Shanghai Jiao Tong University

Project 2: 最短路径算法

吕艺

517021910745

2020 年 5 月 7 日

1 实验目的

本项目主要使用 Dijkstra 算法一些衍生算法在有向非负图中计算最短路径。

这里主要涉及一下两个功能:

1. 给出第 0 个点到第 n-1 个点的最短路路径，给出最短路径的大小和路径

2. 给出从顶点 0 到顶点 n-1 的单调最短路路径

2 最短路径实现思路

2.1 算法介绍

这里我使用的是经典的 Dijkstra 算法。

Dijkstra 算法使用了广度优先搜索解决赋权有向图或者无向图的单源最短路径问题，算法最终得到一个最短路径树。该算法常用于路由算法或者作为其他图算法的一个子模块。

Dijkstra 算法采用的是一种贪心的策略，声明一个数组 dist 来保存源点到各个顶点的最短距离和一个记录是否访问过该节点的数组 visit。

1. 将源点到自己的距离最短距离赋值为 0。将源点到其他节点的最短距离设置为无穷。

2. 从 dist 数组选择最小值，在 visit 数组中标记已访问过该节点

3. 查看能否通过源点该节点到达新的节点，并且可以让源点到该新节点的距离变小，如果可以，更新源节点到新节点之间的最短距离。

4. 回到步骤 2，直到图的所有顶点都被访问过

2.2 具体实现

在这里我采用边表来记录从每个节点出发的边以及他们的权值。

为了降低寻找到距离源点最近的节点的时间复杂度，这里我利用优先级队列的方法，对节点到源点的距离进行排序。

由于最短路径可能不止一条，同时可能存在从两个不同的节点到同一个节点的路径长度相等的情况，我为一个节点都维护了一个 parent 的 vector。在每次进行松弛操作的时候，如果通过边的起点的新的距离小于之前的距离，则直接将 parent 数组替换成包含该条边起点的新向量。如果当通过边的起点的新距离等于之前的距离，那么我就将边的起点加入到当前 parent 的数组中。

在输出路径的时候，我从终点开始访问 parent 数组对应的值，直到访问到原点。由于会存在多条路径，即某一节点的 parent 不为 1 的情况，这里采用了 dfs 的算法，输出所有路径。

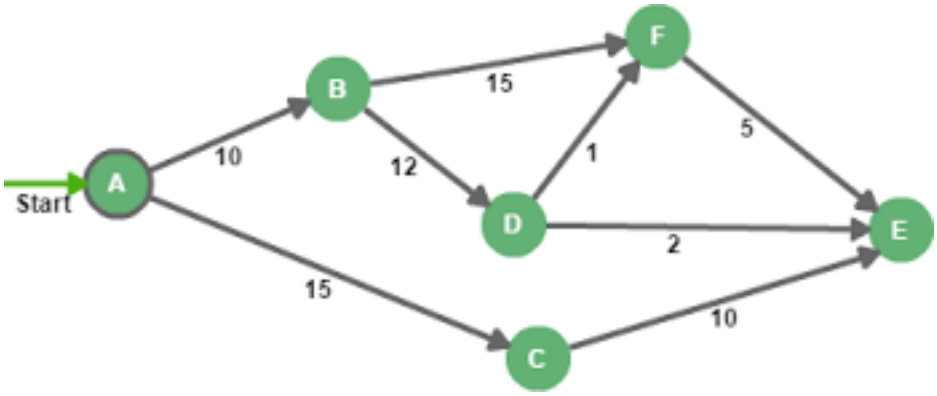

图 1: dijkstra 算法流程图

3 单调最短路径思路

3.1 算法介绍

这里采用了变形后的 Dijkstra 算法。

总体思路与 Dijkstra 算法类似，只是当在进行松弛操作的时候，我们首先比较这条边的权值与之前一条边的权值的大小，如果不满足单调的要求，就忽略这条边。

具体算法如下：

1. 将起点为源点的边，源点到他们的距离以及这条边的权值加入到优先级队列，以源点到他们的距离进行排序。

2. 找出距离源点最近的点，对每条以该节点为起点的边，观察是否该条边的权值大于之前那条比边的权值，如果不符合单调的条件，则跳过

3. 查看能否通过源点该节点到达新的节点，并且可以让源点到该新节点的距离变小，如果可以，更新源节点到新节点之间的最短距离。将该节点，到原点的距离，以及该条边的权值加入到优先级队列中

4. 回到步骤 2，直到图的所有顶点都被访问

3.2 具体实现

在具体实现的时候，我首先对每个节点的边表进行排序，这样可以更快的比较是否这条边的权值大于之前那条边。

之后就像上面介绍的那样运行算法，最后就像 Dijkstra 算法中那样处理 parent 数组，最后利用 parent 数组输出所有单调路径。

4 接口设计

用户接口设计主要参照给定的 Dijkstra.h 文件进行设计，做了一定的修改。

4.1 公共接口

4.1.1 读入测试数据

```
void readFromFile(const char *inputfile = "input.txt");
```

在 readFromFile 接口中，主要实现了从 inputFile 中读入数据，并处理成不同的 Testcase，里面包含每个 testcase 的边表以及节点数目。这些数据就保存在类中。

这里会使用一个字符串处理函数, 将字符串分离成 int 类型的向量返回。

```
vector<int> getNumberFromString(string s);
```

4.1.2 运行功能

```
void run(const char *outputFile = "output.txt");
```

在本函数中会根据测试用例的数量逐个进行测试，调用 run1（生成最短路）和 run2（生成单调路径）两个函数，将结果输出到输出文件里。

4.2 私有接口

4.2.1 最短路算法

```
void run1(int testcase, const char *outputFile);
```

在这个函数中，我主要实现了 Dijkstra 算法，算法的思想和具体实现在第二部分中进行了说明。在运行完 Dijkstra 算法后，我利用 dfs 算法对 parent 数组的结果进行处理，返回所有的最短路径，之后调用 printPathToFile 将数据输出到文件中。

4.2.2 最短单调路径

在这个函数中，我主要实现了变形的 Dijkstra 算法，算法的思想和具体实现在第三部分中进行了说明。在运行完算法后，我利用 dfs 算法对 parent 数组的结果进行处理，返回所有的最短路径，之后调用 printPathToFile 将数据输出到文件中。

4.2.3 输出到文件

```
void printPathToFile(const char *outputFile,int min_dist,
        vector<vector<int>> &all_paths, bool is_run1);
```

这里主要调用 dfs 函数对 parent 数组进行处理，返回所有最短路径，之后输出到文件中。当 is_run1 为 false 的时候，会输出 end。

4.2.4 返回路径

```
vector<vector<int>> dfs(vector<vector<int>> &parent, int src,int dst);
void dfs(vector< vector<int> > &parent, vector<int> temp,
        vector<vector<int>> &all_paths, int cur_node, int src);
```

在这里我从终点开始，对 parent 数组进行访问，依次向前，直到访问到起始节点为止。把路径存储在 all_path 中返回。

5 实验运行说明

1. 编译代码 make
2. 修改 main 文件中的测试文件名称（默认为 input.txt）
3. 运行二进制文件./Dijkstra
4. 结果打印在 output.txt 文件中

6 实验环境

机型：MacBookPro（13-inch, 2018, Four Thunderbolt 3 Ports）
处理器：2.3 GHz Intel Core i5
内存：8GB 1867 MHz DDR3
硬盘：APPLE SSD SM0256G
系统：macOS High Sierra Version 10.13.6

7 项目文件路径组织

```
Dijkstra.cpp        // 功能实现代码
main.cpp            // 运行文件
Dijkstra.h          // 头文件
Makefile            // 编译文件
input.txt           // 输入测试文件
output.txt          // 输出文件
```