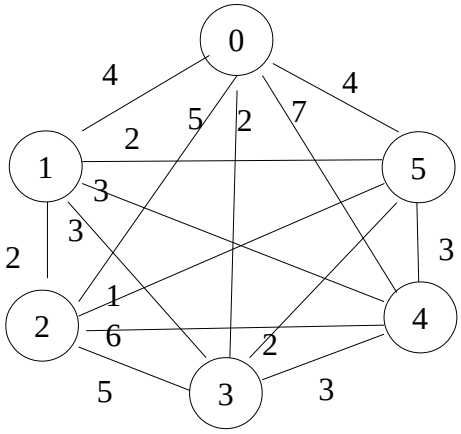
Université de Reims Champagne-Ardenne **Pierre Delisle**
Département de Math., Méca. et Informatique
CHPS0742 - Optimisation
2018/2019

Travaux Pratiques 4
Résolution optimale du problème du Voyageur de Commerce

L'objectif de ce TP est de programmer une méthode par séparation et évaluation pour résoudre de façon optimale le problème du voyageur de commerce. Pour ce faire, nous utiliserons le graphe suivant comme exemple (vu en cours) :



Vous pouvez télécharger le fichier de données correspondant à ce graphe (**graphe2.txt**) sur le site Web du professeur (http://cosy.univ-reims.fr/~pdelisle/enseignement.php).

Lecture du fichier de données

Vous noterez que la structure du fichier correspond à celle du TP 2. Par conséquent, votre fonction de lecture de graphe devrait faire l'affaire. Vérifiez quand même le résultat :

```
0   4   5   2   7   4
4   0   2   3   3   2
5   2   0   5   6   1
2   3   5   0   3   2
7   3   6   3   0   3
4   2   1   2   3   0
```

1) Si vous n'obtenez pas la matrice d'adjacence ci-haut, retravaillez votre code du TP 2 !

Énumération des cycles hamiltoniens d'un graphe complet

Le code de la page suivante est un exemple de programme permettant d'énumérer tous les cycles hamiltoniens d'un graphe complet et de déterminer celui qui est de longueur minimum.

```
void afficherCycleHam(int *t, int n, int longueur) {
  int i;

  printf("\nLongueur : %d | ", longueur);
  for (i = 0; i < n; i++)
    printf("%d ", t[i]);
  printf("\n");
}

void permuter(int *t, int i, int j) {
  int temp = t[i];
  t[i] = t[j];
  t[j] = temp;
}

void permutationsCycles(int *t, int i, int n, int longueur, graphe_t *graphe,
                        int *tabMin, int *min) {
  int j, fin = 0;

  if (i == n) {
    longueur += graphe->mat[t[n-1]][t[0]];
    afficherCycleHam(t, n, longueur);
    if (longueur < *min) {
      *min = longueur;
      for (j = 0; j < n; j++) {
        tabMin[j] = t[j];
      }
    }
  }
  else {
    j = i;
    while (!fin && j < n) {
      permuter(t, i, j);
      longueur += graphe->mat[t[i-1]][t[i]];
      permutationsCycles(t, i+1, n, longueur, graphe, tabMin, min);
      longueur -= graphe->mat[t[i-1]][t[i]];
      permuter(t, i, j);
      if (t[1] == n - 1)
        fin = 1;
      j++;
    }
  }
}

int enumCyclesHam(graphe_t *graphe) {
  int n = graphe->nSommets, i, longueur = 0, longueurMin = INT_MAX;
  int *tabSommets = (int *) malloc(sizeof(int) * n);
  int *tabCycleHamMin = (int *) malloc(sizeof(int) * n);

  for (i = 0; i < n; i++) {
    tabSommets[i] = i;
    tabCycleHamMin[i] = 0;
  }
  permutationsCycles(tabSommets, 1, n, longueur, graphe, tabCycleHamMin, &longueurMin);
  printf("\nCycle hamiltonien minimum :\n");
  for (i = 0; i < n; i++)
    printf("%d ", tabCycleHamMin[i]);
  printf("\nLongueur : %d\n", longueurMin);
  free(tabSommets);
  free(tabCycleHamMin);

  return 0;
}
```

2) Reprenez ce code et adaptez-le pour qu'il fonctionne dans votre programme selon vos structures de données. Vérifiez bien que vous obtenez le résultat suivant :

```
Longueur : 21 | 0 1 2 3 4 5

Longueur : 23 | 0 1 2 3 5 4

Longueur : 21 | 0 1 2 4 3 5

Longueur : 19 | 0 1 2 4 5 3

…

Longueur : 20 | 0 5 2 3 4 1

Cycle hamiltonien minimum :
0 1 2 5 4 3
Longueur : 15
```

Vous noterez que dans ce code, une bonne partie des doublons ne sont pas affichés mais il en reste quand même quelques uns. Libres à vous d'améliorer le code pour qu'il n'y ait plus aucune répétition. (Attention : ne pas exécuter sur de gros problèmes !)

Vous connaissez maintenant toutes les solutions possibles à ce problème. Par contre, cette façon de faire n'est pas très efficace. Il est maintenant temps de faire mieux par une méthode par séparation et évaluation. Nous avons toutefois préalablement besoin de certaines fonctions permettant de calculer la borne, évaluer un sommet et vérifier si une évaluation est exacte.

Calcul de la borne – Heuristique du plus proche voisin

L'heuristique du plus proche voisin permet de trouver une solution approchée au problème du voyageur de commerce. Elle consiste à sélectionner un sommet de départ arbitraire, puis de choisir successivement le sommet non encore sélectionné qui est de plus courte distance du dernier sommet sélectionné. L'algorithme se termine lorsque tous les sommets ont été sélectionnés.

3) Écrivez une fonction implémentant l'heuristique du plus proche voisin à partir du sommet 0. Vérifiez que vous obtenez bien le résultat suivant :

```
Heuristique du plus proche voisin a partir du sommet 0 :
0 3 5 2 1 4
Longueur : 17
```

Cette solution sera utilisée comme borne initiale de la méthode par séparation et évaluation.

Fonction d'évaluation – Arbre couvrant de poids minimum excluant le sommet 0 + les deux arêtes les plus légères adjacentes au sommet 0

Une méthode par séparation et évaluation nécessite l'utilisation d'une fonction d'évaluation qui permettra d'évaluer un sommet de l'arborescence en cours de construction. Dans la méthode proposée, cette évaluation consistera en le poids d'un arbre couvrant de poids minimum du graphe excluant le sommet 0, auquel on ajoutera ensuite la somme des poids des deux arêtes les plus légères adjacentes au sommet 0.

4) Écrivez une fonction permettant de générer un arbre couvrant de poids minimum du graphe excluant le sommet 0. Écrivez ensuite le code permettant de déterminer les deux arêtes les plus légères adjacentes au sommet 0 et de calculer la longueur totale du graphe ainsi obtenu. Vérifiez que vous obtenez bien le résultat suivant :

```
Arbre couvrant de poids minimum excluant le sommet 0 :
2 5
1 2
3 5
1 4
Longueur de l'arbre : 8

Aretes sommet 0 :
0 3
0 1

Longueur totale : 14
```

La longueur du graphe ainsi obtenu correspond à l'évaluation de la racine de l'arborescence de séparation et évaluation. Ce calcul, qui servira pour l'évaluation de chaque sommet de l'arborescence durant l'application de la méthode de séparation et évaluation, génère une solution qui peut être exacte ou non. Elle est exacte si le graphe obtenu est un cycle hamiltonien, ce qu'il faut donc vérifier.

Vérification de l'obtention d'un cycle hamiltonien

Un graphe est un cycle hamiltonien dans lequel tout sommet a un degré égal à 2. À partir de l'ensemble des arêtes d'un graphe, il est donc possible de vérifier si le graphe est hamiltonien ou non. Dans l'application de la méthode par séparation et évaluation, ceci permettra de définir si l'évaluation est exacte ou non.

5) Écrivez une fonction qui, à partir d'un arbre couvrant de poids minimum d'un graphe excluant le sommet 0 et des 2 arêtes les plus légères adjacentes au sommet 0, détermine si le graphe ainsi obtenu est un cycle hamiltonien. Après application au graphe obtenu à la question précédente, vous vous en douterez, vous devriez obtenir le résultat suivant :

```
Ce graphe n'est pas un cycle hamiltonien
```

Méthode par séparation et évaluation

Maintenant que nous savons calculer une borne, évaluer un sommet et déterminer si une évaluation est exacte ou non, il reste à programmer la méthode par séparation et évaluation. Le pseudo-code suivant donne un aperçu de la méthode en utilisant une stratégie de développement en profondeur et en supposant que la borne initiale a été calculée au préalable :

```
Évaluer le sommet par la fonction d'évaluation
Si l'évaluation obtenue est plus petite que la borne
   Vérifier si le graphe obtenu est un cycle hamiltonien
   Si le graphe est un cycle hamiltonien (évaluation exacte)
      Mettre à jour la borne avec la longueur du cycle
   Sinon
      Trouver dans le graphe un sommet de degré strictement supérieur à 2
      Pour chaque arête de ce sommet
         Interdire l'arête                  /*Mettre un poids infini*/
         Separation-Evaluation              /*Avec l'arête interdite*/
         Lever l'interdiction de l'arête    /*Remettre le poids initial*/
```

6) Écrivez une fonction implémentant une méthode par séparation et évaluation pour le problème du voyageur de commerce (bien qu'il soit possible de la développer de façon récursive, il est fortement conseillé de le faire de façon itérative). Vous afficherez ensuite la solution optimale du problème, qui devrait être la même que celle obtenue par la fonction d'énumération développée précédemment.

7) Comparez les solutions et les temps d'exécution obtenus sur les problèmes graphe12.txt, graphe13.txt, graphe14.txt et graphe15.txt disponibles sur le site Web du professeur (http://cosy.univ-reims.fr/~pdelisle/enseignement.php).

8) Maintenant, améliorez votre programme pour aller plus vite (si vous le pouvez) !