Dijkstra和Floyd算法笔记

1. Dijkstra算法

对于一个无向有权图（邻接矩阵存储的）：如下图所示

Dj1

邻接矩阵存储的格式：二维数组martix[size][size] ，注意每个顶点对应一个名称，这个对应关系存储在vector<int> names={A,B,C,D,E,F,G,H,I} 中，

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 索引值 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| A: 0 | 0 | 4 | 10 | ∞ | 3 | ∞ | ∞ | ∞ | ∞ |
| B: 1 | 4 | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| C: 2 | 10 | ∞ | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | 55 |
| D: 3 | ∞ | ∞ | ∞ | 0 | 1 | ∞ | ∞ | ∞ | 7 |
| E: 4 | 3 | ∞ | ∞ | 1 | 0 | 1 | ∞ | ∞ | ∞ |
| F: 5 | ∞ | ∞ | ∞ | ∞ | 1 | 0 | 1 | ∞ | ∞ |
| G: 6 | ∞ | ∞ | ∞ | ∞ | ∞ | 1 | 0 | 1 | ∞ |
| H: 7 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | 1 | 0 | 1 |
| I: 8 | ∞ | ∞ | ∞ | 55 | 7 | ∞ | ∞ | 1 | 0 |

这个算法的关键就是要计算从V0开始到最后一个顶点的位置，但是这是需要一步一步求的，每一步求出最权最小的路径，然后加一组合就可以得出最短的的路径（但是要注意随时会出现的替换的问题）

第一步： 求出从V0出发所能到的具有最小权的顶点（在这个图上看得出是V4）所以，计算得到V0->V4=3（注意我们需要把这个最近一次搜索的顶点保存，以便从这个已经最小的顶点开始进行下一次的搜索，同时也需要设置一个保存被访问的顶点的变量）

第二步：以V4为起点搜索我们V0->V4->V5=3+1=4和V0->V4->V3=3+1=4可以发现二者具有此相同的权总和。所以我们先从考虑V3的情况

第三步：以V3位起点搜索。从图中V0->V4->V3->V8=4+7=11而在第三步中V0->V4->V5->V6->V7->V8=3+1+1+1+1=7<11所以明显后者是一个总权最小的路径

算法的实现：

首先我们需要一个辅助的函数：这个函数用于输出顶点Vv的所有路径的数组（用vector存储）。例如若v=0 那么vec{∞,4,10,∞,3,∞,∞,∞,∞}（注意当V0->V0的时候保存的是0但是在搜索的时候认为是∞）

void dijkstra\_setWeightSummary(int v, vector<int> &vec)

这个是核心的函数

void do\_dijkstra(int v0,vector<int> &Patharc,vector<int> &PathWeights)

vector<int> finalPath(size);//

int v, w, k, min;//

Patharc是用来保存前驱节点的数组

PathWeight这个是保存从V0到对应节点的最小的权的和对的数组

finalPath当finalPath[w]=1的时候表示已经求出了从V0->Vw的最短路径 这是一个标志

v用于循环中表示指定的一个顶点

min表示的是从Vw起始点到权最小的顶点的权值（注意不是累加顶点和）

k便是找到当min被赋值时所对应的Vw的下标

w是表示Vw的下标的循环变量

模拟计算：

输入起始点的坐标为0

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| finalPath | **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| PathWeights | 0 | 4 | 10 | ∞ | 3 | ∞ | ∞ | ∞ | ∞ |
| Patharc | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

开始搜索

设置min=∞

开始从V1搜索从V0->V1的最小的权的大小，最终找到的是V4，此时的min=3，k=4

循环结束我们需要设置最短的路径为finalPath[4]=1 表示我们已经设搜索了这个顶点

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| finalPath | **1** | 0 | 0 | 0 | **1** | 0 | 0 | 0 | 0 |

第二个循环开始 这个循环的目的是修正当前的最短路径以及权的总和（从Vk开始到Vw（这个是新的w）的最短路径是否比现有的路径的总权和短）

首先要判断这个顶点是否已经被搜索到（防止走回头路），很显然finalPath[0]和finalPath[4]不为0 所以他们不会被在此修正。

这是会发现从从V0->V3是没有路通的，所以他的值PathWeights[3]为无穷符号自然满足min（3）+getWeight(V4,V3)的值 =3+1=4

所以PathWeights和Patharc会被更改为

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| PathWeights | 0 | 4 | 10 | **4** | 3 | ∞ | ∞ | ∞ | ∞ |
| Patharc | 0 | 0 | 0 | **4** | 0 | 0 | 0 | 0 | 0 |

但是还要注意还有min+getWeight(V4,V5)也满足条件 所以还得改

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| PathWeights | 0 | 4 | 10 | **4** | 3 | **4** | ∞ | ∞ | ∞ |
| Patharc | 0 | 0 | 0 | **4** | 0 | **5** | 0 | 0 | 0 |

然后，，就没然后了。

第二步开始啦！

这时的v=2啦。

我们还要寻找最小的从V0的到各个点的权最小的值，我们找到了min=4，k=1（注意我们排除了已搜索项目）

所以finalPath的索引为4、0和1不会再参与第二次修正

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| finalPath | **1** | **1** | 0 | 0 | **1** | 0 | 0 | 0 | 0 |

第二个循环：开始搜索以V1为起点的更短路径 当然是什么都搜索不到…

所以第三步开始：

这是v=3

min=4，k=3

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| finalPath | **1** | **1** | 0 | **1** | **1** | 0 | 0 | 0 | 0 |

再以V3为起点搜索最短的顶点，存在一个min+getWeight(V3,V8)=4+7 < ∞(V0和V8没有联通)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| PathWeights | 0 | 4 | 10 | **4** | 3 | **4** | ∞ | ∞ | **11** |
| Patharc | 0 | 0 | 0 | **4** | 0 | **5** | 0 | 0 | **3** |

第四步

v=4。min=4，k=5

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| finalPath | **1** | **1** | 0 | **1** | **1** | **1** | 0 | 0 | 0 |

接着从V5开始搜索最短的路径 发现min+getWeight(V5,V6)=4+1<∞

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| PathWeights | 0 | 4 | 10 | **4** | 3 | **4** | **5** | ∞ | **11** |
| Patharc | 0 | 0 | 0 | **4** | 0 | **5** | **5** | 0 | **3** |

第五步：

v=5，min=5，k=6

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| finalPath | **1** | **1** | 0 | **1** | **1** | **1** | **1** | 0 | 0 |

这个依次类推，min+getWeight(V6,V7)=5+1<∞

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| PathWeights | 0 | 4 | 10 | **4** | 3 | **4** | **5** | **6** | **11** |
| Patharc | 0 | 0 | 0 | **4** | 0 | **5** | **5** | **6** | **3** |

第六步：

v=6，min=6，k=1

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| finalPath | **1** | **1** | 0 | **1** | **1** | **1** | **1** | **1** | 0 |

这个依次类推，min+getWeight(V7,V8)=6+1<11

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| PathWeights | 0 | 4 | 10 | **4** | 3 | **4** | **5** | **6** | **7** |
| Patharc | 0 | 0 | 0 | **4** | 0 | **5** | **5** | **6** | **7** |

第七步：v=7，min=7, k=8

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| finalPath | **1** | **1** | 0 | **1** | **1** | **1** | **1** | **1** | **1** |

这个依次类推，没有什么顶点可供V8访问

所以，第八步

v=8，min=10，k=2

同样的道理，也没有什么顶点供V2访问 所以

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| finalPath | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** |

第九步：退出最外层循环，函数结束

1. Floyd算法 这个算法更厉害 可以算出任意一点的与图上任意一点的最小权的路径，同时也可以精确地算出路径。这个也挺厉害的。

首先还是那个存储的表格：

首先来个简单的

图

设这个权表为D-1

|  |  |  |  |
| --- | --- | --- | --- |
| 索引 | 0 | 1 | 2 |
| A: 0 | 0 | 1 | 2 |
| B: 1 | 1 | 0 | 10 |
| C: 2 | 2 | 10 | 0 |

* + 1. 例子3

我们可以先搜索V1->V0->V2的最小权的路径D-1[1][0]->D-1[0][2]=2+1=3而直接从V1->V2的权总和为D-1[1][2]=10

这么一来，D-1[1][0]->D-1[0][2]< D-1[1][2]

那么就有一个判断条件：D0[v][w]=min{ D-1[v][w] , D-1[v][0]+D-1[0][w] }

例子就到这里，我们来看之前的那个图

他的邻接矩阵的数据结构如下

这个表示的是第一次的权表：

D-1

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 索引值 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| A: 0 | 0 | 4 | 10 | ∞ | 3 | ∞ | ∞ | ∞ | ∞ |
| B: 1 | 4 | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| C: 2 | 10 | ∞ | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | 55 |
| D: 3 | ∞ | ∞ | ∞ | 0 | 1 | ∞ | ∞ | ∞ | 7 |
| E: 4 | 3 | ∞ | ∞ | 1 | 0 | 1 | ∞ | ∞ | ∞ |
| F: 5 | ∞ | ∞ | ∞ | ∞ | 1 | 0 | 1 | ∞ | ∞ |
| G: 6 | ∞ | ∞ | ∞ | ∞ | ∞ | 1 | 0 | 1 | ∞ |
| H: 7 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | 1 | 0 | 1 |
| I: 8 | ∞ | ∞ | ∞ | 55 | 7 | ∞ | ∞ | 1 | 0 |

P-1表示的是用来存储路径 首先初始化一下这样：

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 索引值 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| A: 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| B: 1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| C: 2 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| D: 3 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| E: 4 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| F: 5 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| G: 6 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| H: 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| I: 8 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

在我的例子之中没有使用动态分配的二维数组，而是用了是vector<vector<int>> 存储，其实也没什么，主要是方便。

D-1[k][w]表示的是从Vk->Vw的权，D-1[v][w]表示的是从Vv->Vw的权。对于无向图，权D-1[w][v]= D-1[v][w]表示的就是Vw->Vv所以他们的权之和为Vk->Vv的权然后与Vk->Vv(直接)的权值之和比较。如果小的话，更新D-1[v][w]。注意我们讨论的是无向图，但是对于有向图的话，D-1[v][w]为∞（不会有那两个均不为∞能大于∞），这个∞总是会被替换的（其他路径），但是D-1[w][v]可能是一个别的值。仅仅是方向的区别。别忘了我们需要更新P数组的值。

2.1.2 模拟计算

第一步：k=0

D:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 索引值 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| A: 0 | 0 | 4 | 10 | ∞ | 3 | ∞ | ∞ | ∞ | ∞ |
| B: 1 | 4 | 0 | **∞** | ∞ | **∞** | ∞ | ∞ | ∞ | ∞ |
| C: 2 | 10 | **∞** | 0 | ∞ | **∞** | ∞ | ∞ | ∞ | 55 |
| D: 3 | ∞ | ∞ | ∞ | 0 | **1** | ∞ | ∞ | ∞ | 7 |
| E: 4 | 3 | **∞** | **∞** | 1 | 0 | 1 | ∞ | ∞ | ∞ |
| F: 5 | ∞ | ∞ | ∞ | ∞ | 1 | 0 | 1 | ∞ | ∞ |
| G: 6 | ∞ | ∞ | ∞ | ∞ | ∞ | 1 | 0 | 1 | ∞ |
| H: 7 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | 1 | 0 | 1 |
| I: 8 | ∞ | ∞ | ∞ | 55 | 7 | ∞ | ∞ | 1 | 0 |

D[1][2]=∞而D[1][0]+D[0][2]=14

D[1][4]=∞而D[1][0]+D[0][3]=4+3=7

D[2][1]=∞而D[2][0]+D[0][1]=14

D[2][4]=∞而D[2][0]+D[0][4]=13

D[4][1]=∞而D[4][0]+D[0][1]=7

D[4][2]=∞而D[4][0]+D[0][2]=13

同时上述被修改的位置所对应的P[v][w]要被改为P[v][k]

P

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 索引值 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| A: 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| B: 1 | 0 | 1 | **1** | 3 | **4** | 5 | 6 | 7 | 8 |
| C: 2 | 0 | **1** | 2 | 3 | **4** | 5 | 6 | 7 | 8 |
| D: 3 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| E: 4 | 0 | **1** | **2** | 3 | 4 | 5 | 6 | 7 | 8 |
| F: 5 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| G: 6 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| H: 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| I: 8 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

P[1][2]=P[1][0]=0

P[1][4]=P[1][0]=0

P[2][1]=0

P[2][4]=0

P4][1]=0

P[4][2]=0

循环停止：

从k=1到k=8的关系可以此类推