Софийски университет „Св. Климент Охидски“

Факултет по математика и информатика
Специалност: “Информационни системи ”
Курс: 3, Група: 1
Дисциплина: “Системи, основани на знания”

ДОКУМЕНТАЦИ

Домашно №1: Намиране на най – къс път

Изготвил:
Георги Иванов Минков (ФН: 71600)

Преподаватели:
Проф. д-р Мария Михайлова Нишева - Павлова
Гл. ас. Йоанис Патиас
Маг. Станимира Желязкова

София
Зимен семестър 2017/2018

Съдържание

1. Цел на задачата

Условието на задачата има за цел да представи проблем, свързан с намиране на най – кратък път в граф (предоставен ни под формата – система за автобусен трафик) от възел до възел. За тази цел ни е предоставена таблица носеща инфомация свързана с времето необходима на даден автобус да стигне от точка i до точка j, където двете точки отговарят на спирки от автобусната система. Проблематиката на задачата е свързана с използване на алгоритъм за неинформирано търсене. Търсенето е неинформирано, поради факта, че не можем да предоставим адекватна евристика спрямо подадените данни, т.е. трябва да направим пълно изчерпване, до достигане на целевото състояние.

2. Изисквания към изпълнението на задачата

За да можем коректно да решим заданието, трябва то да бъде разбито на няколко части.

Първо задаваме входните данни, тези данните представляват времето между две спирки (данните за решаване на проблема, не ни интересуват дали са в часове, минути или секунди, поради факта, че това ще са тегла в нашия граф, тези тегла са независими от гледна точка на мерна единица за пресмятане).

След определяне на входните данни, трябва да се вземе решение за вида търсене, което ще бъде използвано, за тази задача, както посочих по – рано, ще използвам неинформирано търсене. В случая съм избрал използването на алгоритъма на Дийкстра за намиране на минимален път. Отделно за изпълнението на алгоритъма на Дийкстра, прилагаме необходимата методика описана подробно в точка 3.2.

Като последна задача остава запазването и представянето на пътя през който трябва да преминем за да достигнем необходимата цел.

3. Архитектура на приложението и реализация на алгоритъма

За програмното решение на задачата използвам следните инструменти от библиотеките:

- потоци за вход и изход от конзолата -> <iostream>;
- потоци за обработка на файлове -> <fstream>;
- използване на вградени крайни стойности -> <climits>
- структура за запазване на елементи – вектор -> <vector>
- използване на текстови данни -> <string>
- Track.h и Track.cpp -> клас Track, който ще запазва данните въведени за продължителността между отделните спирки (readFromFile

функцията) и ще ги обработва, т.е. да изпълнява алгоритъма на Дийкстра (find функцията), намиране и представяне на пътя (path и printPath)

3.1 Описание на програмата

Ще се фокусираме над класа Track, поради факта, че там е реализиран основният алгоритъм.

За запазване на матрицата използваме двумерна матрица, която се запазва в структурата вектор от вектори – с идентификатор matrix, като в него запазваме теглата на възлите. Останалите член – променливи които използваме са: вектор от булев тип – shortPathTreeVisited, за запазване на възлите, през които сме преминали при изчисляване на дървото на минималните пътища, вектор от целочислен тип – minDistance, за запазване на минималните разстояния от подаден възел до съответния индекс (този индекс служи за номериране на възлите), вектор от целичислен тип – parentPath, служещ за запазване на най – краткия път, from и to променливи служещи за определяне на начална и крайна точка на пътя.

Член – функциите, които ползваме са следните:
- Track() – конструктур за инициализиране на стойностите
- ~Track() - деструктур за унищожаване на динамично зададените променливи
- initialize() – функция за инициализиране на стойности при начало на алгоритъма на Дийкстра
- getMinDistanceIndex() – връща индекса на минимално дълечния съсед
- utility() – помощна функция
- find() – реализация на алгоритъма
- print() и printPath() – използват се за обработка за връщане на най – краткия път

3.2 Описание на алгоритъма

3.2.1. Псевдо код

psheudocode of Dijkstra's Algorithm:

```
function find(from, to):
    initialize(); // инициализиране на началните стойности за векторите
// shortPathVisitedTree, minDistance, parentPath
```

```
        minDistance[from] <- 0 // задаваме разстоянието на източника до
// до себе си за 0

        target <- false // флаг, за оптимизация

        for every node
           minIndex <- getMinDistanceIndex() // изчисляване на индекса на
// възела намиращ се на минимално разстояние от текущия

           if minIndex equal to
              target <- true

           shortPathTreeVisited[minIndex] <- true // маркиране на текущия
// възел като обиколен

           for every adjency (as index adj) of vertex:
              if shortPathThreeVisited[adj] equal false AND isVertices AND
minDistance isNot INFINITE
              AND minDistance[minIndex] + matrix[adj][minIndex] <
minDistance[adj] :
                 parentPath[adj] <- minIndex // добавя елемент към пътя
                 minDistance[adj] <- minDistance[minIndex] +
matrix[adj][minIndex] // актуализиране на дистанцията
```

3.2.2 Описание на алгоритъма

След като сме задали информацията свързана с матрицата на съседство и от която до коя точка трябва да намерим път, започва същината на алгоритъма за търсене на Дийкстра. Първо задаваме начални стойности за структурите, които пазят съответно - дали сме минали през съответния възел и дефакто дървото на минимални пътища – shortPathTreeVisited, начална стойност false; задаваме дължината до всички възли да е „безкрайност“, стойност, която ние сме задали, като само от възела източник (възела „от“) задаваме дължина 0; задаваме възелът от който тръгваме да има стойност -1 във вектора служещ за запазване на пътя през който сме минали. В този вектор пазим на съответния индекс от къде сме дошли, т.е. родителя на текущия възел (под текущ се разбира съответният индекс в даден момент).

Следващата стъпка от алгоритъма е да обходим всички възли или докато не стигнем целевия (при обхождане на всички, рано или късно ще стигнем целевия). В този цикъл първата стъпка е да намерим индекса на елемента, който се намира на най – малко разстояние от текущия, функцията getMinDistanceIndex() се грижи

за това, като самата тя работи по стандартния алгоритъм за намиране на минимален елемент. Крайната задача в цикъла е да обиколим с нов цикъл елементите, които са съседи и ако съществува елемент, който е съсед, т.е. съществува път и сумата от минималната дистанция на минималния индекс + текущия възел е по – малка от минималната дистанция на индекса съсед, то добавяме новия възел към пътя и актуализираме, стойносттите през които сме минали, по този начин премахваме „безкрайностите“.

Чрез използването на този алгоритъм си гарантираме намирането на най – къс път.

4. Проведени тестове

За провеждането на тестове използвам примерната матрица, която е подадена, както и такава, която е свободно измислена.

Примерни входове и съответните им изходи:

Вход: 1 4
Изход: 1 4

Вход: 6 3
Изход: 6 7 3