
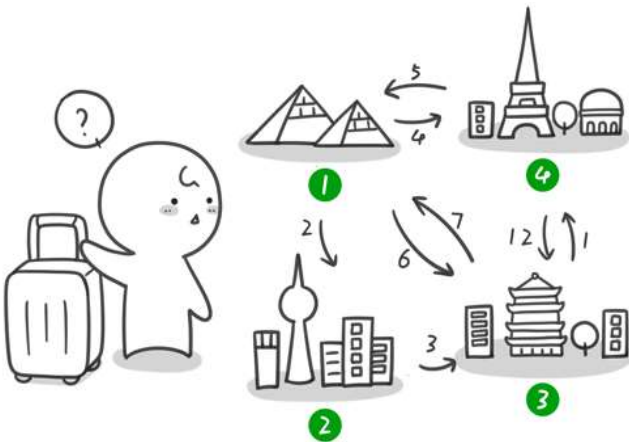


原创 推荐

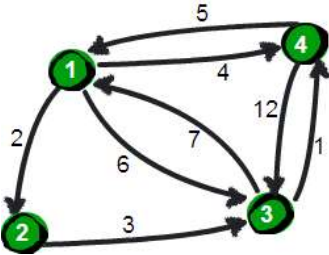
【坐在马桶上看算法】算法6：只有五行的Floyd最短路算法

 ahalei 关注

2014-03-25 09:47:22 36739人阅读 10人评论



暑假，小哼准备去一些城市旅游。有些城市之间有公路，有些城市之间则没有，如下图。为了节省经费以及方便计划旅程，小哼希望在出发之前知道任意两个城市之前的最短路程。



上图中有4个城市8条公路，公路上的数字表示这条公路的长短。请注意这些公路是单向的。我们现在需要求任意两个城市之间的最短路程，也就是求任意两个点之间的最短路径。这个问题这也被称为“多源最短路径”问题。

现在需要一个数据结构来存储图的信息，我们仍然可以用一个4*4的矩阵（二维数组e）来存储。比如1号城市到2号城市的路程为2，则设e[1][2]的值为2。2号城市无法到达4号城市，则设置e[2][4]的值为∞。另外此处约定一个城市自己是到自己的也是0，例如e[1][1]为0，具体如下。

	1	2	3	4
1	0	2	6	4
2	∞	0	3	∞
3	7	∞	0	1

现在回到问题：如何求任意两点之间最短路径呢？通过之前的学习我们知道通过深度或广度优先搜索可以求出两点之间的最短路径。所以进行n2遍深度或广度优先搜索，即对每两个点都进行一次深度或广度优先搜索，便可以求得任意两点之间的最短路径。可是还有没有别的方法呢？

我们来想一想，根据我们以往的经验，如果要让任意两点（例如从顶点a点到顶点b）之间的路程变短，只能引入第三个点（顶点k），并通过这个顶点k中转即a->k->b，才可能缩短原来从顶点a点到顶点b的路程。那么这个中转的顶点k是1~n中的哪个点呢？甚至有时候不只通过一个点，而是经过两个点或者更多点中转会更短，即a->k1->k2->b或者a->k1->k2...->k-i->b。比如上图中从4号城市到3号城市（4->3）的路程e[4][3]原本是12。如果只通过1号城市中转（4->1->3），路程将缩短为11（e[4][1]+e[1][3]=5+6=11）。其实1号城市到3号城市也可以通过2号城市中转，使得1号到3号城市的路程缩短为5（e[1][2]+e[2][3]=2+3=5）。所以如果同时经过1号和2号两个城市中转的话，从4号城市到3号城市的路程会进一步缩短为10。通过这个例子，我们发现每个顶点都有可能使得另外两个顶点之间的路程变短。好，下面我们将这个问题一般化。

当任意两点之间不允许经过第三个点时，这些城市之间最短路径就是初始路程，如下。

	1	2	3	4
1	0	2	6	4
2	∞	0	3	∞
3	7	∞	0	1
4	5	∞	12	0

如现在只允许经过1号顶点，求任意两点之间的最短路径，应该如何求呢？只需判断e[i][1]+e[1][j]是否比e[i][j]要小即可。e[i][j]表示的是从i号顶点到j号顶点之间的路程。e[i][1]+e[1][j]表示的是从i号顶点先到1号顶点，再从1号顶点到j号顶点的路程之和。其中i是1~n循环，j也是1~n循环，代码实现如下。

```
for(i=1;i<=n;i++)
{
    for(j=1;j<=n;j++)
    {
        if ( e[i][j] > e[i][1]+e[1][j] )
            e[i][j] = e[i][1]+e[1][j];
    }
}
```



在只允许经过1号顶点的情况下，任意两点之间的最短路径更新为：

	1	2	3	4
1	0	2	6	4
2	∞	0	3	∞
3	7	9	0	1
4	5	7	11	0

通过上图我们发现：在只通过1号顶点中转的情况下，3号顶点到2号顶点（e[3][2]）、4号顶点到2号顶点（e[4][2]）以及4号顶点到3号顶点（e[4][3]）的路程都变短了。

接下来继续求在只允许经过1和2号两个顶点的情况下任意两点之间的最短路径。如何做呢？我们需要在只允许经过1号顶点时任意两点的最短路径的结果下，再判断如果经过2号顶点是否可以使得i号顶点到j号顶点之间的路程变得更短。即判断e[i][2]+e[2][j]是否比e[i][j]要小，代码实现为如下。

```
//经过1号顶点
for(i=1;i<=n;i++)
    for(j=1;j<=n;j++)
        if ( e[i][j] > e[i][1]+e[1][j] ) e[i][j]=e[i][1]+e[1][j];
//经过2号顶点
for(i=1;i<=n;i++)
    for(j=1;j<=n;j++)
        if ( e[i][j] > e[i][2]+e[2][j] ) e[i][j]=e[i][2]+e[2][j];
```

在线
客服



在只允许经过1和2号顶点的情况下，任意两点之间的最短路径更新为：

	1	2	3	4
1	0	2	5	4
2	∞	0	3	∞
3	7	9	0	1
4	5	7	10	0

通过上图得知，在相比只允许通过1号顶点进行中转的情况下，这里允许通过1和2号顶点进行中转，使得 $e[1][3]$ 和 $e[4][3]$ 的路程变得更短了。

同理，继续在只允许经过1、2和3号顶点进行中转的情况下，求任意两点之间的最短路径。任意两点之间的最短路径更新为：

	1	2	3	4
1	0	2	5	4
2	10	0	3	4
3	7	9	0	1
4	5	7	10	0

最后允许通过所有顶点作为中转，任意两点之间最终的最短路径为：

	1	2	3	4
1	0	2	5	4
2	9	0	3	4
3	6	8	0	1
4	5	7	10	0

整个算法过程虽然说起来很麻烦，但是代码实现却非常简单，核心代码只有五行：

```
for(k=1;k<=n;k++)
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            if(e[i][j]>e[i][k]+e[k][j])
                e[i][j]=e[i][k]+e[k][j];
```

这段代码的基本思想就是：最开始只允许经过1号顶点进行中转，接下来只允许经过1和2号顶点进行中转.....允许经过1~n号所有顶点进行中转，求任意两点之间的最短路径。用一句话概括就是：从i号顶点到j号顶点只经过前k号点的最短路径。其实这是一种“动态规划”的思想，关于这个思想我们将在《啊哈！算法2——伟大思维闪耀时》在做详细的讨论。下面给出这个算法的完整代码：

```
#include <stdio.h>
int main()
{
    int e[10][10],k,i,j,n,m,t1,t2,t3;
    int inf=99999999; //用inf(infinity的缩写)存储一个我们认为的正无穷值
    //读入n和m，n表示顶点个数，m表示边的条数
    scanf("%d %d",&n,&m);

    //初始化
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            if(i!=j) e[i][j]=inf;
            else e[i][j]=0;
```



在线
客服



```

for(i=1;i<=m;i++)
{
    scanf("%d %d %d",&t1,&t2,&t3);
    e[t1][t2]=t3;
}

//Floyd-Warshall算法核心语句
for(k=1;k<=n;k++)
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            if(e[i][j]>e[i][k]+e[k][j] )
                e[i][j]=e[i][k]+e[k][j];

//输出最终的结果
for(i=1;i<=n;i++)
{
    for(j=1;j<=n;j++)
    {
        printf("%10d",e[i][j]);
    }
    printf("\n");
}

return 0;
}

```

有一点需要注意的是：如何表示正无穷。我们通常将正无穷定义为99999999，因为这样即使两个正无穷相加，其和仍然不超过int类型的范围（C语言int类型可以存储的最大正整数是2147483647）。在实际应用中最好估计一下最短路径的上限，只需要设置比它大一点既可以。例如有100条边，每条边不超过100的话，只需将正无穷设置为10001即可。如果你认为正无穷和其它值相加得到一个大于正无穷的数是不被允许的话，我们只需在比较的时候加两个判断条件就可以了，请注意下面代码中带有下划线的语句。



```

//Floyd-Warshall算法核心语句
for(k=1;k<=n;k++)
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            if(e[i][k]<inf && e[k][j]<inf && e[i][j]>e[i][k]+e[k][j])
                e[i][j]=e[i][k]+e[k][j];

```

上面代码的输入数据样式为：

```

4 8
1 2 2
1 3 6
1 4 4
2 3 3
3 1 7
3 4 1
4 1 5
4 3 12

```

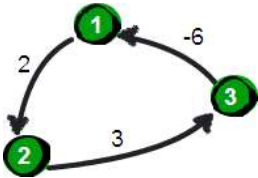
在线
客服



	1	2	3	4
1	0	2	5	4
2	9	0	3	4
3	6	8	0	1
4	5	7	10	0

通过这种方法我们可以求出任意两个点之间最短路径。它的时间复杂度是 $O(N^3)$ 。令人很震撼的是它竟然只有五行代码，实现起来非常容易。正是因为它实现起来非常容易，如果时间复杂度要求不高，使用Floyd-Warshall来求指定两点之间的最短路或者指定一个点到其余各个顶点的最短路径也是可行的。当然也有更快的算法，请看下一节：Dijkstra算法。

另外需要注意的是：Floyd-Warshall算法不能解决带有“负权回路”（或者叫“负权环”）的图，因为带有“负权回路”的图没有最短路。例如下面这个图就不存在1号顶点到3号顶点的最短路径。因为1->2->3->1->2->3->...->1->2->3这样路径中，每绕一次1->2->3这样的环，最短路就会减少1，永远找不到最短路。其实如果一个图中带有“负权回路”那么这个图则没有最短路。



此算法由Robert W. Floyd（罗伯特·弗洛伊德）于1962年发表在“Communications of the ACM”上。同年Stephen Warshall（史蒂芬·沃舍尔）也独立发表了这个算法。Robert W. Floyd这个牛人是朵奇葩，他原本在芝加哥大学读的文学，但是因为当时美国经济不太景气，找工作比较困难，无奈之下到西屋电气公司当了一名计算机操作员，在IBM650机房值夜班，并由此开始了他的计算机生涯。此外他还和J.W.J. Williams（威廉姆斯）于1964年共同发明了著名的堆排序算法HEAPSORT。堆排序算法我们将在第七章学习。Robert W. Floyd在1978年获得了图灵奖。

码字不容易啊，转载麻烦注明出处
【一周一算法】算法6：只有五行的Floyd最短路算法
<http://bbs.ahalei.com/thread-4554-1-1.html>
(出处: 啊哈磊_编程从这里起步)



©著作权归作者所有：来自51CTO博客作者ahalei的原创作品，如需转载，请注明出处，否则将追究法律责任

[超简单Floyd最短路算法](#) [超简单Floyd算法C语言实现](#) [Floyd最短路](#)

17 收藏 分享

上一篇：【坐在马桶上看算法】算法5：解密... 下一篇：【坐在马桶上看算法】算法7：Di...



ahalei
14篇文章, 109W+人气, 174粉丝

关注

在线
客服

17 2 10 分享



ahalei

关注



提问和评论都可以，用心的回复会被更多人看到和认可

Ctrl+Enter 发布

取消

发布

10条评论

按时间正序 | 按时间倒序



hustlijian
1楼 2014-03-26 12:33:55
very nice!



LEONCORE
2楼 2014-03-27 03:07:39
Cool!



wo1148
3楼 2014-03-27 10:05:33
good~~~~~



杨喜儿
4楼 2014-04-02 09:04:50
 $\sim(\geq \forall \leq) \sim$



yan127422
5楼 2014-06-19 09:19:34
由浅入深，通俗易懂



MR_BUG
6楼 2015-05-09 12:32:27
蛮好的



sniperxyz
7楼 2016-05-28 16:39:31
什么时候出算法2的书，希望加一些比如tarjan算法等



gongpeng0701
8楼 2017-12-19 18:03:52
good~



yuer1990712
9楼 2018-01-04 14:28:37



在线客服

17

2

10

分享



ahalei

关注



SMRWXH
10楼 2018-10-19 00:43:53

大佬，我看过您的《啊哈！算法》，蓝皮的那本。讲的太棒了！！！！ 请问您《啊哈！算法2思想光辉》什么时候能出版呢？？ 很期待那！！！！~~~~~

推荐专栏



基于Python的DevOps实战

自动化运维开发新概念
共20章 | 抚琴煮酒

¥ 51.00 326人订阅

订 阅



微服务技术架构和大数据治理实战

大数据时代的微服务之路
共18章 | 纯洁微笑

¥ 51.00 627人订阅

订 阅

猜你喜欢

我的友情链接

万人直播网络架构与CDN网络

Python C API的使用详解（一）

C++反射机制：可变参数模板实现C++反射

【坐在马桶上看算法】算法12：堆——神奇的优先队列...

Python C API 使用详解（二）

通过UNIX域套接字传递文件描述符

Qt高级——Qt日志信息处理



在线
客服

