Algorithmes pour les graphes - TD sur plateforme
Exercice 4 : Résolution par *branch and bound*

Nous ne pouvons pas espérer énumérer tous les circuits hamiltoniens en un temps raisonnable dès lors que le graphe comporte plus de 15 sommets. Rappelons qu'il existe $14!$ (soit près de 8 milliards) circuits différents commençant par 0 quand le nombre de sommets est égal à 15. Cette explosion combinatoire est inévitable dans la mesure où le problème est $\mathcal{NP}$-difficile. Cependant, nous pouvons reculer le moment de l'explosion en coupant les branches de l'arbre de recherche au plus tôt. L'idée est d'appeler une fonction d'évaluation (appelée *bound*) au début de chaque appel récursif. Cette fonction *bound* calcule une borne inférieure du coût du plus court chemin allant du dernier sommet de vus (i.e., `vus[nbVus-1]`) jusqu'au premier sommet de vus (i.e., `0`), et passant par chaque sommet de $nonVus$ exactement une fois. Si la longueur du chemin défini par `vus[0..nbVus-1]` ajoutée à cette borne est supérieure à la longueur du plus court circuit trouvé jusqu'ici (i.e., `pcc`), alors nous pouvons en déduire qu'il n'existe pas de solution commençant par vus, et il n'est pas nécessaire d'appeler *permut* récursivement (nous disons dans ce cas que la branche est coupée).

La fonction d'évaluation la plus simple que nous puissions imaginer est : $(|nonVus| + 1) * dMin$, où $dMin$ est la plus petite longueur d'un arc du graphe. Cette fonction a l'avantage d'être calculable en temps constant à chaque appel récursif (en supposant que $dMin$ est évalué une fois pour toute au début de la recherche).

Votre travail : Vous devez implémenter la procédure `permut` :

```
int permut(int* vus, int nbVus, int* nonVus, int nbNonVus, int longueur, int pcc, int dMin)
```

telle que :
- le tableau `vus[0..nbVus-1]` contient les sommets de la liste vus,
- le tableau `nonVus[0..nbNonVus-1]` contient les sommets de l'ensemble $nonVus$,
- la variable `longueur` contient la longueur du chemin correspondant à `vus[0..nbVus-1]`,
- la variable `pcc` contient la longueur du plus court circuit trouvé depuis le début,
- la variable `dMin` contient la longueur du plus petit arc du graphe.

La postrelation de cette fonction est la même que pour l'exercice 3, mais vous ajouterez une étape d'évaluation pour réduire l'espace de recherche.

Code fourni (téléchargeable sur `http://liris.cnrs.fr/csolnon/TPTSP/code4.c`) : Procédure `main` appelant votre procédure `permut`, et procédure `creeCout` initialisant la variable globale `cout` définissant les coûts des arcs.

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

int** cout;

void creeCout(int nbSommets){
    int coutMin = 10;
    int coutMax = 40;
    int i, j, iseed, it;
    iseed = 1;
    cout = (int**) malloc(nbSommets*sizeof(int*));
    for (i=0; i<nbSommets; i++){
        cout[i] = (int*) malloc(nbSommets*sizeof(int));
        for (j=0; j<nbSommets; j++){
            if (i == j) cout[i][j] = coutMax+1;
            else {
                it = 16807 * (iseed % 127773) - 2836 * (iseed / 127773);
                if (it > 0) iseed = it;
                else iseed = 2147483647 + it;
                cout[i][j] = coutMin + iseed % (coutMax-coutMin+1);
            }
        }
    }
}
```

```
int permut(int* vus, int nbVus, int* nonVus, int nbNonVus, int longueur, int pcc, int dMin){
    /*
     Variable globale :
     - pour tout couple de sommets (i,j), cout[i][j] = cout pour aller de i vers j
     Entree :
     - nonVus[0..nbNonVus-1] = sommets non Vus
     - vus[0..nbVus-1] = sommets Vus
     - pcc = longueur du plus court circuit trouve depuis le premier appel à branch
     - dMin = plus petit cout d'un arc
     Precondition :
     - nbVus > 0 et Vus[0] = 0 (on commence toujours par le sommet 0)
     - longueur = somme des couts des arcs du chemin <vus[0], vus[1], ..., vus[nbVus-1]>
     Postcondition : Soit C l'ensemble des circuits commencant par Vus[0..nbVus-1] et visitant ensuite
     S'il existe un circuit de C de longueur inferieure a borne, alors retourne la longueur du plus pet
     */
    // INSEREZ VOTRE CODE ICI !
}

int main(){
    int nbSommets, i, j, pcc, dMin;
    scanf("%d",&nbSommets);
    int vus[nbSommets];
    int nonVus[nbSommets-1];
    creeCout(nbSommets);
    dMin = INT_MAX;
    for (i=0; i<nbSommets; i++)
        for (j=0; j<nbSommets; j++)
            if ((i != j) && (dMin > cout[i][j]))
                dMin = cout[i][j];
    for (i=0; i<nbSommets-1; i++)
        nonVus[i] = i+1;
    vus[0] = 0;
    pcc = permut(vus,1,nonVus,nbSommets-1,0,INT_MAX,dMin);
    printf("%d\n",pcc);
    return 0;
}
```

Exemple d'exécution : Les temps CPU (en secondes) sont donnés à titre indicatif, pour un processeur 2,6 GHz Intel Core i5.

Entrée	Sortie	Temps CPU
4	72	0.00
6	91	0.00
8	123	0.00
10	134	0.00
12	161	0.00
14	182	0.02
16	198	0.10
18	230	2.16
20	261	107.28