

导航

博客园
首 页
新随笔
联 系
订 阅
管 理

< 2012年7月 >						
日	一	二	三	四	五	六
24	25	26	27	28	29	30
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4

公告

昵称: as_
园龄: 2年9个月
粉丝: 188
关注: 0
+加关注

搜索

找找看

谷歌搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签
我的标签

笔记(4)
OS(2)
SMO(1)
socket(1)
static(1)
SVM(1)
Trie(1)
volatile(1)
bytes(1)
C str(1)
更多

随笔分类

最短路径—Dijkstra算法和Floyd算法

Dijkstra算法

1.定义概览

Dijkstra(迪杰斯特拉)算法是典型的单源最短路径算法，用于计算一个节点到其他所有节点的最短路径。主要特点是**以起始点为中心向外层层扩展，直到扩展到终点为止**。Dijkstra算法是很有代表性的最短路径算法，在很多专业课程中都作为基本内容有详细的介绍，如数据结构，图论，运筹学等等。注意该算法要求图中不存在负权边。

问题描述：在无向图 $G=(V,E)$ 中，假设每条边 $E[i]$ 的长度为 $w[i]$ ，找到由顶点 V_0 到其余各点的最短路径。（单源最短路径）

2.算法描述

1)算法思想：设 $G=(V,E)$ 是一个带权有向图，把图中顶点集合 V 分成两组，**第一组为已求出最短路径的顶点集合**（用 S 表示，初始时 S 中只有一个源点，以后每求得一条最短路径，就将加入到集合 S 中，直到全部顶点都加入到 S 中，算法就结束了），**第二组为其余未确定最短路径的顶点集合（用 U 表示）**，按最短路径长度的递增次序依次把第二组的顶点加入 S 中。在加入的过程中，总保持从**源点 v 到 S 中各顶点的最短路径长度不大于从源点 v 到 U 中任何顶点的最短路径长度**。此外，每个顶点对应一个距离， S 中的顶点的距离就是从 v 到此顶点的最短路径长度， U 中的顶点的距离，是从 v 到此顶点只包括 S 中的顶点为中间顶点的当前最短路径长度。

2)算法步骤:

- a.初始时， **S 只包含源点**，即 $S=\{v\}$ ， v 的距离为0。 U 包含除 v 外的其他顶点，即: $U=\{\text{其余顶点}\}$ ，若 v 与 U 中顶点 u 有边，则 $\langle u,v \rangle$ 正常有权值，若 u 不是 v 的出边邻接点，则 $\langle u,v \rangle$ 权值为 ∞ 。
- b.从 U 中选取一个距离 v 最小的顶点 k ，把 k ，加入 S 中（该选定的距离就是 v 到 k 的最短路径长度）。
- c.以 k 为新考虑的中间点，修改 U 中各顶点的距离；若从源

APUE专题(15)
C/C++(30)
Hadoop/MapReduce(13)
Java(1)
OS/Linux(15)
笔试面试题集锦(16)
基础机器学习算法(9)
其他(3)
数据结构与算法(29)
网络及UNP(18)

随笔档案

2015年3月 (1)
2014年11月 (1)
2013年5月 (1)
2012年11月 (4)
2012年10月 (7)
2012年9月 (17)
2012年8月 (59)
2012年7月 (59)

最新评论

1. Re:new/delete 和
malloc/free 的区别一般汇
总
@dama323什么意思? 没有
看懂。。...

--不系之舟530

2. Re:最短路径—Dijkstra算
法和Floyd算法
算法原理那里, 2 (是从i经过
若干个节点k到j) 好像不等
价于 $\text{Dis}(i,k) + \text{Dis}(k,j) < \text{Dis}(i,j)$ 吧?

--yeshen

3. Re:C++ STL中的vector
的内存分配与释放
@光速小子最后不是有四个
对象存在? vector 中2个 外
面原始对象2个...

--as_

4. Re:C++ STL中的vector
的内存分配与释放
为什么之后destruction了四
次? 能解释一下吗?

--光速小子

5. Re:O(n)回文子串
(Manacher) 算法
@as_恩, 这个问题已经弄明
白了, 可能当时你没注意这
点, , p其实动态申请
str.size()大小就可以了, 因
为str已经是加入\$#号之后的
字符串了。...

--zack Lu

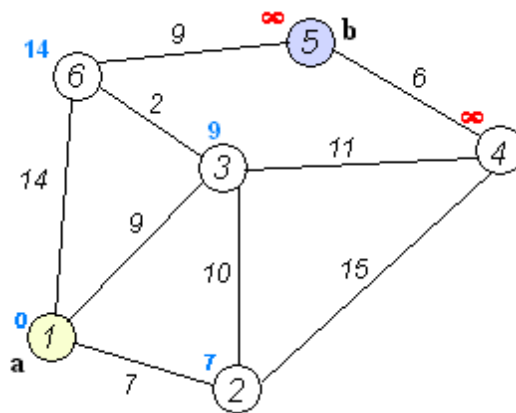
阅读排行榜

1. 最短路径—Dijkstra算法
和Floyd算法(35051)
2. 决策树算法总结(30237)

点v到顶点u的距离(经过顶点k)比原来距离(不经过顶点k)短, 则修改顶点u的距离值, 修改后的距离值的顶点k的距离加上边上的权。

d.重复步骤b和c直到所有顶点都包含在S中。

执行动画过程如下图



3.算法代码实现:



```
const int MAXINT = 32767;
const int MAXNUM = 10;
int dist[MAXNUM];
int prev[MAXNUM];

int A[MAXNUM][MAXNUM];

void Dijkstra(int v0)
{
    bool S[MAXNUM];
    // 判断是否已存入该点到S集合中
    int n=MAXNUM;
    for(int i=1; i<=n; ++i)
    {
        dist[i] = A[v0][i];
        S[i] = false;
        // 初始都未用过该点
        if(dist[i] == MAXINT)
            prev[i] = -1;
        else
            prev[i] = v0;
    }
    dist[v0] = 0;
    S[v0] = true;
```

3. Logistic Regression--
逻辑回归算法汇总**
(22507)

4. HTTP请求报文和HTTP响
应报文(22181)

5. 排序算法汇总总结
(15014)

评论排行榜

1. 最短路径—Dijkstra算法
和Floyd算法(10)

2. C++ STL中的vector的
内存分配与释放(7)

3. 排序算法汇总总结(5)

4. C/C++中static关键字作
用总结(4)

5. HTTP请求报文和HTTP响
应报文(3)

推荐排行榜

1. 最短路径—Dijkstra算法
和Floyd算法(19)

2. 信号量、互斥体和自旋锁
(7)

3. C/C++中static关键字作
用总结(6)

4. Logistic Regression--
逻辑回归算法汇总**(6)

5. HTTP请求报文和HTTP响
应报文(6)

```
for(int i=2; i<=n; i++)
{
    int mindist = MAXINT;
    int u = v0;

    // 找出当前未使用的点j的dist[j]最小值
    for(int j=1; j<=n; ++j)
        if((!S[j]) && dist[j]<mindist)
        {
            u = j;

            // u保存当前邻接点中距离最小的点的号码
            mindist = dist[j];
        }
    S[u] = true;
    for(int j=1; j<=n; j++)
        if((!S[j]) && A[u][j]<MAXINT)
        {
            if(dist[u] + A[u][j] < dist[j])
            //在通过新加入的u点路径找到离v0点更短的路径
            {
                dist[j] = dist[u] + A[u][j];

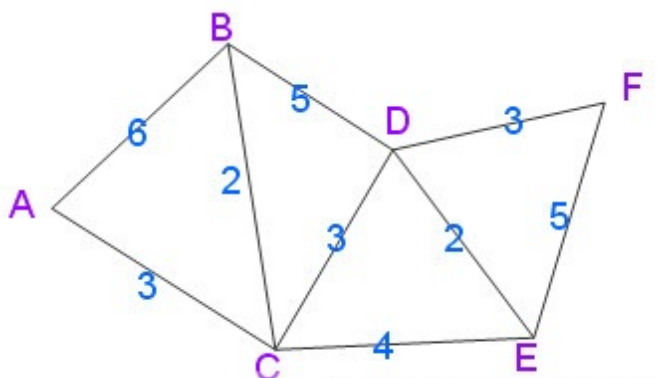
                //更新dist
                prev[j] = u;

                //记录前驱顶点
            }
        }
    }
}
```



4.算法实例

先给出一个无向图



用Dijkstra算法找出以A为起点的单源最短路径步骤如下

步骤	S集合中	U集合中
1	选入 A, 此时 S= <A> 此时最短路径 A→A=0 以 A 为中间点, 从 A 开始找	U= <B、C、D、E、F> A→B=6 A→C=3 A→其他 U 中的顶点= ∞ 发现 A→C=3 权值为最短
2	选入 C, 此时 S= <A、C> 此时最短路径 A→A=0, A→C=3 以 C 为中间点, 从 A→C=3 这条最短路径开始找	U= <B、D、E、F> A→C→B=5 (比上面第一步的 A→B=6 要短) 此时到 B 权值为 A→C→B=5 A→C→D=6 A→C→E=7 A→C→其他 U 中的顶点= ∞ 发现 A→C→B=5 权值为最短
3	选入 B, 此时 S= <A、C、B> 此时最短路径 A→A=0, A→C=3, A→C→B=5 以 B 为中间点, 从 A→C→B=5 这条最短路径开始找	U= <D、E、F> A→C→B→D=10 (比上面第二步的 A→C→D=6 要长) 此时到 D 权值更改为 A→C→D=6 A→C→B→其他 U 中的顶点= ∞ 发现 A→C→D=6 权值为最短
4	选入 D, 此时 S= <A、C、B、D> 此时最短路径 A→A=0, A→C=3, A→C→B=5, A→C→D=6 以 D 为中间点, 从 A→C→D=6 这条最短路径开始找	U= <E、F> A→C→D→E=8 (比上面第二步的 A→C→E=7 要长) 此时到 E 权值更改为 A→C→E=7 A→C→D→F=9 发现 A→C→E=7 权值为最短
5	选入 E, 此时 S= <A、C、B、D、E> 此时最短路径 A→A=0, A→C=3, A→C→B=5, A→C→D=6, A→C→E=7 以 E 为中间点, 从 A→C→E=7 这条最短路径开始找	U= <F> A→C→E→F=12 (比上面第四步的 A→C→D→F=9 要长) 此时到 F 权值更改为 A→C→D→F=9 发现 A→C→D→F=9 权值为最短
6	选入 F, 此时 S= <A、C、B、D、E、F> 此时最短路径 A→A=0, A→C=3, A→C→B=5, A→C→D=6, A→C→E=7, A→C→D→F=9	U 集合已空, 查找完毕。

Floyd算法

1.定义概览

Floyd-Warshall算法（Floyd-Warshall algorithm）是解决任意两点间的最短路径的一种算法，可以正确处理有向图或负权的最短路径问题，同时也被用于计算有向图的传递闭包。
Floyd-Warshall算法的时间复杂度为 $O(N^3)$ ，空间复杂度为 $O(N^2)$ 。

2.算法描述

1)算法思想原理：

Floyd算法是一个经典的动态规划算法。用通俗的语言来描述的话，首先我们的目标是寻找从点*i*到点*j*的最短路径。从动态规划的角度看问题，我们需要为这个目标重新做一个诠释（这个诠释正是动态规划最富创造力的精华所在）

从任意节点*i*到任意节点*j*的最短路径不外乎2种可能，1是直接从*i*到*j*，2是从*i*经过若干个节点*k*到*j*。所以，我们假设Dis(*i,j*)为节点*u*到节点*v*的最短路径的距离，对于每一个节点*k*，我们检查Dis(*i,k*) + Dis(*k,j*) < Dis(*i,j*)是否成立，如果成立，证明从*i*到*k*再到*j*的路径比*i*直接到*j*的路径短，我们便设置Dis(*i,j*) = Dis(*i,k*) + Dis(*k,j*)，这样一来，当我们遍历完所有节点*k*，Dis(*i,j*)中记录的便是*i*到*j*的最短路径的距离。

2). 算法描述:

a. 从任意一条单边路径开始。所有两点之间的距离是边的权，如果两点之间没有边相连，则权为无穷大。

b. 对于每一对顶点 u 和 v ，看看是否存在一个顶点 w 使得从 u 到 w 再到 v 比已知的路径更短。如果是更新它。

3). Floyd算法过程矩阵的计算----十字交叉法

方法：两条线，从左上角开始计算一直到右下角 如下所示

给出矩阵，其中矩阵 A 是邻接矩阵，而矩阵 $Path$ 记录 u, v 两点之间最短路径所必须经过的点

$$A_{-1} = \begin{bmatrix} 0 & 5 & \infty & 7 \\ \infty & 0 & 4 & 2 \\ 3 & 3 & 0 & 2 \\ \infty & \infty & 1 & 0 \end{bmatrix} \quad Path_{-1} = \begin{bmatrix} -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \end{bmatrix}$$

相应计算方法如下：

(1) 把 A_{-1} 划去第 0 行第 0 列和对角线来计算 A_0

不在三条线上的元素所在的 2 阶矩阵为：

$$\begin{bmatrix} 0 & \infty \\ \infty & 4 \end{bmatrix}, \begin{bmatrix} 0 & 7 \\ \infty & 2 \end{bmatrix}, \begin{bmatrix} 0 & 5 \\ 3 & 3 \end{bmatrix}, \begin{bmatrix} 0 & 7 \\ 3 & 2 \end{bmatrix}, \begin{bmatrix} 0 & 5 \\ \infty & \infty \end{bmatrix}, \begin{bmatrix} 0 & \infty \\ \infty & 1 \end{bmatrix}$$

很容易就可以看出不在三条线上的 6 个元素都不发生改变，因此 $A_0 = A_{-1}$, $Path_0 = Path_{-1}$

(2) 把 A_0 划去第 1 行第 1 列和对角线来计算 A_1

按上面所述的判断不在三条线上的元素是否需要发生改变，发生变化的元素用括号括起来了...

$$\begin{bmatrix} 0 & 5 & \infty(9) & 7 \\ \infty & 0 & 4 & 2 \\ 3 & 3 & 0 & 2 \\ \infty & \infty & 1 & 0 \end{bmatrix} \quad \text{所以有 } A_1 = \begin{bmatrix} 0 & 5 & 9 & 7 \\ \infty & 0 & 4 & 2 \\ 3 & 3 & 0 & 2 \\ \infty & \infty & 1 & 0 \end{bmatrix} \quad Path_1 = \begin{bmatrix} -1 & -1 & 1 & -1 \\ -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \end{bmatrix}$$

(3) 把 A_1 划去第 2 行第 2 列和对角线来计算 A_2

$$\begin{bmatrix} 0 & 5 & 9 & 7 \\ \infty(7) & 0 & 4 & 2 \\ 3 & 3 & 0 & 2 \\ \infty(4) & \infty(4) & 1 & 0 \end{bmatrix} \text{ 所以有 } A_2 = \begin{bmatrix} 0 & 5 & 9 & 7 \\ 7 & 0 & 4 & 2 \\ 3 & 3 & 0 & 2 \\ 4 & 4 & 1 & 0 \end{bmatrix} \text{ Path}_2 = \begin{bmatrix} -1 & -1 & 1 & -1 \\ 2 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \\ 2 & 2 & -1 & -1 \end{bmatrix}$$

(4) 把 A_2 划去第 3 行第 3 列和对角线来计算 A_3

$$\begin{bmatrix} 0 & 5 & 9(8) & 7 \\ 7(6) & 0 & 4(3) & 2 \\ 3 & 3 & 0 & 2 \\ 4 & 4 & 1 & 0 \end{bmatrix} \text{ 所以有 } A_3 = \begin{bmatrix} 0 & 5 & 8 & 7 \\ 6 & 0 & 3 & 2 \\ 3 & 3 & 0 & 2 \\ 4 & 4 & 1 & 0 \end{bmatrix} \text{ Path}_3 = \begin{bmatrix} -1 & -1 & 3 & -1 \\ 3 & -1 & 3 & -1 \\ -1 & -1 & -1 & -1 \\ 2 & 2 & -1 & -1 \end{bmatrix}$$

最后 A_3 即为所求结果

3. 算法代码实现



```
typedef struct
{
    char vertex[VertexNum];
    //顶点表
    int edges[VertexNum][VertexNum];
    //邻接矩阵,可看做边表
    int n,e;
    //图中当前的顶点数和边数
}MGraph;

void Floyd(MGraph g)
{
    int A[MAXV][MAXV];
    int path[MAXV][MAXV];
    int i,j,k,n=g.n;
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
        {
            A[i][j]=g.edges[i][j];
            path[i][j]=-1;
        }
    for(k=0;k<n;k++)
    {
        for(i=0;i<n;i++)
            for(j=0;j<n;j++)
                if(A[i][j]>(A[i][k]+A[k][j]))
                {
                    A[i][j]=A[i][k]+A[k][j];
                    path[i][j]=k;
                }
    }
}
```



算法时间复杂度: $O(n^3)$

分类: [数据结构与算法](#)

绿色通道:

好文要顶

关注我

收藏该文

与我联系

as_
关注 - 0
粉丝 - 188
[+加关注](#)

19

0

推荐

反对

(请您对文章做出评价)

« 上一篇: [最小生成树-Prim算法和Kruskal算法](#)
» 下一篇: [贪心算法](#)

posted on 2012-07-31 12:37 as_ 阅读(35052) 评论(10) 编辑 收藏

评论

#1楼

good

支持(0) 反对(0)

2014-03-04 17:45 | ☆y急速の灵感 ★

#2楼

楼主讲得很精辟，好文章

支持(0) 反对(0)

2014-04-01 15:19 | 晓O(n_n)O~

#3楼

如此的详细

支持(0) 反对(0)

2014-08-04 20:27 | 阿鲁巴

#4楼

给赞！

支持(0) 反对(0)

2014-08-12 14:44 | [silent_coder](#) 

#5楼

强大，赞


支持(0) 反对(0)

2014-10-23 15:01 | [千叶树](#) 

#6楼

能否转载？写的好详细！！
以前学的忘了，现在看看觉得好有道理！

支持(0) 反对(0)

2014-11-05 19:07 | [Godfray](#) 

#7楼【楼主】

@Godfray
哈哈 记得标明出处吧

支持(0) 反对(0)

2014-11-17 14:25 | [as_](#) 

#8楼

结合代码总算看懂了Dijkstra算法~


支持(0) 反对(0)

2014-11-29 16:14 | [qinggeouye](#) 

#9楼

竟然可以讲的如此明白

支持(0) 反对(0)

2014-12-02 16:12 | [pppanda](#) 

#10楼

算法原理那里，2（是从i经过若干个节点k到j）好像不等价于
 $Dis(i,k) + Dis(k,j) < Dis(i,j)$ 吧？

支持(0) 反对(0)

2015-04-03 20:33 | yeshe

[刷新评论](#) [刷新页面](#) [返回顶部](#)

注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问网站首页](#)。

【推荐】50万行VC++源码：大型组态工控、电力仿真CAD与GIS源码库

融云，免费为你的App加入IM功能——让你的App“聊”起来！！



最新IT新闻：

- 华为P8国行版4月23日开卖 售价2888元起
- LG显示器沾iPhone 6的光 第一季度利润创4年新高
- 3月彩票销售同比降20.62亿 互联网渠道影响明显
- 苹果手表18K金版零售包装亮相
- 保护环境从点滴做起 世界地球日苹果绿了！
- » 更多新闻...



最新知识库文章：

- 尾调用优化
- 淘宝搜索算法现状
- 对象的职责
- 好对象的7大美德
- iOS应用架构谈（一）：架构设计的方法论
- » 更多知识库文章...

Powered by:

博客园

Copyright © as_