Perfectionnement à la programmation en C

Fiche de TP 6

L2 Informatique 2017-2018

Le voyageur de commerce

Ce TP se déroule en trois séances et est à faire par binômes. Le travail réalisé doit être envoyé au plus tard exactement une semaine après le début de la 3e séance de TP. Il sera disposé sur la plate-forme prévue à cet effet et constitué des programmes répondant aux questions et des éventuels fichiers annexes qui peuvent être demandés.

Contrairement à certains sujets de TP précédents, nous donnons ici une spécification du projet à lire et à mener à bien. Les définitions de types et de fonctions sont laissées libres de choix.

Voici maintenant la spécification du projet.

***

*

Description générale

Le *problème du voyageur de commerce* décrit une situation qui prend en entrée un ensemble de villes avec leur position géographique, et qui consiste à fournir en réponse un ordre de visite de ces villes qui soit le plus court possible.

Plus précisément, l'utilitaire à concevoir, nommé `pvc`, prend en entrée un fichier texte dont chaque ligne est de la forme

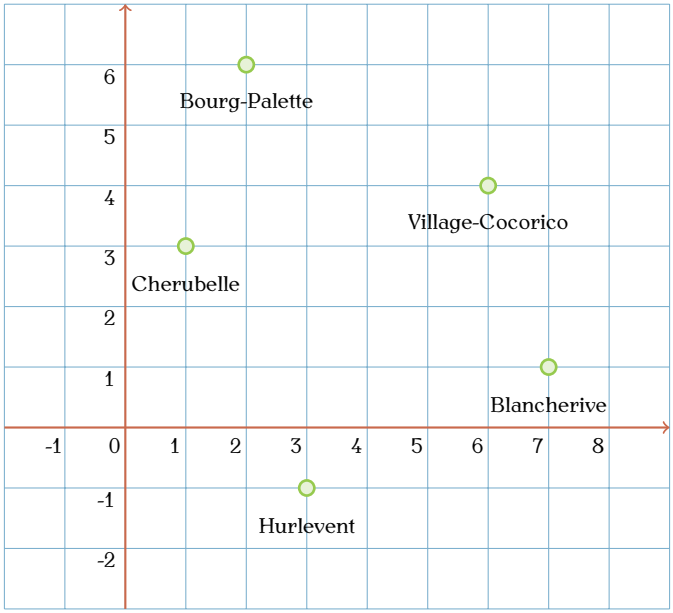
```
VILLE X Y
```

où `VILLE` est un nom de ville et `X` et `Y` sont des entiers qui spécifient la position de la ville dans le plan cartésien discret (`X` est l'abscisse et `Y` est l'ordonnée). Le nom d'une ville ne contient que des caractères alphabétiques ou le caractère `'-'`. Chaque ligne du fichier contient ainsi les données nécessaires à la représentation d'une ville. On appelle *carte* la configuration de villes décrite par un tel fichier.

Par exemple, le fichier texte de contenu

```
Bourg-Palette 2 6
Blancherive 7 1
Hurlevent 3 -1
Cherubelle 1 3
Village-Cocorico 6 4
```

spécifie la carte

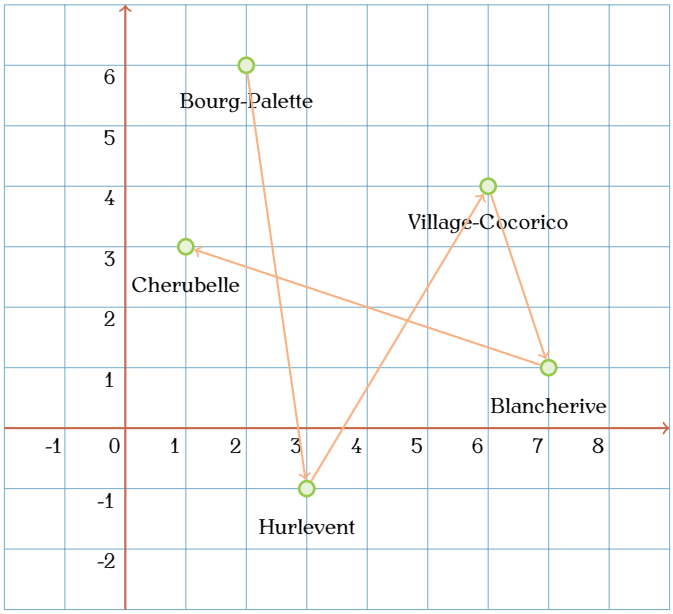


Une *visite* est un ordre de parcours des villes de la carte. La *longueur* d'une visite est le nombre total de kilomètres que demande la visite pour être effectuée [1]. Les visites sont enregistrées dans des fichiers texte sous la forme d'une ligne contenant les villes à visiter dans l'ordre, séparées par une espace. À la fin de la ligne, figure la longueur totale de la visite :

```
VILLE_1 VILLE_2 ... VILLE_N LONGUEUR
```

Par exemple, en reprenant la carte de l'exemple précédent,

1. Chaque unité vaut 1 kilomètre.



illustre la visite des villes de la carte dans l'ordre Bourg-Palette, Hurlevent, Village-Cocorico, Blancherive, Cherubelle. En calculant les distances entre les villes[2], on obtient que la longueur de cette visite est de

$$7.07 \text{ km} + 5.83 \text{ km} + 3.16 \text{ km} + 6.32 \text{ km} \simeq 22.39 \text{ km}.$$

Ce résultat est enregistré dans le fichier texte de contenu

```
Bourg-Palette Hurlevent Village-Cocorico Blancherive Cherubelle 22.39
```

On observe, bien évidemment, que toutes les visites n'ont pas la même longueur. Par exemple, on vérifie sans peine que la visite Bourg-Palette, Cherubelle, Hurlevent, Blancherive, Village-Cocorico est meilleure que la précédente car sa longueur, de 15.27 km, est plus petite.

L'objectif de l'application est de fournir une visite dont la longueur est la plus petite possible en travaillant sur des cartes qui contiennent potentiellement un grand nombre de villes.

Fonctionnement

Une fois lancé, l'utilitaire `pvc` affiche la carte dans une fenêtre graphique. Lorsque la touche `Entrée` est pressée, le programme calcule la meilleure visite possible de la carte donnée selon une méthode expliquée plus loin. Dès qu'il trouve une visite meilleure que celle qu'il a précédemment calculée, il l'affiche sur la sortie standard, la dessine dans la fenêtre graphique (en effaçant au préalable la précédente) et l'écrit dans un fichier texte

2. En utilisant le théorème de Pythagore par exemple.

de nom `CARTE.pvc` où `CARTE` désigne le nom du fichier contenant la carte. Le programme poursuit ainsi ses calculs tant qu'il n'est pas interrompu. Une fois que l'utilisateur considère qu'il a laissé suffisamment de temps de calcul au programme, il peut l'interrompre et lire une solution au problème du voyageur de commerce sur la sortie standard ou bien dans le fichier ainsi créé.

Méthode de calcul

Expliquons à présent une manière efficace pour calculer une visite à partir d'une carte donnée. La première chose à observer est qu'il n'est, en pratique, pas possible d'essayer toutes les visites possibles d'une carte et de ne garder que la meilleure. En effet, une carte de taille 30 demanderait

$$30! = 30 \times 29 \times \cdots \times 2 \times 1 = 265252859812191058636308480000000$$

traitements de visites, ce qui est prohibitif. Nous n'allons donc pas chercher la meilleure solution mais seulement *une bonne solution*, compromis entre temps de calcul raisonnable et longueur faible de la visite.

Appelons C la carte sur laquelle nous travaillons et n le nombre de villes qu'elle contient. La méthode de recherche d'une bonne visite sur C se décrit de la manière suivante :

1. **générer** de manière aléatoire une liste $V = [v_1, \ldots, v_k]$ de visites sur C. L'entier k est fixé (une valeur habituelle pour k est de 128) ;
2. **trier** V dans l'ordre croissant selon la longueur des visites. De cette manière, la meilleure visite de V est celle située en tête. Elle se note v_1 ;
3. **construire** une nouvelle liste V' de visites de la manière suivante.

   Placer dans V' :

   (a) les visites mut(v_1), mut(v_2), …, mut(v_α), où mut est une opération de mutation appliquée sur une visite et α est un entier strictement positif. Le fonctionnement de mut est expliqué plus loin ;

   (b) les visites v_1, v_2, …, v_β provenant de V, où β est un entier strictement positif ;

   (c) un nombre γ de visites générées aléatoirement, où γ est un entier strictement positif.

   Les entiers strictement positifs α, β et γ sont fixés pour toute la durée du calcul et vérifient $\alpha + \beta + \gamma = k$. Ainsi, V' contient k visites, tout comme V ;
4. la **nouvelle liste** de visites V est la liste V' ;
5. **tant que** le programme n'est pas interrompu, aller à l'étape 2.

On peut ainsi espérer qu'à chaque itération du calcul, la meilleure visite contenue dans V possède une longueur de plus en plus petite. Cette liste V s'appelle la *population* et les visites qu'elle contient sont ses *individus*. Le schéma global consiste donc, à partir d'une population d'individus générés aléatoirement, à améliorer les individus petit à petit et de manière aléatoire. L'espoir est d'obtenir un individu qui forme une bonne solution au problème.

Décrivons à présent l'opérateur de *mutation* mut. L'idée consiste à modifier légèrement et aléatoirement une visite pour la rendre potentiellement meilleure. Une visite est vue ici comme une liste de villes. Ainsi, si v est une visite, $v = [a_1, \ldots, a_n]$ où a_1, …, a_n sont les n villes de la carte. La visite mut(v) est la visite calculée par le procédé suivant :

1. générer aléatoirement un entier ℓ compris entre 1 et $\lfloor \frac{n}{2} \rfloor$;
2. générer aléatoirement deux entiers i et j (vérifiant des conditions de cohérence à déterminer) ;
3. remplacer dans v le facteur $a_i, a_{i+1}, \ldots, a_{i+\ell-1}$ par le facteur $a_j, a_{j+1}, \ldots, a_{j+\ell-1}$. Ces deux facteurs ne doivent pas se chevaucher. Cette nouvelle visite constitue la ville calculée par mut(v).

Par exemple, considérons la visite symbolisée par la permutation

$$\sigma := 478219365.$$

Si les entiers $\ell = 3$, $i = 2$ et $j = 7$ ont été générés, on obtient

$$\text{mut}(\sigma) = 436519782.$$

On observe bien en effet que le facteur rouge et le facteur bleu de σ ont été échangés pour former mut(σ). Ces facteurs sont bien spécifiés par ℓ (la longueur), i (la position du début du facteur rouge) et j (la position du début du facteur bleu).

Précisions

Les paramètres α, β et γ doivent pouvoir être choisis par l'utilisateur (des valeurs par défaut sont proposées qui peuvent dépendre du nombre n de villes de la carte). Une phase d'expérimentation est importante pour trouver de bonnes valeurs par défaut.

Une gestion efficace d'erreur doit protéger l'application. Tout ce qui peut provoquer une erreur doit être anticipé (format de carte non valide, coordonnées de villes non cohérentes, *etc.*).

Tout ajout ou toute optimisation pertinente est le bienvenu. Par exemple,

— la possibilité de saisir une carte en cliquant dans une fenêtre graphique de sorte à créer des coordonnées (x, y) pour chaque ville ;

— la connexion de l'application à un site internet contenant une base de données de distances entre villes ;

— la prise en compte, en plus des distances entre villes, d'un coût de transport. La minimisation devra se faire par une fonction linéaire $v \mapsto \lambda_d d + \lambda_c c$ sur la distance et le coût. Elle associe à chaque visite v la quantité mentionnée, où d est la longueur de la visite, c son coût total, et λ_d et λ_c des coefficients rationnels choisis par l'utilisateur ;

— la possibilité de spécifier pour certaines villes de la carte des priorités de visite (par exemple, la ville a_3 doit être visité avant la ville a_7, qui elle même doit être visitée avant la ville a_1).

***

*

Exercice 1. (Architecture du projet)

L'objectif de cet exercice est de concevoir une architecture viable pour le projet présenté.

1. Lire attentivement **l'intégralité du sujet** avant de commencer à répondre aux questions. Synthétiser ce qui est demandé de sorte à résumer l'information contenue dans le sujet en moins de la moitié d'une page (sans compter d'éventuels — mais importants — schémas).

2. Une fois ceci fait, proposer un découpage en modules cohérent du projet. Pour chaque module proposé, décrire les types qu'il apporte ainsi que ses objectifs principaux.

Exercice 2. (Conception du projet)

1. Implanter les modules mis en évidence dans l'exercice précédent. Au fil de l'écriture du projet, il est possible de se rendre compte que le découpage initialement prévu n'est pas complet ou adapté. Si c'est le cas, mentionner l'historique de ses modifications.

2. Maintenir un `Makefile` pour compiler le projet.

3. Dans le rendu final, inclure des cartes d'exemple assez conséquentes (entre 30 et 50 villes) pour que l'application puisse être testée et son efficacité mesurée. Dans le but de réaliser des tests, il est très fortement conseillé de programmer un générateur de cartes aléatoires (les noms des villes importent peu, ces dernières peuvent se nommer `ville-1`, `ville-2`, *etc.*, seules leurs coordonnées sont importantes).

Exercice 3. (Travail théorique)

1. Il n'y a pas grand chose qui soit imposé dans l'implantation de ce projet. Cette grande liberté implique de justifier avec précision les choix effectués dans le rapport, que ce soit pour les algorithmes employés ou les structures de données choisies.
2. Le problème du voyageur de commerce est un problème célèbre. Pour cette raison, une grande bibliographie est présente à son sujet. On inclura ainsi dans le rapport une synthèse d'informations relatives à ce problème (sa description, une explication de sa difficulté, algorithmes d'approximation, *etc.*). Ne pas oublier de citer les sources consultées (et ne s'appuyer que sur celles qui ont de la valeur).