DS4-MPI

Ce sujet est composé de cinq exercices indépendants. Vous devez en choisir 4 à traiter. Nous vous demandons de bien préciser en début de copie, quels sont les exercices choisis.

1 Coupe maximale

Q6 On rappelle que pour une coupe $\delta(S)$, chaque arête de la coupe possède exactement deux extrémités appartenant à V.

Ainsi, en sommant $\deg_{\text{cut}}(x)$ sur tous les sommets, chaque arête de $\delta(S)$ est comptée exactement deux fois, une fois pour chacune de ses extrémités. On obtient donc :

$$\sum_{x \in V} \deg_{\text{cut}}(x) = 2|\delta(S)|.$$

Q7 Lorsqu'on déplace un sommet x d'un côté à l'autre de la partition, les arêtes incidentes à x changent de statut :

- les $\deg_{\text{cut}}(x)$ arêtes qui appartenaient à la coupe n'y appartiennent plus ;
- les $\deg_{\text{out}}(x)$ arêtes qui n'appartenaient pas à la coupe y appartiennent désormais.

La variation de la taille de la coupe est donc :

$$\Delta = \deg_{\text{out}}(x) - \deg_{\text{cut}}(x).$$

Le déplacement augmente strictement la coupe si et seulement si

$$\deg_{\text{out}}(x) > \deg_{\text{cut}}(x).$$

Q8 À chaque itération de l'algorithme, la taille de la coupe augmente strictement. Or $|\delta(S)|$ est un entier compris entre 0 et $|E|$.

Il ne peut donc y avoir qu'un nombre fini d'itérations : l'algorithme termine.

Q9 À la fin de l'algorithme, il n'existe plus de sommet x dont le déplacement augmente strictement la coupe. Par la question précédente, cela signifie que pour tout $x \in V$:

$$\deg_{\text{out}}(x) - \deg_{\text{cut}}(x) \leq 0,$$

c'est-à-dire :

$$\deg_{\text{cut}}(x) \geq \deg_{\text{out}}(x).$$

Q10 On a pour tout sommet x :

$$\deg(x) = \deg_{\text{cut}}(x) + \deg_{\text{out}}(x) \leq 2 \deg_{\text{cut}}(x).$$

En sommant sur tous les sommets :

$$\sum_{x \in V} \deg(x) \leq 2 \sum_{x \in V} \deg_{\text{cut}}(x).$$

Par le lemme de la poignée de main,

$$\sum_{x \in V} \deg(x) = 2|E|, \quad \sum_{x \in V} \deg_{\text{cut}}(x) = 2|\delta(S)|.$$

On obtient donc :

$$2|E| \leq 4|\delta(S)|, \quad \text{soit} \quad |\delta(S)| \geq \frac{|E|}{2}.$$

Comme la taille d'une coupe optimale est toujours inférieure ou égale à $|E|$, l'algorithme fournit une approximation à facteur $\frac{1}{2}$ pour Max-Cut.

Q11 Dans un triangle K_3, on a $|E| = 3$. Une coupe optimale contient 2 arêtes, ce qui est strictement supérieur à $|E|/2 = 1{,}5$.

Plus généralement, dans un graphe complet K_n, une coupe équilibrée contient environ $n^2/4$ arêtes, strictement plus que $|E|/2$ pour $n \geq 3$.

Corrigé — Exercice 2 (programmation OCaml)

Q12 (**Q12**) On calcule la taille de la coupe en comptant chaque arête *une seule fois*. Comme le graphe est non orienté et que la liste d'adjacence contient en général les deux sens, on ne compte l'arête $\{u, v\}$ que lorsque $u < v$.

```
(* g : int list array, sommets 1..n, g.(u-1) = voisins de u *)
let cut_size (g : int list array) (side : bool array) : int =
  let n = Array.length g in
  let acc = ref 0 in
  for u = 0 to n-1 do
    List.iter (fun v ->
      if u < v then
        if side.(u) <> side.(v) then incr acc
    ) g.(u)
  done;
  !acc
```

Q13 (**Q13**) Le degré dans la coupe $\deg_{\text{cut}}(x)$ est le nombre de voisins placés de l'autre côté.

```
let deg_cut (g : int list array) (side : bool array) (x : int) : int =
  let sx = side.(x) in
  let acc = ref 0 in
  List.iter (fun v ->
    if side.(v) <> sx then incr acc)  g.(x)
  !acc
```

Q14 (**Q14**) On a $\deg_{\text{out}}(x) = \deg(x) - \deg_{\text{cut}}(x)$ et $\deg(x) =$ longueur de $g.(x-1)$.

```
let deg_out (g : int list array) (side : bool array) (x : int) : int =
  (List.length g.(x)) - (deg_cut g side x)
```

Q15 (**Q15**) Variation de la coupe lorsqu'on déplace x :

$$\Delta = \deg_{\text{out}}(x) - \deg_{\text{cut}}(x).$$

```
let delta (g : int list array) (side : bool array) (x : int) : int =
  (deg_out g side x) - (deg_cut g side x)
```

Q16 (**Q16**) On cherche un sommet améliorant : un x tel que $\Delta > 0$.

```
let find_improving_vertex (g : int list array) (side : bool array) : int option =
  let n = Array.length g in
  let rec aux x =
    if x = n then None
    else if delta g side x > 0 then Some x
    else aux (x+1)
  in
  aux 0
```

Q17 (**Q17**) Algorithme d'amélioration locale : on initialise tout dans le même côté, puis on applique des déplacements tant qu'il existe une amélioration.

```
let maxcut_local (g : int list array) : bool array =
  let n = Array.length g in
  let side = Array.make n true in
  let rec loop () =
    match find_improving_vertex g side with
    | None -> side
    | Some x ->
        side.(x) <- not side.(x);
        loop ()
  in
  loop ()
```

(Remarque : cet algorithme termine car la taille de la coupe augmente strictement à chaque déplacement et est majorée par $|E|$.)

2 Grammaires

Q18 (**Q18**) On cherche une dérivation à gauche du mot $u = abc$.
On part du symbole initial S :

$$S \Rightarrow SaS \Rightarrow AaS \Rightarrow BaS \Rightarrow aS \Rightarrow aA \Rightarrow aAbA \Rightarrow aBbA \Rightarrow abA \Rightarrow abB \Rightarrow abBcB \Rightarrow abcB \Rightarrow abc$$

Le mot abc admet au moins une dérivation dans la grammaire G, ce qui montre que $u \in L(G)$.

Q19 (**Q19**) On peut construire deux dérivations distinctes pour le mot $u' = aa$.

Premier arbre :

$$S \Rightarrow SaS \Rightarrow AaS \Rightarrow BaS \Rightarrow aS \Rightarrow aSaS \Rightarrow aAaS \Rightarrow aBaS \Rightarrow aaS \Rightarrow aaA \Rightarrow aaB \Rightarrow aa$$

Second arbre :

$$S \Rightarrow SaS \Rightarrow SaSaS \Rightarrow AaSaS \Rightarrow BaSaS \Rightarrow aSaS \Rightarrow aAaS \Rightarrow aBaS \Rightarrow aaS \Rightarrow aaA \Rightarrow aaB \Rightarrow aa$$

Ces deux arbres sont distincts et conduisent au même mot terminal aa. La grammaire G est donc *ambiguë*.

Q20 (**Q20**) Une variable est récursive gauche directe si elle possède une règle de la forme $A \to A\alpha$.
Dans la grammaire G :

- $S \to SaS$ montre que S est récursive gauche ;
- $A \to AbA$ montre que A est récursive gauche ;
- $B \to BcB$ montre que B est récursive gauche.

Les trois variables S, A et B sont donc récursives gauches directes.

Q21 (**Q21**) On élimine la récursivité gauche variable par variable en appliquant l'algorithme donné.

Élimination pour B :

$$B \to BcB \mid \varepsilon$$

devient :

$$\begin{cases} B \to B' \\ B' \to cBB' \mid \varepsilon \end{cases}$$

Élimination pour A :

$$A \to AbA \mid B$$

devient :

$$\begin{cases} A \to BA' \\ A' \to bAA' \mid \varepsilon \end{cases}$$

Élimination pour S :

$$S \to SaS \mid A$$

devient :

$$\begin{cases} S \to AS' \\ S' \to aSS' \mid \varepsilon \end{cases}$$

On obtient ainsi une grammaire G' équivalente à G et ne contenant plus de récursivité gauche directe.

Q22 (**Q22**) On montre que le langage engendré par G (et donc G') est $\{a, b, c\}^\star$.

Inclusion $\subseteq$: Toutes les règles de production ne produisent que des lettres de $\{a, b, c\}$, donc $L(G) \subseteq \{a, b, c\}^\star$.

Inclusion $\supseteq$:
Par récurrence forte sur la longueur du mot, on montre que tout mot de $\{a, b, c\}^\star$ peut être engendré par G, que tout mot de $\{b, c\}^*$ peut être engendré à partir du symbole A et que tout mot de $\{c\}^*$ peut être engendré à partir du symbole B.
En effet le mot vide est bien engendré par les dérivations suivantes :

$$S \Rightarrow A \Rightarrow B \Rightarrow \epsilon$$

$$A \Rightarrow B \Rightarrow \epsilon$$

$$B \Rightarrow \epsilon$$

- Soit u un mot de longueur $n + 1$ avec $n \geq 0$ quelconque dans $\{a, b, c\}^*$.

  Supposons dans un premier temps que u contient au moins un a et donc u s'écrit sous la forme $u = u_1 a u_2$ avec u_1 et u_2 deux mots de $\{a, b, c\}^*$ de longueur au plus n qui sont générés par la grammaire par hypothèse de récurrence. Ainsi :

  $$S \Rightarrow SaS \Rightarrow^* u_1 a u_2 = u$$

  Supposons maintenant que le mot ne contient aucun a mais contient au moins un b. Il est donc de la forme $u = u_1 b u_2$ avec u_1 et u_2 deux mots de $\{b, c\}^*$ de longueur au plus n qui sont générés à partir de A d'après l'hypothèse de récurrence et donc la dérivation $S \Rightarrow A \Rightarrow AbA \Rightarrow^* u_1 b u_2$ convient.

  Enfin si le mot appartient à $\{c\}^*$ alors il est de la forme $u = cu'$ avec u' de longueur n qui est engendré à partir de B par HR et donc $S \Rightarrow A \Rightarrow B \to BcB \Rightarrow cB \Rightarrow cu'$ convient.

- Si le mot est dans $\{b, c\}^*$ on a vu dans la résolution précédente (deuxième cas) qu'il était engendré à partir de A car notre dérivation commencait par $S \Rightarrow A$ d'où l'on dérivait u.

- Si le mot n'est composé que de c on a vu qu'on pouvait le dériver depuis B en prenant la dérivation à partir de la troisième étape.

Ainsi :

$$L(G) = L(G') = \{a, b, c\}^\star.$$

3 Recherche d'une clique de célébrités

3.1 Définitions et propriétés