y_i + u_{i_0} \varepsilon_0 - u_{i_1} \varepsilon_1 \\
&= \sum_{i=0}^{n-1} u_i y_i + u_{i_0} \varepsilon_0 - u_{i_1} \frac{p_{i_0}}{p_{i_1}} \varepsilon_0 \\
&= \sum_{i=0}^{n-1} u_i y_i + p_{i_0} \varepsilon_0 \underbrace{\left(\frac{u_{i_0}}{p_{i_0}} - \frac{u_{i_1}}{p_{i_1}} \right)}_{\geq 0} \\
&\geq \sum_{i=0}^{n-1} u_i y_i
\end{aligned}
$$

Nous pouvons ainsi répéter cette construction jusqu'à ce que l'on obtienne une suite (z_i) de la forme $(1, \ldots, 1, z_{j^*}, 0, \ldots, 0)$. En effet, l'algorithme se termine car à chaque application de la construction, l'un (au moins) des entiers i_0 ou j_0 augmente strictement.

Remarque : la preuve peut sembler compliquée, et on peut penser au premier abord qu'il serait plus simple de démontrer directement que si une solution (y_i) n'est pas de la forme $(1, \ldots, 1, y_{j^*}, 0, \ldots, 0)$, elle n'est pas optimale (en construisant une solution (z_i) vérifiant $\sum_{i=0}^{n-1} u_i z_i > \sum_{i=0}^{n-1} u_i y_i$). On pourrait alors conclure, un argument de compacité assurant l'existence d'une solution optimale. Malheureusement, cela ne fonctionne que si l'on suppose la suite u_i/p_i strictement décroissante. Il est bien sûr possible de se ramener à ce cas, en "regroupant" les objets de même rapport utilité/poids, mais là encore, il est délicat de donner des arguments précis. Le plus raisonnable serait de ne demander une preuve rigoureuse que dans le cas où la suite (u_i/p_i) est strictement décroissante.

Seconde preuve : considérons une solution (y_i). Nous allons imaginer que nous remplissons le sac à dos "en continu", la variable t mesurant le poids du sac. Nous obtenons ainsi un graphe représentant l'évolution de z en fonction de t, de la forme :



Si nous notons f_y la fonction $t \mapsto z(t)$, il est clair que f_y est une fonction affine par morceaux, définie sur l'intervalle $[0, P_y]$ avec $P_y = \sum_{i=0}^{n-1} p_i y_i$, et dont les points anguleux sont les points (t_i, z_i) définis pour $0 \le i \le n$ par :

$$t_i = \sum_{j=0}^{i-1} p_j y_j \qquad z_i = \sum_{j=0}^{i-1} u_j y_j,$$

certains points pouvant être confondus si certains y_i sont nuls. L'intérêt de cette fonction est qu'elle fait apparaître naturellement les quotients u_i/p_i, qui sont les pentes des segments formant le graphe de f_y : la décroissance de ces pentes montre que f_y est une fonction concave.

Notons maintenant f_x la fonction associée à la solution (x_i) fournie par l'algorithme glouton. Pour tout i compris entre 0 et n, nous noterons $(t'_i, z'i)$ le point défini par :

$$t'_i = \sum_{j=0}^{i-1} p_j x_j \qquad z'_i = \sum_{j=0}^{i-1} u_j x_j.$$

Pour montrer que la solution (x_i) est optimale, il suffit de démontrer que $f_x(P_x)$ est supérieur ou égal à $f_y(P_y)$. Pour cela, il suffit de remarquer les deux propriétés :

- $P_x \ge P_y$;en effet, nous sommes dans l'un des deux cas suivants :
  - $P_x = Q$ et $P_y \le Q = P_x$;
  - $P_x < Q$ et x est la suite constante égale à 1. On a encore $P_y = \displaystyle\sum_{i=0}^{n-1} p_i y_i \le \sum_{i=0}^{n-1} p_i = P_x$.
- $f_y(t) \le f_x(t)$ pour tout $t \in [0, P_y]$;en effet, les fonctions f_y et f_x sont dérivables, sauf peut-être en les points $(t_0, \ldots, t_n, t'_0, \ldots, t'_n)$, et si t est un réel compris entre 0 et P_y et distinct des t_i et des t'_i, nous avons $f'_y(t) = \dfrac{u_i}{p_i}$ et $f'_x(t) = \dfrac{u_j}{p_j}$ où i et j sont caractérisés par :

$$t_i < t < t_{i+1} \quad \text{et} \quad t'_j < t < t'_{j+1}.$$

Or $t'_j = \sum_{k=0}^{j-1} p_k x_k = \sum_{k=0}^{j-1} p_i$ (car $x_j \neq 0$), donc $t'_j \geq \sum_{k=0}^{j-1} p_i y_i = t_j$. On en déduit donc que $t > t_j$, i.e. que $i \leq j$, ce qui donne $f'_y(t) = \dfrac{u_i}{p_i} \geq \dfrac{u_j}{p_j} = f'_x(t)$.

Comme $f_x(0) = 0 = f_y(0)$, on a bien $f_y(t) \leq f_x(t)$ pour tout $t \in [0, P_y]$.

Nous obtenons enfin $f_y(P_y) \leq f_x(P_y) \leq f_x(P_x)$ puisque f_x est croissante.

1 − b Remarquons tout d'abord que $\dfrac{16}{15} > \dfrac{21}{22} > \dfrac{19}{20} > \dfrac{15}{17} > \dfrac{13}{15} > \dfrac{7}{9}$. Nous avons ensuite :

$$p_0 + p_1 = 37 < 51 \leq 57 = p_0 + p_1 + p_2$$

donc $i^* = 2$ et la solution optimale est obtenue pour :

$$x_0 = x_1 = 1,\ x_2 = \frac{51-37}{20} = \frac{7}{10},\ x_3 = x_4 = x_5 = 0.$$

Le maximum obtenu est $z_{\max} = 16 + 21 + 19 \times \dfrac{7}{10} = \dfrac{503}{10}$.

1 − c On utilise simplement une variable booléenne `b` qui garde la valeur `true` tant que l'on a $u_{i-1}p_i \geq u_i p_{i-1}$. On obtient facilement la fonction `test` suivante :

```
let test () = let b=ref true and i=ref 1 in
    while (!b) & (!i)<n do
        b := (u.(!i-1)*p.(!i) >= u.(!i)*p.(!i-1));
        incr i;
     done;
     !b;;
```

La terminaison de cet algorithme est assurée puisque la variable `i` est incrémentée d'une unité à chaque passage dans la boucle et ne peut pas dépasser n. Enfin, la propriété :

$$\texttt{b} \text{ est équivalent à } \forall j \in \{1, \ldots, i-1\},\ \frac{u_{j-1}}{p_{j-1}} \geq \frac{u_j}{p_j}$$

est un invariant de boucle, ce qui prouve la correction de l'algorithme.

1 − d Pour calculer i^*, il suffit de calculer la somme $S = \sum_{j=0}^{i-1} p_i$ jusqu'à ce que l'on ait $i = n$ ou $S \geq Q$. Nous commençons donc pas initialiser la variable i à la valeur 1 et la variable S à la valeur p_0. Ensuite, nous ajoutons p_i à S et nous incrémentons i tant que $S < Q$ et $i < n$. À la sortie de la boucle, ou bien $S < Q$ et $i = n = i^*$, ou bien $S \geq Q$ et $i^* = i - 1$. Il ne reste donc qu'à écrire la fonction :

```
let istar () = let s=ref p.(0) and i=ref 1 in
   while (!i<n & !s<Q) do
      s := (!s)+p.(!i);
      incr i;
   done;
   if !s<Q then
       !i
   else
      (!i)-1;;
```

2 − a La recherche de i^* se fait de la façon suivante : on partitionne $\{0,1,\dots,n-1\}$ en deux sous-ensembles E' et E'' grâce à $\mathcal{A}$ de telle sorte que $\forall i \in E',\ \forall j \in E'',\ \dfrac{u_i}{p_i} \geq \dfrac{u_j}{p_j}$. On calcule ensuite $S = \displaystyle\sum_{i\in E'} p_i$, puis on distingue deux cas :

1^{er} cas : $S \geq Q$. On applique récursivement l'algorithme en remplaçant $\{0,1,\dots,n-1\}$ par E' ;on obtient ainsi i^*, J' et J'' tels que :

- $\{i^*\}$, J' et J'' partitionnent E' ;
- $\forall i \in J',\ \forall j \in J'',\ \dfrac{u_i}{p_i} \geq \dfrac{u_{i^*}}{p_{i^*}} \geq \dfrac{u_j}{p_j}$;
- $\displaystyle\sum_{i\in J'} p_i < Q \leq \sum_{i\in J'} p_i + p_{i^*}$.

On pose alors $I' = J'$, $I'' = J'' \cup E''$ et (i^*, I', I'') est une solution au problème posé.

2^{éme} cas : $S < Q$. On applique récursivement l'algorithme en remplaçant $\{0,1,\dots,n-1\}$ par E'' et Q par $Q-S$;on obtient i^*, J' et J'' tels que :

- $\{i^*\}$, J' et J'' partitionnent E'' ;
- $\forall i \in J',\ \forall j \in J'',\ \dfrac{u_i}{p_i} \geq \dfrac{u_{i^*}}{p_{i^*}} \geq \dfrac{u_j}{p_j}$;
- $\displaystyle\sum_{i\in J'} p_i < Q - S \leq \sum_{i\in J'} p_i + p_{i^*}$.

On pose alors $I' = J' \cup E'$, $I'' = J''$ et (i^*, I', I'') est une solution au problème posé.

Remarque : l'algorithme se termine correctement car la somme des p_i est supérieure ou égale à Q.

2 − b Nous sommes dans le cas classique d'un algorithme dichotomique : en notant $T(n)$ le temps de calcul dans le pire des cas pour traiter un problème de taille n, nous avons :

$$\forall m \geq 1,\ T(2^m) \leq K2^m + T(2^{m-1}),$$

le terme $K2^m$ représentant une majoration du temps de calcul de E', E'' et S (K est une constante). Nous en déduisons par une récurrence élémentaire :

$$\forall m \geq 1,\ T(2^m) \leq T(1) + K\sum_{i=1}^{m} 2^i \leq T(1) + 2K(2^m - 1),$$

ce qui prouve que l'algorithme est linéaire (en généralisant, comme proposé par l'énoncé, aux entiers n quelconques).

2 − c Il n'y a presque rien à dire : on calcule en un temps linéaire le triplet (i^*, I', I'') comme dans la question (a). Il reste à poser pour tout i dans $\{0,1,\dots,n-1\}$:

$$x_i = \begin{cases} 1 & \text{si } i \in I', \\ 0 & \text{si } i \in I'', \\ \dfrac{Q - \sum_{j\in I'} p_j}{p_{i^*}} & \text{si } i = i^*, \end{cases}$$

pour obtenir une solution optimale, par application de la question (1a).

Remarque : si l'on commence par trier les quantités u_i/p_i, le problème du sac à dos fractionnaire est résolu en un temps quasi-linéaire. Grâce à ce nouvel algorithme, nous obtenons la solution du problème en temps linéaire.

3 – Une solution du problème du sac à dos 0-1 est également une solution du problème du sac à dos fractionnaire : la solution optimale du sac à dos 0-1 est donc moins bonne que la solution optipale du sac à dos fractionnaire.

4 – **Remarque :** la stratégie proposée n'est pas, comme l'énoncé le dit, une stratégie de type *diviser pour régner*, mais simplement une étude de toutes les solutions réalisables en vue de calculer la solution optimale. C'est en fait un algorithme *naïf*, dont le temps de calcul dans le pire des cas est exponentiel.

Nous utilisons un vecteur $[|x_0; x_1; \ldots; x_{n-1}|]$, dont on modifie les valeurs au fur et à mesure de l'exploration des solutions. Pour cela, on crée deux vecteurs X et X_{opt} remplis de 0 : X sera le vecteur qui nous permettra de décrire toutes les solutions acceptables et X_{opt} permettra de retenir la meilleure solution déjà trouvée. L'utilité z de cette meilleure solution sera stockée dans la référence z_{max}. L'étude de toutes les solutions se fera à l'aide de la fonction récursive `exploration`; celle-ci prend en paramètre un entier $k \in \{0, 1, \ldots, n\}$, le poids *poids* et l'utilité z du sac-à-dos décrit par les valeurs $x_0, x_1, \ldots, x_{k-1}$ (c'est-à-dire $poids = \sum_{0 \leq i < k} x_i p_i$ et $z = \sum_{0 \leq i < k} x_i u_i$) et étudie les deux choix $x_k = 1$ ou $x_k = 0$, en mettant à jour (éventuellement) le vecteur X_{opt} et la référence z_{max}. Ces mises à jour se font quand $k = n$, i.e. quand la composition du sac-à-dos est connue, au moyen de la fonction `copier` qui permet de recopier X dans X_{opt}. Cela donne :

```
let copier X Y =
   for i=0 to (vect_length X) - 1 do
      Y.(i) <- X.(i)
   done;;

let sacados_bis() =
let Xopt = make_vect n 0 and zmax = ref 0 and X = make_vect n 0 in
let rec exploration k poids z = match k=n with
   | true -> if z > !zmax then
               begin     (* on a trouv\'e une solution meilleure que Xopt *)
               zmax := z;
               copier X Xopt;
               end
   | _ -> if poids + p.(k) <= Q then
                begin     (* si le poids le permet, on explore en ajoutant l'objet k *)
                X.(k) <- 1;
                exploration (k+1) (poids + p.(k)) (z + u.(k));
                X.(k) <- 0;
                end;
         exploration (k+1) poids z in  (* dans tous les cas, on explore sans l'objet k *)
   exploration 0 0 0;
   !zmax, Xopt;;
```

5 – Si $T(n)$ désigne le temps de calcul dans le pire des cas pour des données de taille n, nous avons

$$T(n) = 2T(n-1) + K_1 n + K_2$$

où K_1 et K_2 sont des constantes : $K_1 n + K_2$ représente le temps de recopie des n entrées de $\mathtt{x}$ auquel s'ajoute le temps (constant) pris pour effectuer les différents tests. En posant $\alpha_n = \dfrac{T(n)}{2^n}$, nous obtenons :

$$\alpha_n = \frac{T(n)}{2^n} = \alpha_{n-1} + \frac{K_1 n + K_2}{2^n} = \ldots = T(0) + \sum_{k=1}^{n} \frac{K_1 k + K_2}{2^k}.$$

Comme la série de terme général $\dfrac{K_1 k + K_2}{2^k}$ converge, la suite α_n a une limite non nulle L. On en déduit que $T(n)$ est équivalent à $L\,2^n$ quand n tend vers l'infini : le temps de calcul est exponentiel (c'était évident puisque dans le pire des cas, il y a 2^n suites réalisables, qui doivent toutes être étudiées).

6 − a On a $p_0 + p_1 = 37 < 51$. Comme $51 - 37 = 14$ est strictement inférieur à p_2, p_3 et p_4, la meilleure solution réalisable contenue dans le sommet C est $(1, 1, 0, 0, 0, 1)$: sa valeur est égale à $u_0 + u_1 + u_5 = 44$.

6 − b $p_0 + p_2 = 35 < 51$. Comme $p_3 = 17 > 51 - 35$, on est obligé de choisir $x_3 = 0$. Par contre, $p_4 = 15 \leq 51 - 35$. On peut donc choisir $x_4 = 1$ ou $x_4 = 0$. En traitant de la même façon la variable x_5, on obtient trois solutions réalisables contenues dans E : $(1, 0, 1, 0, 1, 0)$, $(1, 0, 1, 0, 0, 1)$ et $(1, 0, 1, 0, 0, 0)$, de valeurs respectivement $u_0 + u_2 + u_4 = 48$, $u_0 + u_2 + u_5 = 42$ et $u_0 + u_2 = 35$. La meilleure solution réalisable contenue dans E est donc $(1, 0, 1, 0, 1, 0)$, de valeur 48.

6 − c On applique la question (1a), en remarquant que $15/17 \geq 13/15 \geq \dfrac{7}{9}$: $x_3 = x_4 = 1$ et $x_5 = \dfrac{4}{9}$ donne la solution fractionnaire optimale, avec $15x_3 + 13x_4 + 7x_5 = 31 + \dfrac{1}{9}$. On en déduit que $M = u_0 + 31 + \dfrac{1}{9} = 47 + \dfrac{1}{9}$ est une évaluation du sommet F.

On en déduit que les solutions réalisables contenues dans F sont strictement moins bonnes que la solution réalisable $(1, 0, 1, 0, 1, 0)$: la solution optimale du problème $(\mathcal{P})$ n'est donc pas contenue dans F.

6 − d La méthode est la même que précédemment : la solution maximale du problème fractionnaire défini pour le sommet G est $(x_1, x_2, x_3, x_4, x_5) = (1, 1, \dfrac{9}{17}, 0, 0)$. La quantité $M = u_1 + u_2 + \dfrac{9}{17} u_3 = 47 + \dfrac{16}{17}$ est donc une évaluation du sommet G. Cette valeur étant également strictement inférieure à 48, le sommet G ne contient pas de solution réalisable optimale.

6 − e Les calculs précédents prouvent que le problème $\mathcal{P}$ a pour solution optimale la suite $x = (1, 0, 1, 0, 1, 0)$, avec $\displaystyle\sum_{i=0}^{5} u_i x_i = 48$ et $\displaystyle\sum_{i=0}^{5} p_i x_i = 48 \leq 51$.

Remarque : dans ces dernières questions, le problème parle de **la** solution réalisable optimale alors qu'*a priori* il pourrait exister plusieurs solutions réalisables optimales.

3 Exercice sur les automates finis

1 − Soit $\mathcal{A} = < Q, A, E, I, T >$ un automate reconnaissant L. Soit alors $\mathcal{A}_\mathrm{n} = < Q_\mathrm{n}, A, E_\mathrm{n}, I_\mathrm{n}, T_\mathrm{n} >$ défini de la façon suivante :

a) $Q_\mathrm{n} = Q \cup \{i, t\}$ où i et t sont deux éléments distincts n'appartenant pas à Q ;

b) $I_{\mathrm{n}} = \{i\}$ et $T_{\mathrm{n}} = \{t\}$;

c) $E_{\mathrm{n}} = E \cup \{(i, a, q),\ \exists\, i_0 \in I,\ (i_0, a, q) \in E\} \cup \{(q, a, t),\ \exists\, t_0 \in T,\ (q, a, t_0) \in E\}$

$$\cup\{(i, a, t),\ \exists\, i_0 \in I,\ \exists\, t_0 \in T,\ (i_0, a, t_0) \in E\}$$

Autrement dit, pour chaque transition (q_0, a, q_1) de $\mathcal{A}$, on définit dans l'automate $\mathcal{A}_{\mathrm{n}}$:

- une transition si $q_0 \notin I$ et $q_1 \notin T$: la transition (q_0, a, q_1) ;
- deux transitions si $q_0 \in I$ et $q_1 \notin T$: les transitions (q_0, a, q_1) et (i, a, q_1) ;
- deux transitions si $q_0 \notin I$ et $q_1 \in T$: les transitions (q_0, a, q_1) et (q_0, a, t)
- quatre transitions si $q_0 \in I$ et $q_1 \in T$: les transitions (q_0, a, q_1), (i, a, q_1), (q_0, a, t) et (i, a, t).

Par construction, l'automate $\mathcal{A}_{\mathrm{n}}$ est normalisé. Il reste donc à montrer qu'il reconnaît le langage $L \setminus \{1_{A^*}\}$:

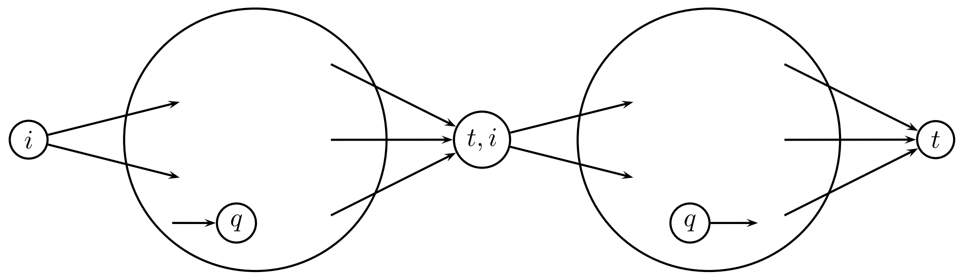
- soit $u \in L(\mathcal{A})$ différent du mot vide, et soit $i_0 \xrightarrow[\mathcal{A}]{a_1} q_1 \xrightarrow[\mathcal{A}]{a_2} q_2 \ldots \xrightarrow[\mathcal{A}]{a_{n-1}} q_{n-1} \xrightarrow[\mathcal{A}]{a_n} t_0$ un calcul réussi d'étiquette u. Alors $i \xrightarrow[\mathcal{A}_{\mathrm{n}}]{a_1} q_1 \xrightarrow[\mathcal{A}_{\mathrm{n}}]{a_2} q_2 \ldots \xrightarrow[\mathcal{A}_{\mathrm{n}}]{a_{n-1}} q_{n-1} \xrightarrow[\mathcal{A}_{\mathrm{n}}]{a_n} t$ est un calcul réussi de $\mathcal{A}_{\mathrm{n}}$ d'étiquette u et $u \in L(\mathcal{A}_{\mathrm{n}})$.

- soit $u \in L(\mathcal{A}_{\mathrm{n}})$ et soit $i \xrightarrow[\mathcal{A}_{\mathrm{n}}]{a_1} q_1 \xrightarrow[\mathcal{A}_{\mathrm{n}}]{a_2} q_2 \ldots \xrightarrow[\mathcal{A}_{\mathrm{n}}]{a_{n-1}} q_{n-1} \xrightarrow[\mathcal{A}_{\mathrm{n}}]{a_n} t$ est un calcul réussi d'étiquette u. On peut remarquer que les états $q_1, \ldots, q_{n-1}$ sont des éléments de E, puisqu'ils sont à la fois origine et extrémité de transitions de l'automate normalisé $\mathcal{A}_{\mathrm{n}}$. D'autre part, $n \geq 1$ et $u \neq 1_{A^*}$ car $i \neq t$. On montre ensuite que $u \in L$ en distinguant deux cas :
  - si $n = 1$, $u = a_1 \in A$ et $i \xrightarrow[\mathcal{A}_{\mathrm{n}}]{a_1} t$;il existe donc $i_0 \in I$ et $t_0 \in T$ tels que $i_0 \xrightarrow[\mathcal{A}]{a_1} t_0$: $u \in L$;
  - si $n \geq 2$, il existe $i_0 \in I$ et $t_0 \in T$ tels que $i_0 \xrightarrow[\mathcal{A}]{a_1} q_1$ et $q_{n-1} \xrightarrow[\mathcal{A}]{a_n} t_0$. On en déduit donc que $i_0 \xrightarrow[\mathcal{A}]{a_1} q_1 \xrightarrow[\mathcal{A}]{a_2} q_2 \ldots \xrightarrow[\mathcal{A}]{a_{n-1}} q_{n-1} \xrightarrow[\mathcal{A}]{a_n} t_0$ est un calcul réussi de $\mathcal{A}$ d'étiquette u : $u \in L$.

2 – Considérons une copie $\mathcal{A}' =< Q', A, E', I', T' >$ de l'automate $\mathcal{A}$;autrement dit, les ensembles Q et Q' sont disjoints et il existe une bijection φ de Q sur Q' telle que $\varphi(I) = I'$, $\varphi(T) = T'$ et $E' = \{(\varphi(p), a, \varphi(q)),\ p, q \in Q,\ a \in A,\ (p, a, q) \in E\}$. Soit alors $\mathcal{A}_q =< Q_q, A, E_q, E_q, I_q, T_q >$ l'automate défini comme suit :

- Q_q est la réunion de Q et de Q', dans laquelle les états t et $\varphi(i)$ ont été identifiés ;
- $I_q = \{q\}$;
- $T_q = \{\varphi(q)\}$;
- $E_q = E \cup E'$ (avec un abus d'écriture dû à l'identification de t et de $\varphi(i)$).

En reprenant le schéma proposé par l'énoncé, l'automate $\mathcal{A}_q$ admet la représentation suivante (φ n'apparaît pas pour ne pas alourdir le schéma) :



On peut également remarquer que les états i et $\varphi(t)$ peuvent être supprimés (i n'est pas accessible et $\varphi(t)$ n'est pas co-accessible). $\mathcal{A}_q$ est alors un automate (non normalisé) qui reconnaît le langage G_q. En effet, un calcul réussi dans cet automate passera une et une seule fois par l'état $\{t, \varphi(i)\}$, et sera donc de la forme :

$$q \xrightarrow[\mathcal{A}_q]{v} \{t, \varphi(i)\} \xrightarrow[\mathcal{A}_q]{u} \varphi(q).$$

On a alors par définition de E_q un calcul $q \xrightarrow[\mathcal{A}]{v} t$ et un calcul $i \xrightarrow[\mathcal{A}]{u} q$. On en déduit qu'il existe un calcul réussi de $\mathcal{A}$: $i \xrightarrow[\mathcal{A}]{u} q \xrightarrow[\mathcal{A}]{v} t$, et donc $vu \in G_q$.

Réciproquement, si un mot g est élément de G_q, il existe une factorisation $g = vu$ et un calcul réussi de $\mathcal{A}$ de la forme $i \xrightarrow[\mathcal{A}]{u} q \xrightarrow[\mathcal{A}]{v} t$. On en déduit que $q \xrightarrow[\mathcal{A}_q]{v} \{t, \varphi(i)\} \xrightarrow[\mathcal{A}_q]{u} \varphi(q)$ est un calcul réussi de $\mathcal{A}_q$, i.e. que g est élément de $L(\mathcal{A}_q)$.

3 – On a $\mathrm{Conj}(L) = L \cup \left(\bigcup_{\substack{q \in L \\ q \neq i,\ q \neq t}} G_q \right)$, donc $\mathrm{Conj}(L)$ est reconnaissable par automate, par stabilité de $\mathrm{Rec}(A)$ par union finie.