UNIVERSITÉ DE NANTES

POLYTECH NANTES

Mini-projet de c++

PolyHash

Google Hash Code 2018

Étudiants développeurs ⇒ Hanene Ben Ntar, Xinyao Jiang, Sébastien Napoléon, Victor Troussel & Alexis de Guisti

Professeur encadrant ⇒ Fabien Picarougne

Présentation du projet

Dans le cadre du mini-projet de C++ de quatrième année en informatique de Polytech Nantes, il nous a été donné de constituer des groupes de 4 à 5 personnes afin de travailler sur le *final round* du Google Hash Code 2018 dans un temps imparti : du 19 novembre 2018 au vendredi 11 janvier 2019.

Le *final round* du Google Hash Code 2018 a centré un challenge lié à la localisation de la population mondiale, c'est-à-dire qu'en s'appuyant sur des études de la *World Bank*, la population est de plus en plus concentrée dans les villes, les métropoles (50% en 2008 et 54% en 2016). L'objectif de ce défi lancé par la multinationale est ici de créer des villes avec une répartition de bâtiments la plus optimum possible. C'est-à-dire, placer des bâtiments utilitaires (services, pharmacies, magasins hôpitaux…) le plus proche des plus gros regroupement résidentiels de personnes.

L'objectif du travail de réflexion et de programmation ici, est de construire des maps répartissant ces organisations de constructions avec des fichiers d'entrée déjà fournis par l'encadrant du défi, afin d'écrire la solution dans un fichier texte de sortie .out. Afin de pouvoir situer le niveau d'optimisation de la solution, on évalue une notion de score calculée selon le nombre d'utilitaires attribués à chaque bâtiment d'habitation. Cependant les utilitaires entourant les résidences doivent respecter la distance de Manhattan* dictée par chaque Map, c'est-à-dire qu'à partir d'un certain calcul de distance, l'utilitaire n'est plus comptabilisé comme assigné aux habitants. Autre contrainte à prendre en compte dans ce défi est qu'un seul utilitaire du même type peut être comptabilisé par résidence, c'est-à-dire que pour mieux visualiser le problème, il ne peut y avoir qu'une pharmacie entourant un immeuble d'habitations. Dans ce challenge, il y a bien-sûr quelques contraintes à respecter, un nombre de lignes et de colonnes pour la construction du fichier sortant, une distance de Manhattan à respecter dans toute la map traitée pour comptabiliser le score des utilitaires attitrés à chaque résidence, ainsi que le nombre de constructions qui doit sortir lors de la solution. Dans le fichier d'entrée, la première ligne contient les quatre contraintes énoncées précédemment, ensuite la deuxième ligne du fichier nous renseigne sur le type de bâtiment (U pour un bâtiment utilitaire et R pour une résidence), les deux nombres

suivants correspondent respectivement au nombre de lignes puis au nombre de colonnes de chaque construction ensuite la dernière information, nous informe de la capacité d'une habitation ou du type de l'utilitaire, en fonction de la première donnée de la ligne. les lignes suivantes étant composées de points et de dièses, représentent la forme que la construction devra avoir sur la map, le '.' correspondant à un espace vide et le '#' à un élément de construction.

Lors du Google Hash Code officiel, les projets pouvaient être codés sous n'importe quel langage, mais pour notre solution, nous devions apporter une résolution du problème en C++.

*Distance de Manhattan: Aussi connue comme taxi-distance, est la distance entre un point A et B dans un environnement quadrillé, où l'on se déplace à la fois en x et en y de la manière suivante, $|X_B - X_A| + |Y_B - Y_A| = distance\ (A, B)$

Plan du rapport

Différentes phases du projet

Après la constitution du groupe travail de ce projet, il y a deux phases principales de travail sur le challenge qui sont les suivantes :

- La phase d'arbitrage et de mesure de temps

  Deadline : 14 décembre 2018

- La phase de placement de maps et de résolution du problème

Deadline : 11 janvier 2019

Phase d'arbitrage et de mesure de temps

La phase d'arbitrage et de mesure de temps se divise en quatre parties distincte:

- Le parsing des maps
- Validation des solutions
- Calcul du score
- Calcul de temps d'exécution

1-Parsing des maps :

Dès le début, nous nous sommes concentrés sur la construction d'une architecture cohérente et avons fondé une structure solide. Une fois réalisé, le parsing des maps n'a pas posé de réel problème. Le premier parseur que nous avons réalisé était le parseur des maps d'entrées.

```
4 7 2 3
R 3 2 25
.#
##
.#
U 1 4 1
####
U 2 2 5
##
##
```

```
//On recupère la première ligne et on la traite
getline(myfile,line);
int  CPT = 0;
bool isResidence = false;
//On split les caractères avec comme delimiteur " "
sep = split(line, ' ');
this->map = new ;
```

Ces maps d'entrée nous donnent les dimensions de la carte ainsi que la distance de Manhattan maximale.Elles nous donnent également le plan, le type et la capacité de tous les bâtiments que nous pourrons utiliser pour atteindre notre solution.

Le second *parser* que nous avons réalisé était le *parser* des fichiers de sorties, *parser*

qui nous servira au calcul de score ainsi qu'à la validation de la solution.
Celui ci nous indique les coordonnées et l'identifiant des bâtiments placés.
{ insérer map de sorties}

```
//Si le fichier est ouvert
if (myfile.is_open())
{
    getline(myfile,line);
    this->nbBatiment = stoi(line,&sz);

    while(getline(myfile,line)){
        sep = split(line,' ');
        map->listeBatiments.at(stoi(sep[0],&sz))->posX.push_back(stoi(sep[1],&sz));
        map->listeBatiments.at(stoi(sep[0],&sz))->posY.push_back(stoi(sep[2],&sz));
    }
}
```

2- Validation de solution :

Pour le processus de validation, il a s'agit de contrôler les potentiels chevauchements d'un bâtiment sur un autre. On ne prend pas en compte les voisins de même type pour les résidences, ils ne seront simplement pas comptés dans la partie scoring.

3- Calcul de score :

Pour la validation du score, nous avons parcouru les cases des résidences et en se basant sur la distance de Manhattan, nous avon incrémenté le score et nous avons avancé en fonction des cases voisines de celle analysée.

4- Calcul du temps d'exécution :

Pour calculer temps d'exécution, la fonction prend en paramètre le chemin du dossier qui contient les exécutables et itère cette répertoire et pour tous les exécutables trouvés elle le lance 10 fois et calcule le temps d'exécution à chaque fois et fait la moyenne des 10 à chaque.

Phase de placement et de résolution

Notre stratégie pour le placement est la suivante :

-On place l'utilitaire de plus petit périmètre à la position (posFirstX,posFirstY), avec posFirstX=0 et posFirstY=0
-On place les résidences de capacité maximal aux position x et y avec x entre posFirstX - (distanceMax) et posFirstX + (distanceMax) et y entre posFirstY -
(distanceMax) et posFirstY + (distanceMax)
-dDstanceMax est initialisée à la distance max de map
-On itère par rapport à la distanceMax
-Tant que cette distance est <=dmaxCondition(qui est égale à sqrt(2)*) (this->map->sizeX*this->map->sizeY)) on itère
- Et lors d'une itération on effectue ces tâches:
-1. on multiplie distanceMax par 1.5
-On place les utilitaires rur le contour de rectangle dans les cotés sont
-Côté 1 on fixe la colonne à posFirstY-distanceMax et on varie le numéro de ligne entre posFirstX-distanceMax et posFirstX+distanceMax
-Côté 2 on fixe la ligne à posFirstX+distanceMax et on varie le numéro de colonne entre posFirstY-distanceMax et posFirstY+distanceMax
-Côté 3 on fixe la colonne à posFirstY+distanceMax et on varie le numéro de ligne entre posFirstX+distanceMax et posFirstX-distanceMax
-Côté 4 on fixe la ligne à posFirstX -distanceMax et on varie le numéro de ligne entre posFirstY + distanceMax et posFirstY -distanceMax
-(posFirstX, posFirstY, distanceMax);
-On place à l'intérieur de ce rectangles (si c'est possible ==> cases non occupées) les résidences de capacité maximal.

Pour faire fonctionner la fonction de calcul de score :

Pour chaque résidence placée sur la carte, on regarde dans un rayon de 1.5x la distance max et on récupère les bâtiments qui sont dans ce rayon.
après ça , on itère sur chaque cellule de chaque résidence et chaque celle de chaque bâtiment qui sont potentiellement voisin et on essaye de trouver la distance minimale entre chaque résidence et chaque bâtiment.
Si un bâtiment d'un certains type est voisin avec une résidence, on s'assure qu'aucun

bâtiment de ce même type n'est déjà voisin avec la résidence, si c'est le cas, le score pour l'utilitaire courant n'est pas compté

Répartition du travail

- Lors de la première phase de travail, Xinyao et Alexis ont travaillé sur la fonction de score en voulant développer la comptabilisation les cases et en basant la recherche par le voisinage, Sébastien et Victor se sont occupés du parsing des maps ainsi que de la validation de la solution, et Hanene s'est chargé de la fonction de calcul de temps d'exécution
- Lors de la seconde partie, Victor a repris ce qui ne fonctionnait pas lors de la première phase, Les binômes d'Hanene et Sebastien ainsi que celui de Xinyao et Alexis ont développé une solution différente de résolution de map, seulement la première s'est révélée fonctionnelle. Pour le rapport, Tout l'équipe a travaillé dessus pour le peaufiner au mieux.

Sources

Pour les images, elles sont étés prises sur les sites respectifs de Polytech Nantes, L'université de Nantes et Google Hash Code.