Numéro de place 10356 0

Numéro d'inscription 14665

Nom SHOFFRAY

Prénom NILS

Signature

CONCOURS CENTRALE•SUPÉLEC

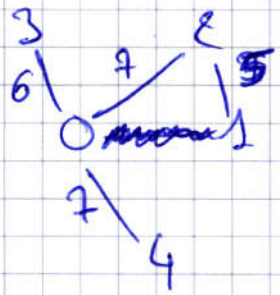
Épreuve Option informatique

Ne rien porter sur cette feuille avant d'avoir complètement rempli l'entête

Feuille 01 / 03

I Généralités

1. Un arbre couvrant de poids minimal de G_2 est :



2. Soit $G = (S, A)$ un graphe :

- Supposons G sans cycles et $|A| > |S| - 1$ on appelle B les $|S|-1$ premières arrêtes de A tel que $T = (S, B)$ n'ait pas de cycle. Soit i_0 et j_0 tels que $(i_0, j_0) \notin B$ et i_0, j_0 dans la même composante connexe de T. Alors il existe une chaîne $(s_0 \dots s_k)$ $s_0 = i_0$ et $s_k = j_0$ mais alors $(s_0 \dots s_k, i_0)$ forme un ou plusieurs cycle de $T' = (S, B \cup [(i_0, j_0)])$ et T' est un sous graphe de G qui possède donc des cycles, c'est absurde

$$|A| \leq |S| - 1.$$

- Supposons G connexe et sans perte de généralité on prend $|A| = |S| - 2$ il existe donc un ou plusieurs sommets de G qui n'appartiennent pas à la même composante connexe de les autres ce qui est absurde. Donc $|A| \geq |S| - 1$.

Ne rien écrire dans la partie barrée

3. si G est un arbre ssi G est connexe donc $|A| \geq |S|-1$ et G sans cycle donc $|A| \leq |S|-1$ ssi G connexe et $|A| = |S|-1$ $\Rightarrow$ G est sans cycle et $|A| = |S|-1$

4. Supposons trouvés G et G' deux arbres couvrant de poids minimaux donc :

$$f(G) = f(G') \iff \sum_{a \in A} f(a) = \sum_{a \in A'} f(a) \iff \sum_{a \in A \setminus A'} f(a) = \sum_{a \in A' \setminus A} f(a)$$

or f injective et $\forall a \in A \cup A'$ $f(a) > 0$ donc $\sum_{a \in A \setminus A'} f(a) \neq \sum_{a \in A' \setminus A} f(a)$ absurde. On a donc que $G = G'$ et l'unicité de l'arbre couvrant de poids minimal.

5. Soit $(s_0, \dots, s_k) \in S^{k+1}$ des sommets tels que $s_0 \dots s_k$ forment un cycle dont a est une arête de poids maximal dans le cycle ainsi $\forall i,j \in [\![0,k]\!]^2$ $f(s_i s_j) \leq f(a)$. Si a dans l'arbre couvrant minimal alors il suffit de prendre l'arbre couvrant ne contenant pas a mais toutes les arêtes du cycle nécessaires. Dans ce cas le poids du nouvel arbre est inférieur à celui qui couvrait a et supposé minimal ce qui est absurde. $a \notin B^*$

6. Cette séparation est :

```
match lst with
  [] -> [], []
| [a] -> [a], []
| a::b::c -> let l1, l2 = separate c in
             a::l1, b::l2 ;;
```

7.
```
let rec fusion l1 l2 =
    match l1, l2 with
    [], [] -> []
    | l, [] -> l1
    | [], l2 -> l2
    | a::b, c::d -> if a <= c then a::fusion (b l2)
                    else c::fusion (l1 d) ;;
```

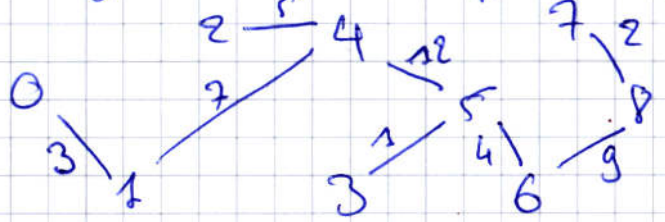
8.
```
let tri_fusion lst =
    fusion (separation lst) ;;
```

La complexité temporelle de ce tri est en $O(n \log(n))$ avec n la taille de la liste

II Algorithme de Kruskal inversé

9. L'algorithme de Kruskal a une complexité temporelle en $O(|S||A|)$

10. L'algorithme de Kruskal inversé appliqué à G_3 donne :



11. On modifie légèrement fusion :
```
let rec fusion l1 l2 = match l1 l2 with
    [], [] -> []
    | l, [] -> l1
    | [], l2 -> l2
    | a::b, c::d -> if a < c then a::fusion (b l2)
                    else if a > c then c::fusion (l1 d)
                    else a::fusion (b d) ;;
```
(* si a = c on laisse de (sur 2 *)

```
let tri_aretes g = let n = Array.length g in let l = ref [] in
  for i = 0 to n-1 do
    add_arc g.(i) with
  let rec arc l i =
    match l with
      [] -> []
    | a::b -> let j,p = a in
              if i<j then (p,i,j)::arc(b,i)
              else arc(b,i) in
  for i = 0 to n-1 do
    l := !l @ (arc g.(i) i)
  done
  l := tri_fusion !l ;
  List.rev !l ;;
```

12.
```
let rec connectes g s t paids_mes =
  match g.(s) with
    [] -> false
  | a::b -> let j,p = a in
            if j=t && p=paids_mes then false
            else if j=t && p<paids_mes then true
            else if j<>t && p<paids_mes then connectes g j t paids_mes
```

```
let connectes g s t paids_mes =
  let rec arc l = match l with
    [] -> false
  | a::b -> let j,p = a in
            if j=t && p<paids_mes then true
            else arc b
  in arc g.(s)
```

(1) (2) (3) (4)

Numéro de place 10356O

Numéro d'inscription 14665

Nom XHOFFRAY

Prénom NIS

Signature

CONCOURS CENTRALE-SUPÉLEC

Épreuve option informatique

Ne rien porter sur cette feuille avant d'avoir complètement rempli l'entête

Feuille 02/03

II Algorithme de Kruskal inversé

12 (1)(2)(3)(4) ~~elif ... then~~

elif j <> i && p <. poid_max then aux g.(j)

else aux b in ;

aux g.(s) ;;

13. let kruskal_inverse g = let b = Array.make (Array.length g) [] in

let a = tri_aretes g in ; let ~~bool = ref true~~ ; let n = Array.length g in ;

let rec aux _ l = let bool = ref true in

match l with

[] -> b

| (p,s,i)::q -> let p, s, i = a in

for i = 0 to ~~Array.length g~~ n-1 do

if not (mene g s i p) then bool := false

done ;

if !bool = true then ~~(i,p)::b.(s), s~~ begin

b.(s) <- (i,p) :: b.(s) ;

b.(i) <- (s,p) :: b.(i) end

else bool := true ;

aux q in ;

aux a ;;

Ne rien écrire dans la partie barrée

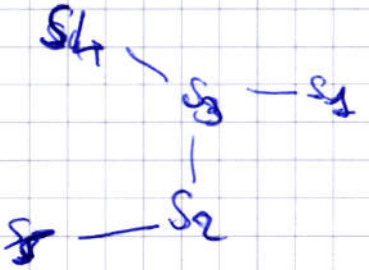
14. si G est connexe puisque l'algorithme de Kruskal inversé vérifie si le sous graphe donné en entrée est connexe avant de rendre cette donnée on obtient bien un arbre couvrant de G. un sous graphe de G connexe à |S|-1 arêtes et d'après G partie 3, G sous arbre donné est couvrant.

15. Kruskal inversé renvoie l'arbre couvrant de G possédant les arêtes de plus petits poids possible donc le graphe donné en sortie de kruskal inversé est bien un arbre couvrant minimum.

16 l'algorithme de kruskal a une complexité temporelle en $O(|S||A|(|S|+|A|))$

III. Difficulté du calcul de l'arbre optimal

17. si $X = \{s_1, s_4, s_5\}$ est un graphe de steiner admet moins de 2 sommets de steiner est :

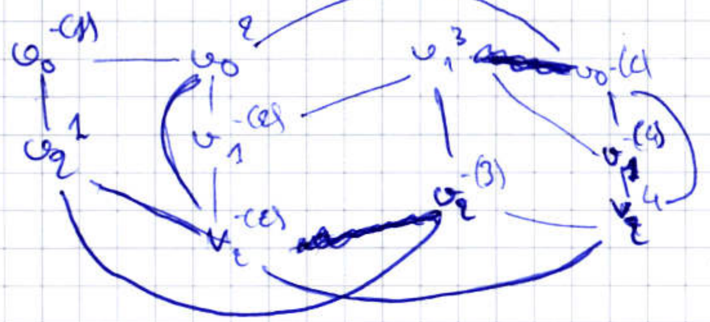


puisque qu'un graphe de steiner est un arbre il est connexe donc ici par exemple graphe couvrant $\{s_1, s_4, s_5\}$ est connexe il est nécessaire qu'il atteigne s_3 ou s_2 il aura donc au moins 1 sommet de steiner.

A De la satisfaisabilité au Stabb

18. un stable de cardinal maximal de G_1 est : $S = \{s_0, s_1, s_5\}$

en effet 3 ne peut pas appartenir à un stable de G_1 et les 5 sommets de cardinal maximum restants possèdent tous au moins 2 voisins sauf s_5 donc si on en choisit un il ne reste plus que deux choix et dès qu'on effectue ce choix puisque notre choix est vrai on ne peut plus agrandir le stable, le cardinal maximum est 3

19. 

20. Si S est un stable de taille n dans G_φ, alors puisque dans G_φ il y a des arêtes entre tous les sommets d'une même clause et entre les sommets des variables et leurs négation si on prend un sommet de S dans la valeur associée à ce sommet vraie la clause dans laquelle il se situe et on ne peut choisir un sommet de S qui contredira celui qu'on étudie puisqu'il est voisin avec toutes ses négations.

Ainsi en prenant les valeurs des variables associées aux sommets de S on obtient un modèle de φ, donc φ est satisfaisable.

21. De même si φ satisfaisable alors on a $(v_1 \ldots v_n)$ des variables qui satisfont φ peut importe comment on les complètes et alors il suffit de choisir un sommet de G_φ de même valeur que v_1 puis de continuer par tous les v_n en vérifiant que le sommet choisi n'est pas adjacent au précédent, cela existe par construction de G_φ. En réunissant ses sommets on obtient un stable de G_φ de cardinal n.

B. Du stable à l'arbre de Steiner

Q3. Si on possède un stable X de G de cardinal k dès il existe un arbre de Steiner de G' contenant les $\{s_a, a \in A\}$ et k sommets n'appartenant pas à X donc possédant $|S| - k$ sommets de Steiner et X comme sommets terminaux. Puisque les sommets de $S \setminus X$ ne sont pas dans le stable dès il existe une arête pour chacun et chacun à une autre sommet dans il sont voisins d'un sommets de X.

Q4. Si on sait trouver un arbre de Steiner de G'_ρ dès on sait trouver un stable de G_ρ et donc un modèle de ρ s'il existe. Donc si on sait trouver un arbre de Steiner, à temps polynomial, on sait trouver un stable de G_ρ en temps polynomial de G'_ρ et donc on sait vérifier si ρ satisfaisable en temps polynomiaux.

IV. Approximation du problème

A. Algorithme de Floyd-Warshall

Q5.
```
let matrice_adjacence g = let n = Array.length g in
    let m = Array.make_matrix n n infinity in
    let aux l s =
      match l with
      [] -> ()
      | a::q -> let t,p = a in
                m.(s).(t) <- p;
                aux q s in
    for s = 0 to n-1 do
      aux g.(s) s done;
    m;;
```

IV Approximation du problème

A. Algorithme de Floyd-Warshall

Q6. $\Pi^{(0)}$ est la matrice telle que $\forall (s,t) \in S^2$ $\Pi^{(0)}_{st}$ est le poids minimal d'une chaîne allant de s à t et dont les sommets intermédiaires sont strictement inférieurs à 0 donc $\forall i \in [\![0, n-1]\!]$ $M^{(0)}_{ii} = 0$ et $\forall (s,t) \in S^2$ $s \neq t$ $\Pi^{(0)}_{st} = d^{(0)}_{st} = +\infty$ si s et t ne sont pas voisins avec ℓ

$$\Pi^{(0)} = \begin{pmatrix} 0 & d_{01} & \cdots & d_{0,n-1} \\ d_{10} & 0 & \cdots & +\infty \\ \vdots & \vdots & \ddots & \vdots \\ d_{n-1,0} & +\infty & \cdots & 0 \end{pmatrix}$$

Q7. Pour $k \in [\![0, n-1]\!]$ et $(s,t) \in S^2$ dire qu'il existe une chaîne allant de s à t de poids minimal inférieur à $\Pi^{(k)}_{st}$ et contenant k dans ce cas $\Pi^{(k+1)}_{st} = \Pi^{(k)}_{sk} + \Pi^{(k)}_{kt}$

Sinon il n'existe pas de tel chemin et donc $\Pi^{(k+1)}_{st} = \Pi^{(k)}_{st}$

Donc dans tous les cas $\boxed{\Pi^{(k+1)}_{st} = \min\left(\Pi^{(k)}_{st}, \Pi^{(k)}_{sk} + \Pi^{(k)}_{kt}\right)}$

Ne rien écrire dans la partie barrée

28. let floyd_warshall g = let n = Array.length g in
let dist = Array.make_matrix n n infinity in
let pred = Array.make_matrix n n -1 in
for i = 0 to n-1 do
dist.(i).(i) ← 0
pred.(i).(i) ← i
dist.(i).(j) ← g.(i).(j)
pred.(i).(j) ← i
dist.(j).(i) ← g.(j).(i)
pred.(j).(i) ← j
done
for k = 0 to n-1 do
for s = 0 to n-1 do
for t = 0 to n-1 do
dist.(s).(t) ← min (dist.(s).(t)) (dist.(s).(k) + dist.(k).(t))
if min (dist.(s).(t)) (dist.(s).(k) + dist.(k).(t)) < dist.(s).(t)
then pred.(s).(t) ← pred.(k).(t)
done
done
done
dist, pred ;;

29. On parcourt le retour n par chaque valeur de k dans [0, n-1] donc la complexité de l'algorithme de floyd_warshall est en $O(n^3)$

30 let rec chaine_min pred s t =
match ~~pred~~ (s).(t) with
| s → [s]
| -1 → []
| j → ~~pred.(s).(j)~~ j :: (chaine_min pred s ~~pred.(s).(j)~~ j) ;;

D. Solution approchée

31 Par construction on a X ⊂ S puis Y ⊂ S jusqu'à l'étape 3 on rajoute des sommets de G.
Si st ∉ B_1 \ A dans g(st) = f(st) et sinon on remplace l'arête par une chaîne de poids minimal, donc avec des arêtes et des sommets de G.
Au final on trouve bien que H_2 est un sous-graphe de G avec Y ⊂ S donc c'est un arbre de Steiner. Ainsi T l'est également et les sommets terminaux de T sont ~~ceux~~ de X

32 let renumeroter tab_X = let n = Array.length tab_X in
let num = Array.make n ~~-1~~ in
for i = 0 to n-1 do
~~num.(i)~~ if tab_X(i) then num.(i) ← num(i) done;
num ;;