Data Structures and Algorithms Laboratory Projects Report

Graph (Shortest Paths Problems)

常家奇 2017141493004 第二组

Date: 2019-12-14

Chapter 1: Introduction

> 迪杰斯特拉算法（英语：Dijkstra's algorithm）由荷兰计算机科学家艾兹赫尔·迪杰斯特拉在1956年提出。迪杰斯特拉算法使用了广度优先搜索解决赋权有向图的单源最短路径问题。该算法存在很多变体；迪杰斯特拉的原始版本找到两个顶点之间的最短路径，但是更常见的变体固定了一个顶点作为源节点然后找到该顶点到图中所有其它节点的最短路径，产生一个最短路径树。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。举例来说，如果图中的顶点表示城市，而边上的权重表示城市间开车行经的距离，该算法可以用来找到两个城市之间的最短路径。

> 该算法的输入包含了一个有权重的有向图G，以及G中的一个来源顶点S。我们以V表示G中所有顶点的集合。每一个图中的边，都是两个顶点所形成的有序元素对。(u, v) 表示从顶点 u 到 v 有路径相连。我们以 E 表示G中所有边的集合，而边的权重则由权重函数 $w: E \to [0, \infty]$ 定义。因此，$w(u, v)$ 就是从顶点 u 到顶点 v 的非负权重（weight）。边的权重可以想像成两个顶点之间的距离。任两点间路径的权重，就是该路径上所有边的权重总和。已知 V 中有顶点 s 及 t，Dijkstra 算法可以找到 s 到 t 的最低权重路径(例如，最短路径)。这个算法也可以在一个图中，找到从一个顶点 s 到任何其他顶点的最短路径。

> 最初的迪杰斯特拉算法不采用最小优先级队列，时间复杂度是$O(|V|^2)$通过斐波那契堆实现的迪杰斯特拉算法时间复杂度是$O(|E| + |V| \log |V|)$ (其中$|E|$是边数) 。对于不含负权的有向图，这是当前已知的最快的单源最短路径算法

Chapter 2: Algorithm Specification

主要数据结构设计说明

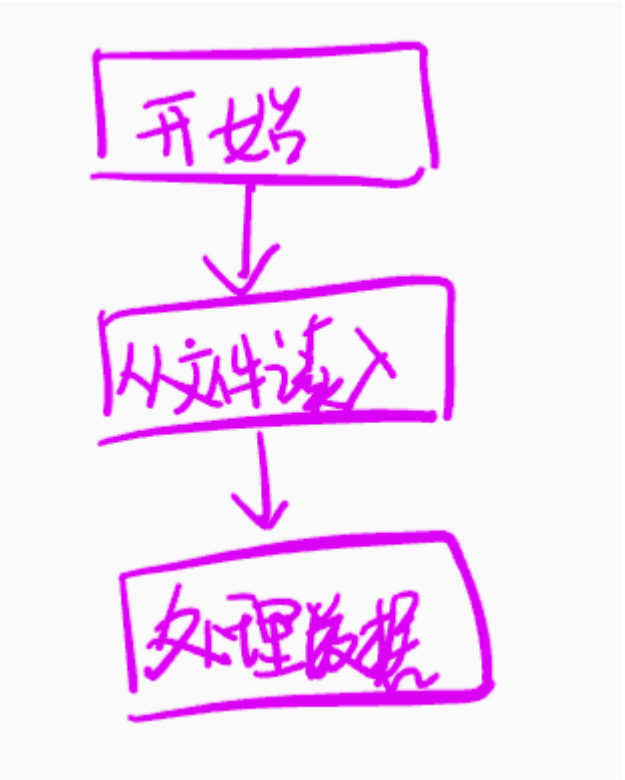
- `vertex` 结构体，代表城市节点。
  - `name` ,代表城市名。
  - `pos_x`,`pos_y` 代表城市坐标。
- `graph` 结构体，代表和图相关的信息。
  - `id_cache` ，代表存储 `vertex` -> `id` 的字典
  - `vertex_cache` ，代表存储 `id` -> `vertex` 的字典
  - `string_to_vertex` ，代表存储 `name` -> `vertex` 的字典
- `graph_data` 结构体，代表和图相关的数据信息
  - `V` ,代表图的节点数目
  - `prev`，`d`，`used` ，代表最短路算法中中间变量。
  - `cost` ，代表有向图中边的权重。

系统设计思想

> 将系统分离为3个模块：读入模块，记录信息模块，最短路模块。

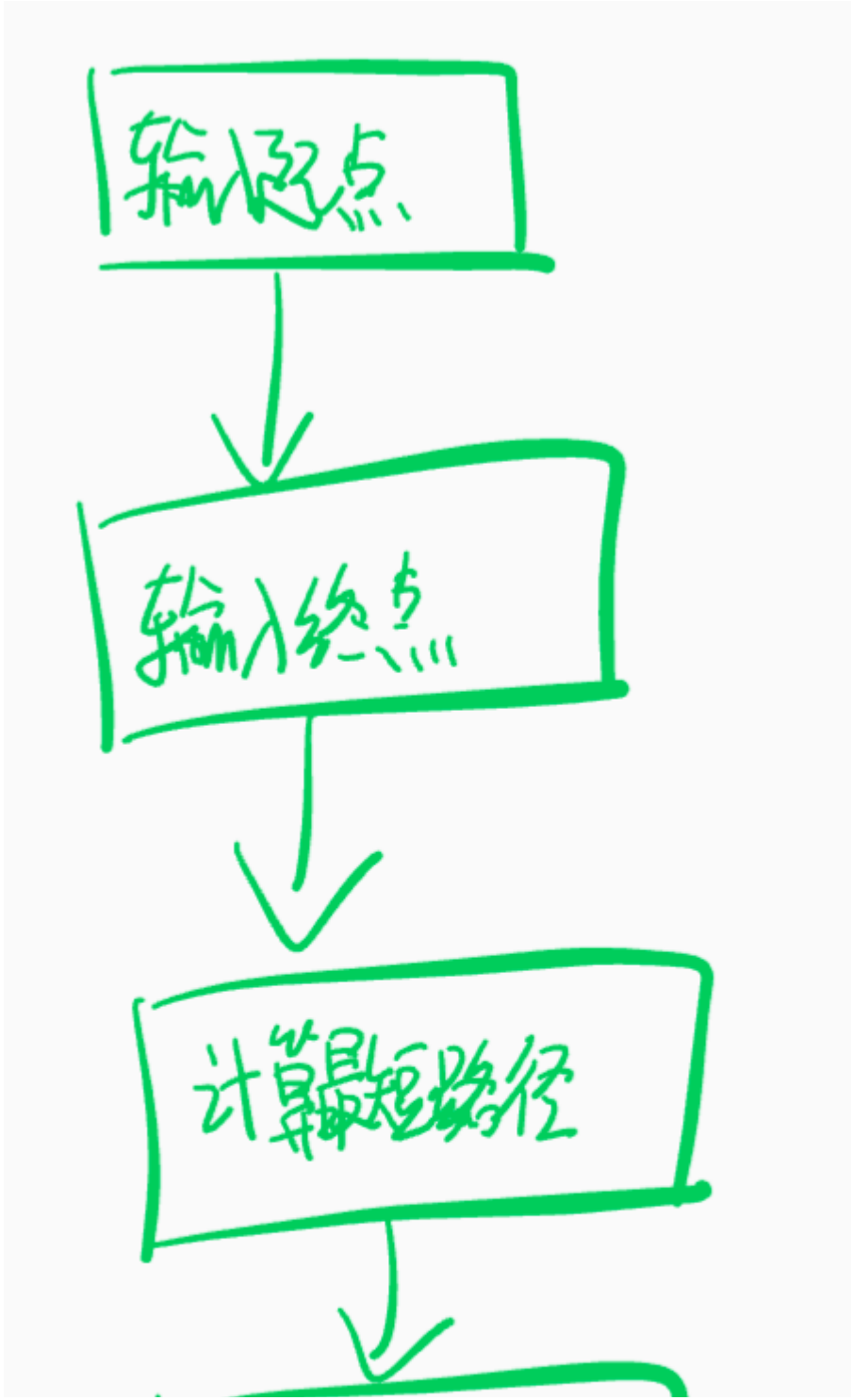
流程图：

读入模块 & 记录信息模块



- 首先，输入包含地图信息的文件名
- 然后，从地图文件中读入所需信息
- 最后，将这些信息保存在图结构中

最短路模块


输入起点
输入终点
计算最短路径

- 首先，输入要求路径的起点
- 然后，输入要求路径的终点
- 其次，根据保存的地图信息计算出最短路径
- 最后，输出最短路径以及路径长度

Chapter 3: Testing Results

Test Case 1:

从 `Seattle` 到 `Washington, D.C.` 的最短路:

- Result:

> Please input mapdata filename:map-usa.txt
> Please input begin city :Seattle
> Please input end city :Washington, D.C.
> Your path from Seattle to Washington, D.C. is:
> Seattle(64,31) -> Minneapolis(349,100) -> Chicago(415,152) -> Washington, D.C.(536,176) ->
> Cost is: 2070

符合要求。

Test Case 2:

从 `Washington, D.C.` 到 `Seattle` 的最短路:

- Result:

> Please input mapdata filename:map-usa.txt
> Please input begin city :Washington, D.C.
> Please input end city :Seattle
> Your path from Washington, D.C. to Seattle is:
> Washington, D.C.(536,176) -> Chicago(415,152) -> Atlanta(462,273) -> Seattle(64,31) ->
> Cost is: 3050

符合要求

Test Case 3:

从 `Washington, D.C.` 到 `Los Angeles` 的最短路:

- Result:

> Please input begin city :Washington, D.C.
> Please input end city :Los Angeles
> Your path from Washington, D.C. to Los Angeles is:
> Washington, D.C.(536,176) -> Chicago(415,152) -> Atlanta(462,273) -> Dallas(310,296) -> Phoenix(138,262) -> Las Vegas(104,216)
> -> Los Angeles(58,241) ->
> Cost is: 2910

符合要求。

Chapter 4: Analysis and Comments

- 算法分析：
  算法时间长短与地图大小成正相关，与起点终点无关。
- 算法核心：

```
void dijkstra(int s,graph_data& data){
  std::fill(data.d.begin(),data.d.end(),10000000);
  std::fill(data.used.begin(),data.used.end(),false);
  std::fill(data.prev.begin(),data.prev.end(),-1);
  data.d[s] = 0;

  while (true){
    int v = -1;
    for (int u = 0;u<data.V;u++)
      if (!data.used[u] && (v==-1||data.d[u]<data.d[v])) v = u;
    if (v==-1) break;
    data.used[v] = true;

    for (int u=0;u<data.V;u++){
      if (data.d[u]>data.d[v]+data.cost[v][u]){
        data.d[u] = data.d[v] + data.cost[v][u];
        data.prev[u] = v;
      }
    }
  }
}
```

- 算法不足：

求最短路时采用宽度优先搜索，遍历了所有节点，可以使用优先队列优化，将复杂度降低到log，还可以限制搜索的宽度（A*算法）

Declaration

> **We hereby declare that all the work done in this project titled "Huffman Code" is of our independent effort as a group.**

Duty Assignments:

> **Programmer: 常家奇**
> **Tester: 常家奇**
> **Report Writer: 常家奇**

Reference

- NO REFERENCE