UNIVERSITÉ DE SHERBROOKE
DÉPARTEMENT D'INFORMATIQUE

Devoir # 5
IFT 436
Algorithmes et structures de données

Devoir à remettre au plus tard le 30 novembre 2018 avant 10h30. **Exceptionnellement, ce devoir peut être fait en équipe de trois personnes**. Il peut être également fait en équipe de quatre personnes sous la condition suivante : les deux représentations d'un graphe (matrice d'adjacence et listes d'adjacence) sont utilisées dans chacun des trois algorithmes (i.e., deux versions de chaque algorithme).

Réalisez une étude de performance comparative du temps de calcul de trois algorithmes pour le problème du voyageur de commerce. Un de ces algorithmes peut être celui du devoir #2, dont la solution est donnée à la page suivante pour une variante du problème (le départ n'est pas nécessairement à partir de v_1), les autres peuvent être des heuristiques ou des algorithmes de votre choix, mais qui retournent une solution approximative (pas nécessairement optimale).

Le problème du voyageur de commerce à résoudre est le suivant.

> Soit un ensemble de n villes $\{v_1, v_2, \ldots, v_n\}$ et une fonction de distance d entre chaque paire de villes distinctes telle que $d(v_i, v_j) = d(v_j, v_i)$ (symétrie). Trouver le plus court chemin qui permet à un voyageur de commerce de visiter chaque ville une et une seule fois à partir de v_1 tout en revenant à son point de départ.

Algorithme naïf selon la stratégie « force brute »

```
1.  function VoyageurCommerce(G)
      {p est un tableau d'une permutation de villes numérotées 1 à n}
2.    p_0 := p := [1, 2, 3, ..., n]
3.    min := ∞      chemin := ∅
4.    do  {Boucler sur les permutations des villes}
        {Calculer le coût du chemin représenté par p}
5.        c := Coût(p);
6.        if c < min then
7.            min := c      chemin := p
        {Générer la permutation qui suit p dans l'ordre lexicographique}
8.        p := NextPermutation(p)
9.    while p ≠ p_0
10.   return (chemin, min)
```

Le pseudocode de la fonction *NextPermutation* est dans les notes complémentaires du chapitre 3 (transparents #58 des graphes non orientés). Une telle fonction existe aussi dans la librairie standard de *C++* :

http ://www.cplusplus.com/reference/algorithm/next_permutation

Notez que la fonction *Coût* dépend de la représentation du graphe orienté en mémoire soit par une matrice d'adjacence, soit par des listes d'adjacence.

Heuristiques pour le problème du voyageur de commerce

Puisque le problème du voyageur de commerce est un problème de la classe NP (et même NP-complet) et que « probablement » P $\neq$ NP, plusieurs chercheurs ont proposé des heuristiques afin de calculer des solutions « acceptables » pour des instances particulières de ce problème. Rappelons qu'une heuristique est une méthode de résolution de problème basée sur des règles du pouce minutieusement choisies en fonction d'un ensemble d'instances donné d'un problème difficile. Généralement, une heuristique ne permet toujours d'obtenir une solution optimale, mais parfois « proche ». Il existe deux types d'heuristiques : celles propres à la construction d'une solution et celles propres à l'amélioration d'un solution. Dans ce devoir, on considère seulement des heuristiques du premier type.

Les heuristiques propres à la construction d'une solution construisent, comme leur qualificatif l'indique, une solution en partant de zéro. À chaque itération, une décision est prise concernant l'élément (selon la terminologie de la stratégie gloutonne) à ajouter à la solution, par exemple, l'ajout d'une ville dans un tour. La décision repose donc sur le choix d'un élément qui minimise ou maximise une estimation du coût de la solution. Voici quelques-unes de ces heuristiques pour le problème du voyageur de commerce. [1]

— *Nearest neighbor heuristic* — Cette heuristique choisit aléatoirement une ville de départ (qui est la ville courante) et construit itérativement un tour en lui ajoutant la ville, absente dans le tour, la plus proche de la ville courante (la ville ajoutée devient la ville courante).

— *Nearest insertion heuristic* — Cette heuristique commence avec un tour de deux villes (le tour courant). À chaque itération, elle ajoute la ville, absente dans le tour, dont la distance est minimale par rapport à toutes les villes du tour. Aussi, le ville est ajoutée dans le tour courant de façon à minimiser la longueur du nouveau tour (qui devient le tour

1. Franz Rothlauf, Design of Modern Heuristics : Principles and Application, Springer, Berlin, 2011, pp. 85–88 (en réserve à la bibliothèque).

courant). Ainsi, elle peut être, soit ajoutée au début ou à la fin, soit insérée entre deux villes.

— *Cheapest insertion heuristic* — Cette heuristique est semblable à la précédente, mais elle choisit la ville, absente dans le tour, qui augmente le moins possible le coût du tour.

— *Furthest insertion heuristic* — Cette heuristique commence avec un tour de deux villes qui sont les plus éloignées l'une de l'autre (le tour courant). À chaque itération elle ajoute la ville, absente dans le tour, qui augmente le plus la longueur du tour lorsqu'elle est insérée dans la meilleure position du tour courant (le nouveau tour devient le tour courant). L'idée derrière cette heuristique est d'insérer en premier les villes qui sont les plus éloignées les unes des autres.

La réalisation de cette étude doit respecter le plan qui suit.

1. Programmation de trois algorithmes ou heuristiques de votre choix en une ou deux versions selon l'utilisation d'une ou de deux représentations (matrice d'adjacence et listes d'adjacence) d'un graphe.
2. Utilisation de structures de données uniformes afin de ne pas biaiser les résultats.
3. Programmation des algorithmes dans le même langage de programmation afin de ne pas biaiser les résultats. Aucun langage de programmation n'est imposé.
4. Génération aléatoire d'instances du problème du voyageur de commerce, c'est-à-dire avec différents ordres de graphes, différentes tailles de graphes et différentes distances entre les villes.
5. Construction à la main ou aléatoire d'instances du problème du voyageur de commerce en imposant la contrainte suivante sur la fonction de distance, $d(v_i, v_j) \leq d(v_i, v_k) + d(v_k, v_j)$.
6. Mesure du temps réel de calcul pour chaque algorithme et cela pour chaque échantillon de données d'entrée (instance du problème) si cela est possible.
7. Report des résultats sur un ou plusieurs graphiques. À partir des points obtenus, dessinez une courbe (ligne brisée, interpolation, etc.), une pour chaque algorithme, pour un total **d'au moins trois** courbes.
8. Report de la différence des longueurs des chemins obtenus par un algorithme avec solution exacte et par un algorithme avec une heuristique sur un ou plusieurs graphiques (selon la nature des données).

9. Comparaison des résultats expérimentaux avec les résultats théoriques. Vous devez comparer les courbes obtenues au point précédent avec les courbes théoriques (que vous devez estimer).
10. Conclusion de l'étude.

Vous devez respecter les règles présentées dans le chapitre 2 quant à l'utilisation du générateur de nombres aléatoires et à la prise des mesures du temps de calcul (à moins de justifier parfaitement le non-respect de l'une ou l'autre de ces règles).

Vous devez me remettre :

— le code qui génère les échantillons de données d'entrée (2 points, incluant l'utilisation correcte du générateur de nombres aléatoires) ;
— le code qui fait l'appel des algorithmes (et cela pour chaque échantillon de données d'entrée) et qui prend les mesures (2 points, incluant une prise des mesures du temps de calcul correcte) ;
— le code des trois algorithmes ;
— les scripts s'il y a lieu ;
— un rapport substantiel (d'au moins 10 pages) contenant, entre autres, la description de votre expérience, la description des algorithmes, le graphique ou les graphiques et vos conclusions (6 points).

Vous devez également faire une démonstration du déroulement de votre expérience sur un des ordinateurs du département ou encore sur votre propre ordinateur, mais à l'université, où **chaque membre de l'équipe doit être sur place pour présenter sa partie du travail** (retrait de 5 points si cela n'est pas fait).

Pendant votre démonstration, vous devrez exécuter les trois algorithmes avec une ou plusieurs petites instances du problème pour des fins de vérification d'exactitude.

Vous trouverez à la page suivante une liste de points à vérifier. Vous devez annexer cette liste à votre rapport, en cochant chaque case pour laquelle le point correspondant est vérifié. Si tel n'est pas le cas, vous devez fournir des explications qui justifient toute divergence.

Point à vérifier	Case à cocher	Explication si divergence
Votre étude comporte au moins trois algorithmes.		
Les graphes sont représentés par une matrice d'adjacence		
Les graphes sont représentés par des listes d'adjacence		
La programmation des algorithmes est uniforme.		
Les structures de données sont uniformes d'un algorithme à l'autre.		
Les complexités de calcul des algorithmes sont présentes.		
Les complexités en espace des algorithmes sont présentes.		
Les stratégies de conception des algorithmes sont différentes.		
Au moins 50 échantillons de données ont été générés.		
La taille des échantillons de données varie progressivement de 10 à 10 000 (valeurs de m—taille du graphe et n—ordre du graphe).		
Des échantillons de données considèrent les meilleurs cas (si applicable).		
Des échantillons de données considèrent les pires cas (si applicable).		
Chaque échantillon a été soumis à chacun des algorithmes.		
La prise de mesure des temps de calcul a été réalisée correctement.		
La génération des données aléatoires n'est pas biaisée.		
La comparaison des temps de calcul des algorithmes a été mise en évidence sous la forme d'un ou plusieurs graphiques.		
La comparaison des temps de calcul des algorithmes a été mise en évidence avec les résultats théoriques sous la forme de d'un ou plusieurs graphiques.		
L'environnement d'exécution ne biaise pas les temps d'exécution.		
Les références à des travaux empruntés sur le Web ou ailleurs sont présentes dans le rapport.		
Le rapport comporte au moins 10 pages (sans ce tableau).		