Algorithmique
Partiel n° 2

Info-Spé – Epita

D.S. 312354.45 BW (10 mai 2011 - 9 :00)

Consignes (à lire) :

- Vous devez répondre sur **les feuilles de réponses prévues à cet effet**.
  - Aucune autre feuille ne sera ramassée (gardez vos brouillons pour vous).
  - Répondez dans les espaces prévus, **les réponses en dehors ne seront pas corrigées** : utilisez des brouillons !
  - Ne séparez pas les feuilles à moins de pouvoir les ré-agrafer pour les rendre.
  - Aucune réponse au crayon de papier ne sera corrigée.
- La présentation est notée en moins, c'est à dire que vous êtes noté sur 20 (le barème est sur 30 mais la note sera ramenée à 20) et que les points de présentation (2 au maximum) sont retirés de cette note.
- **Les algorithmes :**
  - Tout algorithme doit être écrit dans le langage Algo (pas de C, Caml ou autre).
  - Tout code Algo non indenté ne sera pas corrigé.
  - Tout ce dont vous avez besoin (types, routines) est indiqué en **annexe** (dernière page) !
- Durée : 3h00

Exercice 1 (Gisement épuisant... – *5 points*)

Des mineurs veulent sécuriser la circulation entre les différents points d'extraction (représentés par des sommets) reliés par des galeries (figure 1). Tous ces points d'extraction sont accessibles séparément depuis l'extérieur et le réseau est suffisamment complexe pour qu'en général plusieurs itinéraires permettent de passer d'un point d'extraction à un autre. Il n'est donc pas nécessaire que chaque galerie soit sécurisée pour qu'il existe toujours entre deux points d'extraction au moins un itinéraire qui le soit.

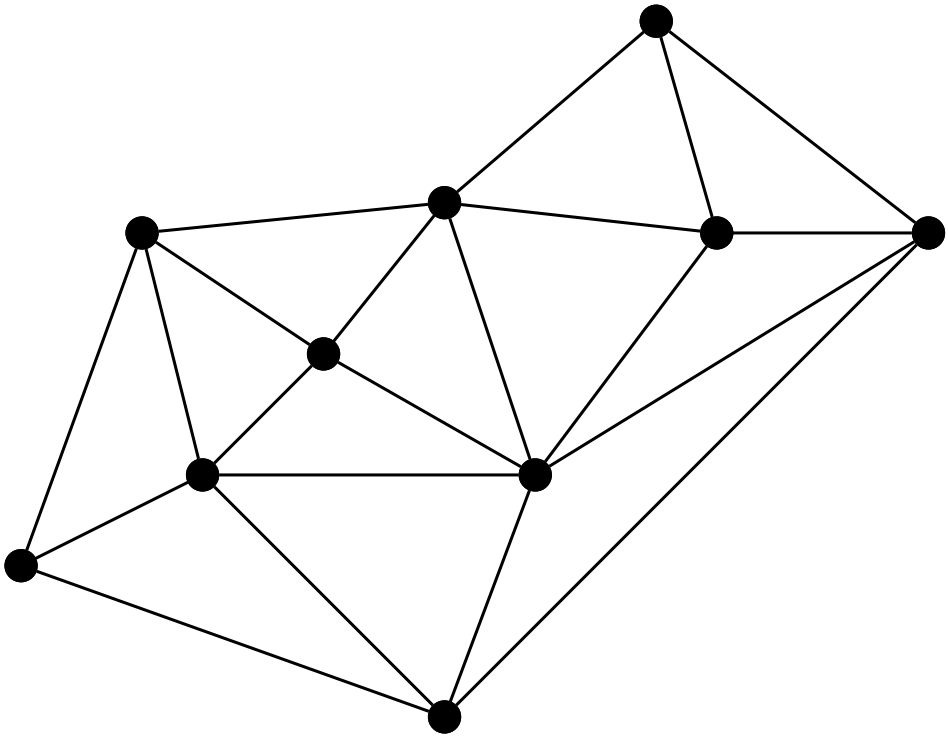

FIGURE 1 – Graphe associé à une mine et son réseau.

1. On veut déterminer le plus petit nombre de galeries à sécuriser : à quoi correspond la solution en termes de graphes ?
2. Dans le cas de la figure 1, combien faut-il sécuriser de galeries ?
3. Proposer une solution graphique (Surligner les galeries que vous vous proposez de sécuriser).
4. En généralisant à un réseau de N points d'extraction, combien faudrait-il sécuriser de galeries ?
5. Justifier cette réponse.

On affine l'analyse du problème : Pour chaque galerie, nous avons évalué le coût des travaux de sécurisation (voir figure 2).

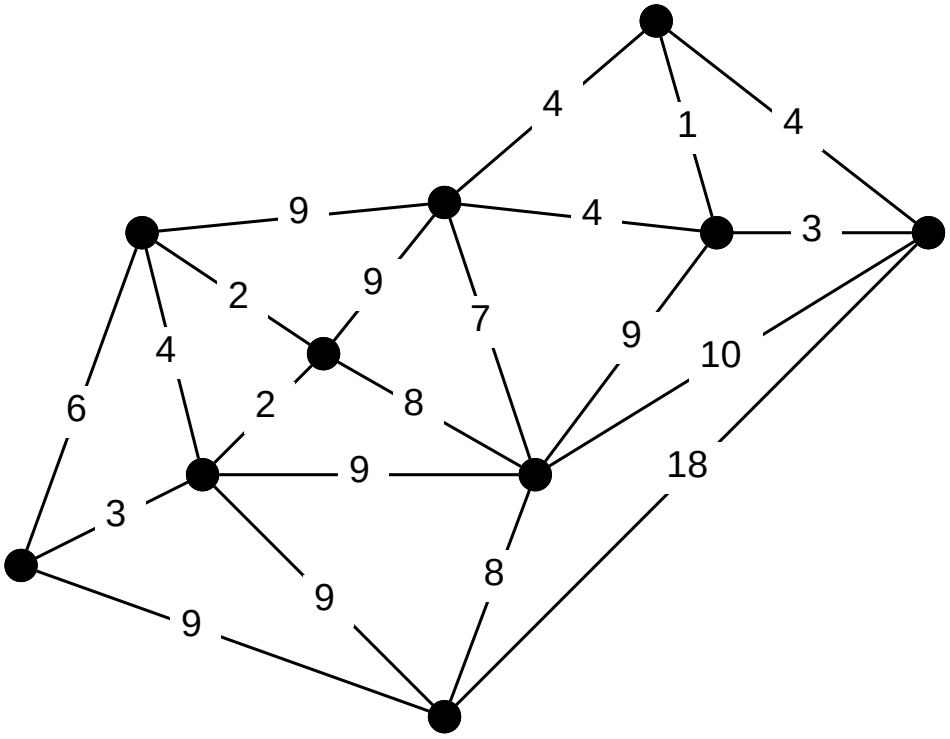

FIGURE 2 – Graphe valué associé à la sécurisation des galeries d'un réseau de grottes.

6. Comment dans ce cas sécuriser l'accès à toutes les grottes au moindre coût ?
7. Proposer une solution graphique (Tracer en gras les galeries qu'il faut sécuriser).
8. Cette solution est-elle unique ?
9. Pourquoi ?

Exercice 2 (Mangez des crêpes – *16 points*)

Après avoir creusé, une bonne crêpe pour se remettre ?

Le but ici est de savoir dans combien de temps nous pourrons manger une crêpe à la banane flambée.

Ci-dessous la recette, avec pour chaque tâche sa durée en secondes.

La recette	Durée (en sec.)	Réf.
Mettre la farine dans un saladier,	3	A
ajouter deux œufs,	30	B
ajouter le lait doucement et mélanger.	600	C
Dans une poêle mettre du rhum.	3	D
Couper les bananes en fines lamelles,	300	E
les mélanger au rhum.	30	F
Faire chauffer le mélange,	120	G
faire flamber le mélange.	10	H
Faire cuire une crêpe,	10	I
verser du mélange bananes-rhum sur la crêpe.	10	J

Quelques précisions concernant l'ordre des tâches :

- la préparation de la pâte à crêpe et celle du mélange rhum-banane peuvent se faire en parallèle ;
- ce n'est que lorsque la crêpe sera cuite et que le mélange sera prêt qu'on pourra verser du mélange sur la crêpe et enfin la déguster ;
- les autres étapes se réalisent séquentiellement (on doit mettre la farine avant les œufs, on doit mettre le rhum avant les bananes).

1. Modéliser la recette sous forme de graphe :
   – Les sommets sont les tâches.
   – Les tâches *debut* et fin représentent le début et la fin du projet.
   – La représentation du graphe doit être planaire[1] !
   La graphe qui représente la recette est sans circuit. De plus, tous les sommets sont atteignables depuis un sommet donné (ici la commande de la crêpe = la tâche *debut*, sommet n° 1 dans la représentation machine). Le graphe sera ici en représentation dynamique. Dans toute la suite de l'exercice, le graphe aura ces spécifications !

2. **Le cuisinier est tout seul en cuisine :** la durée minimum est donc la somme des temps nécessaires à chaque tâche. Par contre, il faut l'aider à trouver l'ordre dans lequel il va pouvoir faire les différentes tâches : la solution est un tri topologique du graphe.

   (a) Une solution de tri topologique évidente est donnée par l'ordre de la recette. Donner une autre solution : compléter celle donnée sur les feuilles de réponses.

   (b) Comment utiliser un parcours en profondeur pour trouver une solution de tri topologique dans un graphe de projet ?

   (c) Écrire l'algorithme correspondant : la solution de tri topologique doit être stockée dans une pile de pointeurs (le graphe est représenté en dynamique).

3. **Le cuisinier a trouvé des aides, les tâches peuvent donc être réalisées en parallèle :**

   (a) La date au plus tôt de la tâche i est la date à laquelle la tâche peut commencer au plus tôt. Comment calculer les dates au plus tôt de chaque tâche à partir de ce type de graphe ?
   Remplir le tableau donnant les dates au plus tôt pour la recette de la crêpe.

   (b) Quel temps faudra-t'il au minimum pour pouvoir déguster notre crêpe ?
   Comment obtenir cette date (la durée minimale du projet) ?

   (c) La date au plus tard de la tâche i est la date au plus tard où la tâche peut commencer sans entraîner de retard sur le projet. Comment calculer les dates au plus tard de chaque tâche ?
   Remplir le tableau donnant les dates au plus tard pour la recette de la crêpe.

   (d) Comment obtient-on les tâches critiques dans ce type de projet ?
   Donner les tâches critiques pour le présent projet.

   (e) Écrire l'algorithme qui détermine le temps minimum de réalisation d'un projet représenté par un graphe ayant les mêmes spécifications que ci-dessus.
   Vous devez obligatoirement utiliser l'algorithme de la question 2.

1. Pour mémoire, cela signifie que les arcs ne doivent pas se couper !

Exercice 3 (Construire un ARPM par suppression – *9 points*)

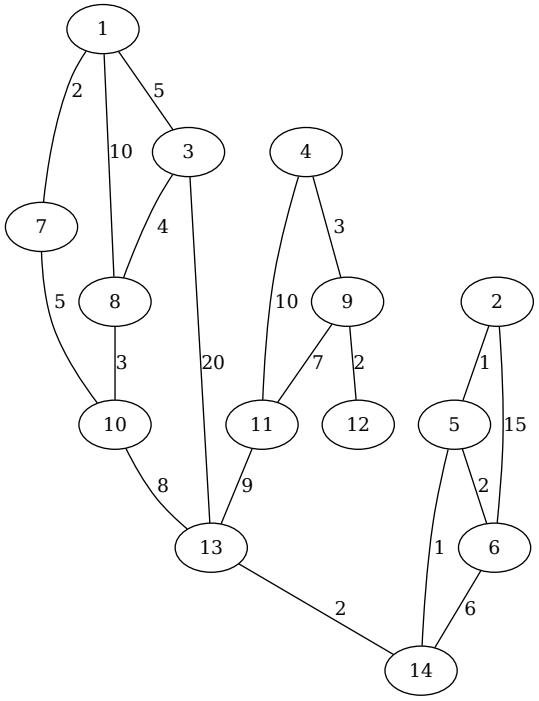

FIGURE 3 – Graphe valué non orienté

Le but ici est de construire un ARPM (Arbre de Recouvrement de Poids Minimum) sur un principe *inverse* de celui de l'algorithme de Kruskal : au lieu d'ajouter des arêtes dans l'arbre en construction en partant d'un arbre vide, on va supprimer les arêtes inutiles en partant du graphe d'origine. Au final, cet algorithme construit un arbre (graphe non-orienté sans cycle et connexe) dont la somme des poids est minimum.

Pour cet algorithme il nous faut la liste des arêtes triées mais dans l'ordre décroissant (du plus fort au plus faible coût). Ensuite, pour que le résultat reste un arbre, il faut s'assurer que les arêtes supprimées ne sont pas nécessaires à la connexité.

Nous exprimerons le résultat de l'algorithme avec la *liste* des arêtes supprimées représentée (la liste) par une matrice T telle que si $T[i,j] \neq 0$ alors l'arête (i,j) a été supprimée (bien évidemment, comme le graphe est non-orienté, on vérifie que $\forall\ i,j\ T[i,j] = T[j,i]$).

On supposera que le graphe en entré est connexe. On supposera également que la liste des arêtes triées est déjà construite. Elle est fournie à notre algorithme sous la forme d'un ensemble correspondant à la spécification suivante :

types
```
  arete = ^s_arete
  s_arete = enregistrement
    t_listsom          src, dst
    reel               cout
  fin enregistrement s_arete
  ensemble = /* Type "opaque" */
```

Opérations

/* l'ensemble est-il vide */
est_vide(ensemble e) : booleen
/* nombre d'éléments de l'ensemble */
card(ensemble e) : entier
/* Renvoie et supprime l'élément maximum de l'ensemble e */
supprime_max(ensemble e) : arete

1. Donner le principe d'un algorithme qui, à l'aide d'un parcours profondeur, détermine si deux sommets sont connectés dans un graphe.

2. Écrire la fonction `lie_rec(dst,ps,T,M)` qui teste (elle renvoie donc un booléen) s'il existe un chemin depuis le sommet pointé par `ps` jusqu'au sommet de numéro `dst`. La fonction utilise le principe de la question précédente. La matrice `T` décrit les arêtes qui ont été supprimées du graphe et le vecteur de booléens `M` sert de vecteur de marques pour le parcours profondeur. Cette fonction sera appelée par la fonction suivante :

```
algorithme fonction lie : booleen
  parametres locaux
    t_graphe_d        g
    entier            dst
    t_listsom         ps
    t_mat_entiers     T
  variables
    t_vect_booleens   M
    entier    i
debut
  pour i ← 1 jusqu'a g.ordre faire
    M[i] ← faux
  fin pour
  retourne (lie_rec(dst, ps, T, M))
fin algorithme fonction lie
```

3. Lorsque l'on supprime les arêtes du graphe, comment sait on que l'on peut arrêter l'algorithme ?

4. Écrire la procédure `revdel(g,E,T)` qui applique l'algorithme de construction de l'ARPM par suppression des arêtes dans le graphe `g`, à partir de l'ensemble d'arête `E` et indique dans la matrice `T` les arêtes supprimées.

5. Donner la liste des arêtes supprimées par cet algorithme appliqué au graphe de la figure 3.

Annexes

Représentation dynamique des graphes :

```
types
    t_listsom = ↑ s_som
    t_listadj = ↑ s_ladj
    s_som     = enregistrement
       entier      som
       t_listadj   succ
       t_listadj   pred
       t_listsom   suiv
    fin enregistrement s_som
    s_ladj      = enregistrement
       t_listsom   vsom
       reel        cout
       t_listadj   suiv
    fin enregistrement s_ladj
    t_graphe_d  = enregistrement
       entier     ordre
       booleen    orient
       t_listsom  lsom
    fin enregistrement t_graphe_d
```

Autres types utiles :

```
constantes
    Max = /* une valeur suffisante !*/
types
    t_vect_entiers  = Max entier
    t_vect_reels    = Max reels
    t_vect_booleens = Max booleen
    t_mat_entiers = Max × Max entier
    t_mat_reels   = Max × Max reel
```

Routines autorisées :

Piles : le type `t_pile`

Toutes les opérations sur les piles peuvent être utilisées à condition de préciser le type des éléments.

- `pile_vide () : t_pile`
- `est_vide (t_pile` p`) : booleen`
- `empiler( t_elt_pile` elt`, t_pile` p`) : t_pile`
- `depiler (t_pile` p`) : t_pile`
- `sommet (t_pile` p`) : t_elt_pile`

Autres

Les fonctions *max, min, abs*, ainsi que les valeurs ∞ et $-\infty$ sont aussi autorisées. De même pour la fonction *recherche (entier s, t_graphe_d G)* qui retourne le pointeur sur le sommet s dans le graphe G (de type `t_listsom`).