

最短路径问题---Dijkstra算法详解

原创 Ouyang_Lianjun 最后发布于2017-03-08 16:42:46 阅读数 348612 ☆ 收藏

前言
Nobody can go back and start a new beginning,but anyone can start today and make a new ending.
Name:Willam
Time:2017/3/8

1、最短路径问题介绍

问题解释：
从图中的某个顶点出发到达另外一个顶点的所经过的边的权重和最小的一条路径，称为最短路径

解决问题的算法：

- 迪杰斯特拉算法（Dijkstra算法）
- 弗洛伊德算法（Floyd算法）
- SPFA算法

这篇博客，我们就对Dijkstra算法来做一个详细的介绍

2、Dijkstra算法介绍

- 算法特点：

迪科斯彻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的一个子模块。
- 算法的思路

Dijkstra算法采用的是一种贪心的策略，声明一个数组dis来保存源点到各个顶点的最短距离和一个保存已经找到了最短路径的顶点的集合：T，初始时的路径权重被赋为 0（dis[s] = 0）。若对于顶点 s 存在能直接到达的边（s,m），则把dis[m]设为w（s, m）,同时把所有其他（s不能直接到达的）顶点长度设为无穷大。初始时，集合T只有顶点s。
然后，从dis数组选择最小值，则该值就是源点s到该值对应的顶点的最短路径，并且把该点加入到T中，OK，此时完成一个顶点，
然后，我们需要看看新加入的顶点是否可以到达其他顶点并且看看通过该顶点到达其他点的路径长度是否比源点直接到达短，如果是，那么就替换这些dis中的值。
然后，又从dis中找出最小值，重复上述动作，直到T中包含了图的所有顶点。

3、Dijkstra算法示例演示

下面我求下图，从顶点v1到其他各个顶点的最短路径

👍
452

🔗

💬
84

📖

☆

📱

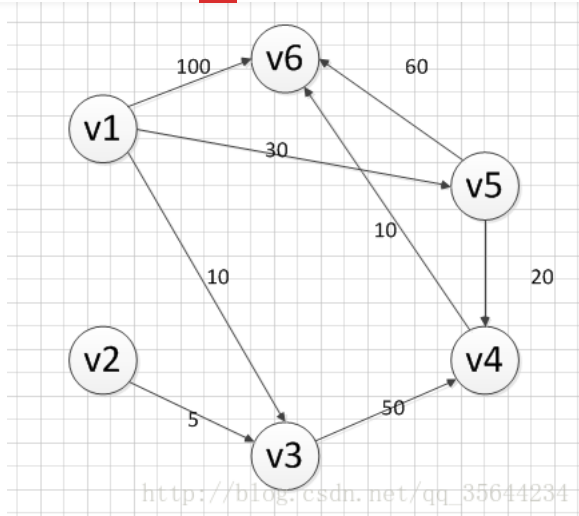
<

>

👍

🔊

举报



452

84

首先第一步，我们先声明一个dis数组，该数组初始化的值为：

Dis	0	∞	10	∞	30	100
-----	---	----------	----	----------	----	-----

我们的顶点集T的初始化为： $T=\{v1\}$

既然是求 $v1$ 顶点到其余各个顶点的最短路程，那就先找一个离 $v1$ 号顶点最近的顶点。通过数组 dis 可知当前离 $v1$ 顶点最近是 $v3$ 顶点。当选择了 2 号顶点后（下标从0开始）的值就已经从“估计值”变为了“确定值”，即 $v1$ 顶点到 $v3$ 顶点的最短路程就是当前 $dis[2]$ 值。将 $v3$ 加入到 T 中。

为什么呢？因为目前离 $v1$ 顶点最近的是 $v3$ 顶点，并且这个图所有的边都是正数，那么肯定不可能通过第三个顶点中转，使得 $v1$ 顶点到 $v3$ 顶点的路程进一步了。因为 $v1$ 顶点到其它顶点的路程肯定没有 $v1$ 到 $v3$ 点短。

OK，既然确定了一个顶点的最短路径，下面我们就要根据这个新入的顶点 $v3$ 会有出度，发现以 $v3$ 为弧尾的有： $\langle v3, v4 \rangle$ ，那么我们看看路径： $v1-v3-v4$ 是否比 $v1-v4$ 短，其实这个已经是很明显的了，因为 $dis[3]$ 代表的就是 $v1-v4$ 的长度为无穷大，而 $v1-v3-v4$ 的长度为： $10+50=60$ ，所以更新 $dis[3]$ 的值，得到效果：

Dis	0	∞	10	60	30	100
-----	---	----------	----	----	----	-----

因此 $dis[3]$ 要更新为 60 。这个过程有个专业术语叫做“松弛”。即 $v1$ 顶点到 $v4$ 顶点的路程即 $dis[3]$ ，通过 $\langle v3, v4 \rangle$ 这条边松弛成功。这便是 Dijkstra 重要思想：通过“边”来松弛 $v1$ 顶点到其余各个顶点的路程。

然后，我们又从除 $dis[2]$ 和 $dis[0]$ 外的其他值中寻找最小值，发现 $dis[4]$ 的值最小，通过之前是解释的原理，可以知道 $v1$ 到 $v5$ 的最短距离就是 $dis[4]$ 的值，然后把 $v5$ 加入到集合 T 中，然后，考虑 $v5$ 的出度是否会影响我们的数组 dis 的值， $v5$ 有两条出度： $\langle v5, v4 \rangle$ 和 $\langle v5, v6 \rangle$ ，然后我们发现： $v1-v5-v4$ 的长度为： 50 ， $dis[3]$ 的值为 60 ，所以我们要更新 $dis[3]$ 的值。另外， $v1-v5-v6$ 的长度为： 90 ，而 $dis[5]$ 为 100 ，所以我们需要更新 $dis[5]$ 的值。更新后的 dis 数组如下图：

Dis	0	∞	10	50	30	90
-----	---	----------	----	----	----	----

举报

然后，继续从 dis 中选择未确定的顶点的值中选择一个最小的值，发现 $dis[3]$ 的值是最小的，所以把 $v4$ 加入到集合 T 中，此时集合 $T=\{v1, v3, v5, v4\}$ ，然后，考虑度是否会影响我们的数组 dis 的值， $v4$ 有一条出度： $\langle v4, v6 \rangle$ ，然后我们发现： $v1-v5-v4-v6$ 的长度为： 60 ，而 $dis[5]$ 的值为 90 ，所以我们要更新 $dis[5]$ 的值

Dis	0	∞	10	50	30	60
-----	---	----------	----	----	----	----

然后，我们使用同样原理，分别确定了v6和v2的最短路径，最后dis的数组的值如下：

Dis	0	∞	10	50	30	60
-----	---	----------	----	----	----	----

因此，从图中，我们可以发现v1-v2的值为： ∞ ，代表没有路径从v1到达v2。所以我们得到的最后的结果为：

1	起点	终点	最短路径	长度
2	v1	v2	无	∞
3		v3	{v1,v3}	10
4		v4	{v1,v5,v4}	50
5		v5	{v1,v5}	30
6		v6	{v1, v5,v4,v6}	60
7				

4、Dijkstra算法的代码实现 (c++)

• Dijkstra.h文件的代码

```
1  /*****  
2  /*      程序作者：Willam      */  
3  /*      程序完成时间：2017/3/8      */  
4  /*      有任何问题请联系：2930526477@qq.com      */  
5  /*****  
6  // @尽量写出完美的程序  
7  
8  #pragma once  
9  // #pragma once是一个比较常用的C/C++杂注，  
10 // 只要在头文件的最开始加入这条杂注，  
11 // 就能够保证头文件只被编译一次。  
12  
13 #include<iostream>  
14 #include<string>  
15 using namespace std;  
16  
17 /*  
18 本程序是使用Dijkstra算法实现求解最短路径的问题  
19 采用的邻接矩阵来存储图  
20 */  
21 // 记录起点到每个顶点的最短路径的信息  
22 struct Dis {  
23     string path;  
24     int value;  
25     bool visit;  
26     Dis() {  
27         visit = false;  
28         value = 0;  
29         path = "";  
30     }  
31 };  
32 };  
33  
34 class Graph_DG {
```



452



84



```
37     int edge;           //图的边数
38     int **arc;          //邻接矩阵
39     Dis * dis;          //记录各个顶点最短路径的信息
40 public:
41     //构造函数
42     Graph_DG(int vexnum, int edge);
43     //析构函数
44     ~Graph_DG();
45     //判断我们每次输入的的边的信息是否合法
46     //顶点从1开始编号
47     bool check_edge_value(int start, int end, int weight);
48     //创建图
49     void createGraph();
50     //打印邻接矩阵
51     void print();
52     //求最短路径
53     void Dijkstra(int begin);
54     //打印最短路径
55     void print_path(int);
56 };
```



452



84



• Dijkstra.cpp文件的代码

```
1  #include"Dijkstra.h"
2
3  //构造函数
4  Graph_DG::Graph_DG(int vexnum, int edge) {
5      //初始化顶点数和边数
6      this->vexnum = vexnum;
7      this->edge = edge;
8      //为邻接矩阵开辟空间和赋初值
9      arc = new int*[this->vexnum];
10     dis = new Dis[this->vexnum];
11     for (int i = 0; i < this->vexnum; i++) {
12         arc[i] = new int[this->vexnum];
13         for (int k = 0; k < this->vexnum; k++) {
14             //邻接矩阵初始化为无穷大
15             arc[i][k] = INT_MAX;
16         }
17     }
18 }
19 //析构函数
20 Graph_DG::~Graph_DG() {
21     delete[] dis;
22     for (int i = 0; i < this->vexnum; i++) {
23         delete this->arc[i];
24     }
25     delete arc;
26 }
27
28 //判断我们每次输入的的边的信息是否合法
29 //顶点从1开始编号
30 bool Graph_DG::check_edge_value(int start, int end, int weight) {
31     if (start<1 || end<1 || start>vexnum || end>vexnum || weight < 0) {
32         return false;
33     }
34     return true;
35 }
36
37 void Graph_DG::createGraph() {
38     cout << "请输入每条边的起点和终点（顶点编号从1开始）以及其权重" << endl;
```



举报

```
42     int weight;
43     int count = 0;
44     while (count != this->vexnum) {
45         cin >> start >> end >> weight;
46         //首先判断边的信息是否合法
47         while (!this->check_edge_value(start, end, weight)) {
48             cout << "输入的边的信息不合法, 请重新输入" << endl;
49             cin >> start >> end >> weight;
50         }
51         //对邻接矩阵对应上的点赋值
52         arc[start - 1][end - 1] = weight;
53         //无向图添加上这行代码
54         //arc[end - 1][start - 1] = weight;
55         ++count;
56     }
57 }
58 }
59
60 void Graph_DG::print() {
61     cout << "图的邻接矩阵为:" << endl;
62     int count_row = 0; //打印行的标签
63     int count_col = 0; //打印列的标签
64     //开始打印
65     while (count_row != this->vexnum) {
66         count_col = 0;
67         while (count_col != this->vexnum) {
68             if (arc[count_row][count_col] == INT_MAX)
69                 cout << "∞" << " ";
70             else
71                 cout << arc[count_row][count_col] << " ";
72             ++count_col;
73         }
74         cout << endl;
75         ++count_row;
76     }
77 }
78
79 void Graph_DG::Dijkstra(int begin){
80     //首先初始化我们的dis数组
81     int i;
82     for (i = 0; i < this->vexnum; i++) {
83         //设置当前的路径
84         dis[i].path = "v" + to_string(begin) + "-->v" + to_string(i + 1);
85         dis[i].value = arc[begin - 1][i];
86     }
87     //设置起点的到起点的路径为0
88     dis[begin - 1].value = 0;
89     dis[begin - 1].visit = true;
90
91     int count = 1;
92     //计算剩余的顶点的最短路径 ( 剩余this->vexnum-1个顶点 )
93     while (count != this->vexnum) {
94         //temp用于保存当前dis数组中最小的那个下标
95         //min记录的当前的最小值
96         int temp=0;
97         int min = INT_MAX;
98         for (i = 0; i < this->vexnum; i++) {
99             if (!dis[i].visit && dis[i].value<min) {
100                 min = dis[i].value;
101                 temp = i;
102             }
103         }
104         //cout << temp + 1 << " " << min << endl;
105         //把temp对应的顶点加入到已经找到的最短路径的集合中
```

452

84

84

84

84

84

84

84

84

84

84

84

84

84

84

84

84

84

84

84

84

84

84

84

84

84

84

84

84

84

```
108     for (i = 0; i < this->vexnum; i++) {
109         //注意这里的条件arc[temp][i]!=INT_MAX必须加, 不然会出现溢出, 从而造成程序异常
110         if (!dis[i].visit && arc[temp][i]!=INT_MAX && (dis[temp].value + arc[temp][i]) < dis[i].value) {
111             //如果新得到的边可以影响其他为访问的顶点, 那就就更新它的最短路径和长度
112             dis[i].value = dis[temp].value + arc[temp][i];
113             dis[i].path = dis[temp].path + "-->v" + to_string(i + 1);
114         }
115     }
116 }
117
118 }
119 void Graph_DG::print_path(int begin) {
120     string str;
121     str = "v" + to_string(begin);
122     cout << "以"<<str<<"为起点的图的最短路径为:" << endl;
123     for (int i = 0; i != this->vexnum; i++) {
124         if(dis[i].value!=INT_MAX)
125             cout << dis[i].path << "=" << dis[i].value << endl;
126         else {
127             cout << dis[i].path << "是无最短路径的" << endl;
128         }
129     }
130 }
```

452

84

84

452

452

452

452

452

452

- main.cpp文件的代码

```
1  #include"Dijkstra.h"
2
3
4  //检验输入边数和顶点数的值是否有效, 可以自己推算为啥:
5  //顶点数和边数的关系是: ((Vexnum*(Vexnum - 1)) / 2) < edge
6  bool check(int Vexnum, int edge) {
7      if (Vexnum <= 0 || edge <= 0 || ((Vexnum*(Vexnum - 1)) / 2) < edge)
8          return false;
9      return true;
10 }
11 int main() {
12     int vexnum; int edge;
13
14     cout << "输入图的顶点个数和边的条数:" << endl;
15     cin >> vexnum >> edge;
16     while (!check(vexnum, edge)) {
17         cout << "输入的数值不合法, 请重新输入" << endl;
18         cin >> vexnum >> edge;
19     }
20     Graph_DG graph(vexnum, edge);
21     graph.createGraph();
22     graph.print();
23     graph.Dijkstra(1);
24     graph.print_path(1);
25     system("pause");
26     return 0;
27 }
```

输入:

```
1 6 8
2 1 3 10
3 1 5 30
4 1 6 100
5 2 3 5
```



举报

8 5 6 60

9 5 4 20

452

输出：

```
E:\vs2015 coding\最短路径 (Dijkstra算法) \Debug\最短路径 (Dijkstra算法) .exe
输入图的顶点个数和边的条数:
6 8
请输入每条边的起点和终点（顶点编号从1开始）以及其权重
1 3 10
1 5 30
1 6 100
2 3 5
3 4 50
4 6 10
5 6 60
5 4 20
图的邻接矩阵为:
00 00 10 00 30 100
00 00 5 00 00 00
00 00 00 50 00 00
00 00 00 00 00 10
00 00 00 20 00 60
00 00 00 00 00 00
以v1为起点的图的最短路径为:
v1-->v1=0
v1-->v2是无最短路径的
v1-->v3=10
v1-->v5-->v4=50
v1-->v5=30
v1-->v5-->v4-->v6=60
请按任意键继续. . . http://blog.csdn.net/qq_35644234
```

452

84

赏

从输出可以看出，程序的结果和我们之前手动计算的结果是一样的。

点赞 452 收藏 分享 ...



Ouyang_Lianjun

发布了70 篇原创文章 · 获赞 1131 · 访问量 104万+

私信

想对作者说点什么

fgg13 2年前

“这个图所有的边都是正数，那么肯定不可能通过第三个顶点中转，使得 v1顶点到 v3顶点的路程进一步缩短了。” 这句话太关键了！我就一直没想到这个事实。

查看回复

炸腿 1年前

请问楼主，Dijkstra 能在多项式时间内求到全局最优吗？还是只能得到局部最优？

查看回复

wendy_zhao119 3周前

简单易懂，感谢！

登录 查看 83 条热评

简单易懂——Dijkstra算法讲解

阅读数 7万+

我的机器学习教程「美团」算法工程师带你入门机器学习 已经开始更新了，欢迎大家订阅~任何关于算... 博文 来自： Machine Lear...

数据结构--Dijkstra算法最清楚的讲解

阅读数 7万+

迪杰斯特拉(Dijkstra)算法是典型最短路径算法，用于计算一个节点到其他节点的最短路径。它的主要特... 博文 来自： heroacool的专...

最短路径dijkstra算法精品代码（超详解）

阅读数 1万+

一：简介 这个算法用于解决图中单源最短路径问题。所谓单源节点是指给定源节点，求图中其它节... 博文 来自： KING_阿飞的...