École polytechnique — Majeure Informatique
Promotion 2000

Cours de Conception et analyse d'algorithmes
Examen du mercredi 26 mars 2003. 2 heures
Documents autorisés : poly et notes de cours
Correction succincte

1. Algorithme glouton.

Algorithme.

Couleurs : entiers.
Trier les intervalles par dates de début croissantes : $s_1 \leq \ldots \leq s_n$.
Pour i allant de 1 à n
  donner à l'intervalle i la plus petite couleur qui ne crée pas de conflit
  avec les intervalles déjà coloriés.

Analyse.

Soit $i \mapsto f(i)$ le coloriage construit par l'algorithme, $i \mapsto g(i)$ un coloriage optimal, celui qui a le plus long préfixe en commun avec f sur la suite $(g(1), g(2), \ldots, g(n))$. Montrons par l'absurde que $f = g$.

Sinon, il existe un premier i_0 tel que $f(i_0) = c \neq g(i_0) = c'$. Comme les couleurs c et c' sont toutes deux compatibles avec les coloriages des intervalles $1, \ldots, i_0 - 1$, tous ceux de ces intervalles qui sont coloriés c ou c' sont déjà terminés à l'instant s_{i_0}. Par choix de notre ordre de numérotation, aucun des intervalles $i_0 + 1, \ldots, n$ n'est encore commencé à l'instant s_{i_0}. Soit h le coloriage obtenu à partir de g en échangeant les couleurs c et c' pour les intervalles $i \geq i_0$. L'observation ci-dessus montre que h est encore un coloriage valide, Par définition de notre algorithme, $c < c'$, donc h utlise autant de couleurs que g, donc h est optimal. Mais h a un plus long préfixe en commun avec f puisque $h(i_0) = f(i_0)$: ceci contredit la définition de g.

Complexité.

Cet algorithme peut être implémenté en $n \log n$: on commence par trier les $\{s_i\} \cup \{t_i\}$, puis on parcourt le temps par ordre croissant en maintenant dans une structure d'arbre équilibré les couleurs utilisées à l'instant considéré.

2. Programmation dynamique.

Utilisons un tableau T de booléens tel que $T[i, j]$ soit égal à vrai si et seulement s'il est possible de replier les i premiers segments du mètre de façon à tenir dans la boîte avec l'extrémité finale de ℓ_i en position j.

Initialisation :
$T[0, j] \leftarrow$ vrai pour tout $j \in \{0, 1, \ldots k\}$.
$T[i, j] \leftarrow$ faux pour tout $i \in \{1, \ldots, n\}$ et $j \in \{0, 1, \ldots k\}$.

Récurrence :
Pour i allant de 1 à n faire
    Pour j allant de 0 à $k - \ell_i$ faire
        si $T[i-1, j+\ell_i]$ alors $T[i,j] \leftarrow$ vrai
    Pour j allant de ℓ_i à k faire
        si $T[i-1, j-\ell_i]$ alors $T[i,j] \leftarrow$ vrai

Résultat :
$T[n,0] \vee T[n,1] \vee \ldots \vee T[n,k]$.

3. Flots.

Modélisation : réseau à $n+m+2$ sommets : le source s, n sommets $p_1, \ldots, p_n$, m sommets $s_1 \ldots s_m$, et le puits t.

Pour tout $j \in \{1, \ldots, n\}$ et tout $i \in \{1, \ldots, m\}$ tels que $p_j \in S_i$, on a un arc (p_j, s_i) de capacité 1. De plus, pour tout $j \in \{1, \ldots, n\}$, on a un arc (s, p_j) de capacité $\lceil \sum_{i:p_j \in S_i} 1/|S_i| \rceil$, et pour tout $i \in \{1, \ldots m\}$, on a un arc (s_i, t) de capacité 1.

On résoud un problème de flot maximal sur le réseau ainsi défini.

Analyse.

Puisque la coupe $\{s, p_1, \ldots, p_n, s_1, \ldots, s_m\}, \{t\}$ a capacité m, le flot maximal aura valeur au plus m. S'il a valeur exactement m, ce flot, qui par l'algorithme du cours a valeur entière sur chaque arc, associera exactement un conducteur à chaque S_i, et la contrainte de capacité sur les arcs (s, p_j) garantit que la répartition obtenue sera équitable.

Il existe évidemment un flot de valeur m, à savoir le flot *fractionnaire* qui fait passer exactement $1/|S_i|$ quantité de flot sur chaque arc (p_j, s_i) ; donc la flot maximal aura valeur exactement m, et le flot entier de valeur m obtenu par l'algorithme du cours donnera donc toujours une répartition équitable.

4. NP-complétude.

On vérifie facilement que $4SAT$ est dans NP, puisque SAT est dans NP et qu'il est facile de vérifier en outre que toutes les clauses de la formule SAT donnée en entrée ont taille 4.

Pour la NP-complétude, on fait une réduction à partir de $3SAT$. Soit $C_i = (x \vee y \vee z)$ la i-ième clause de la formule $3SAT$ F donnée en entrée. On crée quatre nouvelles variables t_i, u_i, v_i, w_i et neuf nouvelles clauses de taille quatre, $(x \vee y \vee z \vee \overline{t_i})$ et $(t_i \vee \overline{u_i} \vee v_i \vee w_i)$, $(t_i \vee u_i \vee v_i \vee w_i)$, $(t_i \vee \overline{u_i} \vee \overline{v_i} \vee w_i)$, $(t_i \vee u_i \vee \overline{v_i} \vee w_i)$, $(t_i \vee \overline{u_i} \vee v_i \vee \overline{w_i})$, $(t_i \vee u_i \vee v_i \vee \overline{w_i})$, $(t_i \vee \overline{u_i} \vee \overline{v_i} \vee \overline{w_i})$, $(t_i \vee u_i \vee \overline{v_i} \vee \overline{w_i})$. La conjonction des $9m$ clauses ainsi créées définit la formule $4SAT$ F'. Il est clair que F' se construit à partir de F en temps polynomial.

On observe que les 9 clauses associées à C_i ne peuvent être simultanément satisfaites que si $t_i =$ vrai et $(x \vee y \vee z) =$ vrai. Il est donc clair que F' est satisfiable si et seulement si F est satisfiable.

5. Algorithme d'approximation.

Soit $a'_i = \min\{a_i, 1\}$ et soit $A' = \{a'_1, \ldots, a'_n\}$. On observe qu'OPT a la même valeur pour A et pour A', et que l'algorithme de l'exercice conne les objets aux mêmes personnes pour A et pour A'. On peut donc supposer, dans notre analyse, que tous les a_i sont inférieurs ou égaux à 1.

L'algorithme optimal satisfait au plus $s = \sum_i a'_i$ personnes puisqu'il doit donner au moins 1 à chaque personne.

Si, par l'algorithme de l'exercice, la personne p reçoit les objets $a'_i, \ldots, a'_j$, alors

$$a'_i + \cdots + a'_{j-1} < 1 < a'_i + \cdots + a'_j,$$

et donc $a'_i + \cdots + a'_j < 2$, donc l'algorithme va donner des objets à strictement plus que $s/2$ personnes. Parmi ces personnes, seule la dernière pourrait ne pas être satisfaite. Donc le nombre de personnes satisfaites par l'algorithme est supérieur ou égal à $s/2$, donc à OPT/2.

6. Algorithme probabiliste.

(i).

```
i<- 0;   j<- 0;
tant que j<n faire
  etude de cas sur la valeur de (x(2i),x(2i+1)):
     cas   (0,0):  debut  z(j)<- 0; j++; i++; fin
     cas   (0,1):  debut  z(j)<- 1; j++; i++; fin
     cas   (1,0):  debut  z(j)<- 2; j++; i++; fin
     cas   (1,1):   i++
```

Complexité : un échec pour trois succès en moyenne, donc $4n/3$ passages dans la boucle en moyenne.

L'algorithme donne toujours le résultat souhaité, mais son temps d'exécution est aléatoire : c'est donc un algorithme de type Las Vegas

(ii).

```
succes <- faux; i<- 0;
tant que non succes faire
  etude de cas sur la valeur de (x(2i),x(2i+1)):
     cas   (0,1):  debut  z<- 0; succes<- vrai fin
     cas   (1,0):  debut  z<- 1; succes<- vrai fin
     autres cas :  i++
```

La probabilité qu'un passage dans la boucle soit couronné de succès est $2p(1-p)$, et le nombre de passages est donc $1/(2p(1-p))$ en moyenne.

Ici encore l'algorithme est de type Las Vegas.