Algorithmique II
Année 2006-2007

Florent Bouchez
Jean-François Pineau

TD n°9

Bzzz, bzzz, bzzz
Version avec solutions

Question 0.1. Rappel

Il y a n gentilles petites abeilles et m jolies petites fleurs. Chaque abeille a a peut être soit très attirée par une fleur soit pas du tout. Cet état vaut 0 ou 1, a été mesuré par un éromètre et est noté $\heartsuit(a, f)$. On suppose qu'aucune abeille n'est trop difficile pour n'aimer aucune jolie fleur, et qu'aucune fleur n'est trop repoussante au point de n'attirer aucune abeille. Comment maximiser l'amour cosmique entre les fleurs et les abeilles ?

Solution: *Avec des flots : orienter de X vers Y une source s avec des s vers $x \in X$ de capacité 1, un puit avec de $y \in Y$ vers p de capacité 1. Et les (x, y) de capacité $+\infty$.*

Exercice 1 Prouver le théorème suivant :

Théorème 1 *(König – Egerváry) Dans un monde merveilleux d'abeilles et de fleurs, l'amour cosmique maximal est le coût d'une couverture minimale des relations abeille/fleur possibles par une sélection d'abeilles et de fleurs.*

Solution:

Appliquer couplage max = max flot = min cut. Il existe une coupe de taille $\min(n, m)$. *Une coupe min coupe un groupe d'abeille V_A de la source et un groupe de fleurs V_F du puit. Il n'y a pas d'arêtes de $A - V_A$ vers $F - V_F$ sinon la coupe serait $+\infty$. Donc toutes les arêtes qui vont vers $F - V_F$ viennent de V_A, donc V_A les couvre. De même toutes les arêtes qui viennent de $A - V_A$ vont vers V_F donc V_F les couvre. Les arêtes restantes vont de V_A vers V_F et donc $V_A \cup V_F$ est une couverture des arêtes. Elle est bien minimale car si il existait par exemple $f \in V_F$ inutile, c'est que toute arête (a, f) est couverte par V_A et donc il n'y a pas d'arête de $A - V_A$ vers f. Donc si on change la coupe pour que f soit dans $F - V_F$, la nouvelle coupe ne coupe toujours pas d'arête $+\infty$ et donc son coût diminue de 1, ce qui est contradictoire avec la minimalité de la coupe initiale.*

Réciproquement une couverture (minimale) définit bien une coupe (minimale pour la même raison que ci-dessus) et donc un flot max et donc un couplage max.

Exercice 2 Prouver le théorème suivant :

Théorème 2 *(Hall) Montrer que toutes les gentilles petites abeilles peuvent trouver une fleur pour elle si et seulement si pour tout sous-ensemble A d'abeilles, l'ensemble $N(A) = \{f \in \text{jolies fleurs} \mid \exists a \in A \mid \heartsuit(a, f) = 1\}$ est au moins aussi grand que A.*

Solution: $\Rightarrow$ *facile.*

$\Leftarrow$ *À partir de König-Egervary : couplage saturant $A \Leftrightarrow$ couverture minimale de taille $|A|$. Sur une telle couverture, on peut remplacer tout $f \in F$ par un de ses voisin $a \in N(f) \subset A$. Donc on a $\Leftrightarrow$ A est une couverture de taille minimale.*

Soit une couverture minimale S. Si $\forall x \in S, x \in A$, c'est bon. Sinon soit $F' \subset F$ maximal tel que $F' \subset S$. Si tous les voisins de F', $N(F') \subset A$, sont dans S, alors F' est inutile ce qui est impossible. Soit $A' \subset N(F')$ maximal tel que $A' \cap S = \emptyset$. Alors $N(A') \subset S$ puisque qu'il faut couvrir les arêtes qui vont de A' vers $N(A')$. Donc $N(A') \subset F'$ puisque $N(A') \subset F$ et par définition de F'. Supposons qu'il existe $f \in F'$ tel que $f \notin N(A')$. Alors tous les voisins de f sont dans $N(F') - A' \subset S$ donc f est inutile. C'est impossible, donc $N(A') = F'$. Si on remplace F' par A' dans la couverture S, cela reste une couverture : en enlevant F' seules les arêtes entre A' et F' ne sont plus couvertes, et A' les couvre toutes. Or par hypothèse $|N(A')| \geq |A'|$, donc la couverture reste minimale. Donc on peut trouver une couverture qui ne contient que des abeilles. Comme aucune abeille n'est isolée, la couverture doit donc les contenir toutes.

Autre solution par les élèves : $\Leftarrow$ *Soit une coupe de taille minimale définie par V_A et V_F. Soit $A' = A - V_A$. Les voisins de A' sont dans V_F car la coupe est min. De plus, tout élément de V_F est voisin d'un élément de A' sinon on pourrait l'enlever et trouver une coupe plus petite. Donc $N(A') = V_F$. Le coût de la coupe est $|V_A| + |V_F| = |V_A| + |N(A')| \geq |V_A| + |A'| = n$. Comme il existe une coupe minimale de coût n ($V_A = A$ et $V_F = \emptyset$), le coût est exactement n et donc il existe un couplage de coût n, c'est-à-dire que toutes les abeilles ont trouvé une fleur.*

Exercice 3 Applications de Hall :

Question 3.1. *System of Distinct Representatives :*

Soit $(X_1, \ldots, X_n)$ une famille de sous-ensembles d'un ensemble A. Un SDR est une famille $(x_1, \ldots, x_n)$ d'éléments de A telle que pour tout i, $x_i \in X_i$ et pour tout $j \neq i$, $x_i \neq x_j$. Quelle est la condition d'existence d'un SDR ?

Solution: *C'est exactement le même problème. On cherche à saturer $(x_1, \ldots, x_n)$.*

Question 3.2. Prouver que pour $k > 0$, si chaque abeille aime exactement k fleurs, et que chaque fleur est aimée par exactement k abeilles, alors il est toujours possible que chaque abeille trouve une fleur et que chaque fleur trouve une abeille.

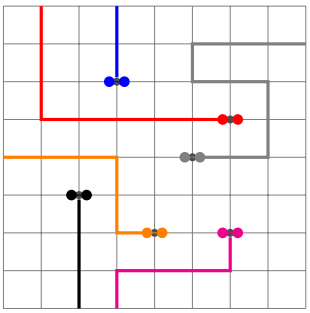
Solution: *Déjà, $|E| = \sum_{a \in A} |N(a)| = k|A|$ et de même $|E| = k|F|$ donc autant d'abeilles que de fleurs.*

Ensuite, soit $A' \subset A$.

$$k|A'| = \sum_{a \in A'} |N(a)| = \sum_{af \in E, a \in A'} 1 \leq \sum_{af \in E, f \in N(A')} 1 = \sum_{f \in N(A')} |N(f)| = k|N(A')|$$

Donc il existe un couplage qui sature A, donc il sature aussi F : c'est un couplage parfait.

Exercice 4 Tron v0.7



Des concurents sans pitié s'affrontent dans une arène représentée par une grille. Chaque *lightcycle* part d'un point situé au bord de la grille, et se déplace sur les arêtes de la grille en laissant derrière lui un mur. Le but est évidemment d'éviter de se prendre le mur d'un collègue (ou même le sien) dans la tronche. Les coins aussi sont dangereux évidemment. On suppose qu'on connaît uniquement la position actuelle des *lightcycles* et pas où sont les murs.
Comment déterminer si cette position est valide ou non (i.e. vérifier que les joueurs ont effectivement pu arriver à leurs positions respectives en respectant les règles) ?

Solution: *Problème de flot à maximiser encore. Placer des capacités 1 sur les nœuds et non sur les arêtes : dédoubler chaque point de la grille en un nœud d'entrée et de sortie, avec une capacité 1 entre e et s. Chaque arête x, y de la grille a un lien infini de $s(x)$ vers $e(y)$ et de $s(y)$ vers $e(x)$. Plus définition d'une source qui va à tous les composants, et d'un puit où vont tous les points du bord de la grille.*

Exercice 5 Cache d'instructions *direct-mapped*.

On a une boîte très très longue et c couleurs différentes. La boîte est coloriée avec les couleurs : chaque unité de longueur a une couleur donnée, et l'ordre des couleurs est toujours le même, si bien que l'on a un coloriage périodique de période c unités de longueur.

Dans notre boîte on veut mettre n gros serpents. Les gros serpents sont de taille entière en unités de longueur, et sont aussi coloriés, de la même manière que la boîte. On connaît la taille de tous les serpents, et aussi la couleur de leur queue (ce qui fait qu'on sait exactement comment ils sont coloriés).

Le but est de rentrer tous les serpents dans la boîte, sachant que dans une partie de la boîte on ne peut mettre qu'un serpent (ils sont gros), et en plus, chaque partie du corps d'un serpent doit être de la même couleur que la partie de la boîte qui l'entoure.

Exemple : c = {rouge,vert,bleu}, et trois serpents *vbr*, *b*, et *br*. Alors une solution possible est :

b	r	v	b	r		b

avec un trou entre le deuxième et le troisième serpent.

Question 5.1. Proposez une méthode qui marche et prouvez que la longueur de boîte utilisée n'est pas optimale.

Solution: *Glouton : on rajoute à chaque fois l'intervalle le moins loin qui n'est pas déjà pris.*

On suppose qu'on a le droit d'utiliser plusieurs boîtes. Les boîtes peuvent commencer à n'importe qu'elle couleur mais toutes doivent avoir une taille multiple de c. On veut minimiser la somme des longueurs des boîtes utilisées.

Question 5.2. Modélisez à l'aide de graphes bipartis et prouvez qu'une méthode gloutonne renvoie l'optimal.

Solution: *Biparti avec X les têtes de serpent et Y les queues. But = couplage parfait maximal. Glouton dépend de l'ordre dans lequel on considère les $x \in X$. Soit $x_1, \cdots, x_n$ l'ordre choisi par glouton. Soit une solution optimale S_{opt}. Si glouton donne S_{opt} tout va bien. Sinon, il existe i minimal tel que le y associé dans glouton soit $y_g \neq y_{opt}$ celui associé dans la solution optimale. Soit x_j à qui glouton a associé y_{opt}. Alors comme glouton a choisi y_g, c'est que y_g commence avant y_{opt} par rapport à la fin de x_i. En étudiant les trois cas possible pour le début de y_{opt} (par rapport à la fin de x_i : avant y_g, avant y_{opt}, ou avant $c-1$), on voit que l'on peut échanger dans la solution optimale y_{opt} et y_g en diminuant le coût. Donc on peut transformer la solution optimale jusqu'à optenir la solution gloutone, en diminuant le coût, donc la solution gloutonne est aussi optimale.*

Question 5.3. Modifiez la solution trouvée à la question précédente pour obtenir une solution de même coût qui est minimale en nombre de boîtes utilisées (on ne prouvera pas la minimalité).

Solution: *Fusion des boîtes=cycles où des trous intersectent. La fusion de deux cycles crée un cycle dont la taille est la somme des tailles des cycles initiaux. Quand plus de trous intersectent, on peut plus fusionner sans augmenter le coût de c. Noter que l'ordre de fusion des cycles deux à deux importe peu. Pour le prouver, il faudrait prouver d'abord*

qu'une solution optimale peut être toujours trouvée par un glouton ; puis, considérer deux boîtes dont les trous ne s'intersectent pas, alors un glouton ne mettrait jamais côte à côte un serpent de la première boîte avec un serpent de la deuxième.

Question 5.4. La solution obtenue à la question précédente a un coût ¢ minimal. S'il restait au moins deux boîtes, montrez comment la modifier pour obtenir une solution avec une seule boîte. Quelle est la longueur minimale de cette boîte ?

Solution: *Il n'y a plus de trous qui s'intersectent entre plusieurs boîtes. Considérer un serpent s dans une boîte. Alors dans une autre boîte, soit il y a un serpent plus grand (et qui contient donc les trous devant et derrière s), soit il y a deux serpents dont le trou entre les deux est au niveau de s (et donc s contient ce trou). Si on raisonne ainsi sur toutes les boîtes, il existe donc un serpent s qui contient au moins un trou de chaque autre boîte. Par exemple le serpent le plus long convient. Si on supprime s, on peut alors fusionner toutes les boîtes puisque le trou laissé par s intersecte tous les autres. Le coût reste donc ¢, et il ne reste plus qu'à rajouter s quelque part.*

On peut considérer que les serpents sont tous de taille inférieure ou égale à c (quitte à travailler sur d'autres serpents qui ont la même taille modulo c, puis rajouter à la fin des blocs de taille c au milieu des serpents qui en avaient besoin) donc rajouter s à la fin coûte c de plus.

On ne peut pas faire mieux puisque s'il restait au moins deux boîtes, il était impossible de faire le même coût avec une seule, et la longueur d'une boîte doit être multiple de c, donc le coût est bien minimal est égal à ¢+c.

On revient au cas initial : on n'a qu'une boîte et sa taille n'est pas forcément multiple de c.

Question 5.5. Utilisez l'algorithme de la question précédente pour obtenir une solution optimale.

Solution: *Soit une solution optimale. Si on la transforme en une solution de la question d'avant, on ralonge la boîte jusqu'à ce qu'elle soit de taille ¢ multiple de c. Cela crée un espace à la fin de la boîte. Soit alors un serpent virtuel s capable d'être mis exactement à cet endroit.*

Si on applique l'algorithme à l'ensemble des serpents plus s, on obtient une boîte de taille minimale, donc ¢. Si on enlève ce serpent virtuel s, qu'on coupe de la boîte cet endroit et qu'on colle le début de la boîte à la fin, on obtient bien une solution optimale.

Le problème est qu'on ne connaît pas s... Mais on s'en fiche, car des s possibles, il n'y en a que c^2 différent. Donc on teste avec tous les s possible et on garde la meilleure solution.

On peut faire aussi moins de tests en remarquant que plus s est gros mieux c'est (ça représente ce qu'on gagne en taille par rapport à ¢ à la fin). Donc on part d'un s petit (de taille nulle, mais entre deux couleurs fixes) et à chaque étape, soit on le fait grossir vers la droite, soit on le décale vers la droite. Ce qui fait qu'on n'a que $2c$ étapes à faire.

À chaque étape, si on a trouvé une solution avec s de taille x, on ne cherche plus que des cas qui ont besoin un s plus grand, donc on fait grossir s à $x+1$ au niveau de la tête. Sinon, ça ne sert à rien de chercher une solution avec s de taille $x+k$ car si une telle solution existe, on trouvera d'abord la solution avec s de taille x, au même endroit, et on fera alors grossir s de 1 en 1 jusqu'à arriver à la taille $x+k$; donc on continue de chercher une solution avec s de taille x. Cette solution a un s qui commence plus tard (car sinon on serait tombée dessus avant, quand $|s| \leq x$), donc on décale s de 1 vers la droite.