西安邮电大学第十二届数学建模竞赛参赛作品

参赛队编号： 162

赛题类型代码： B

蚂蚁的最优路径分析

摘 要：针对空间给定两点，在限定条件下，本文对蚂蚁移动过程进行分析，根据图上两点间的对应关系建立模型——带权无向图模型，并讨论模型的精度和稳定性，最后又根据建立的模型改进得到了确定的路线图。

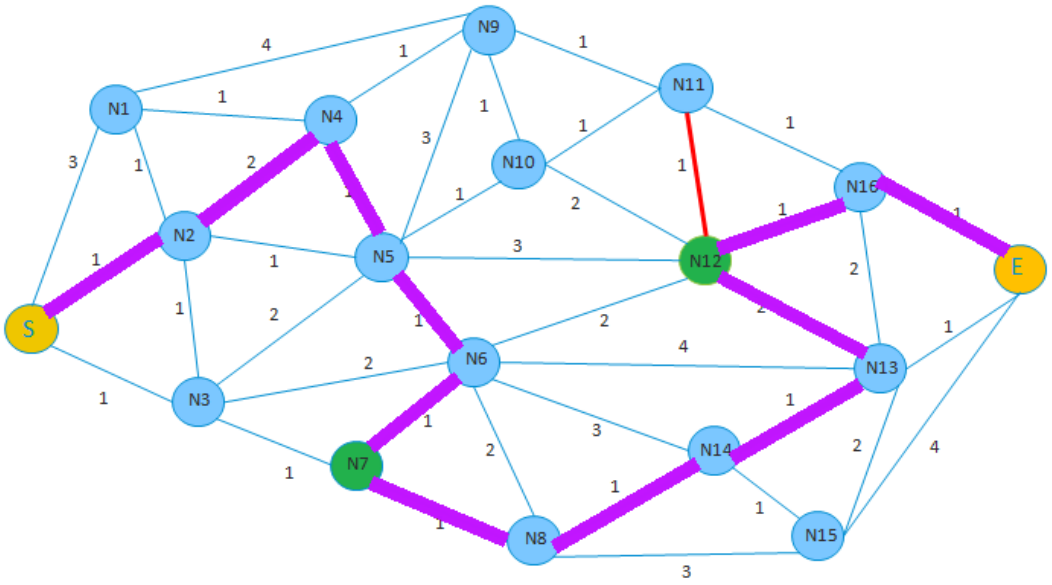
基于带权无向图数学模型，运用了 Dijkstra 算法，建立了最佳路线模型，通过C++编程，利用每个条件对结果的影响程度大小对条件进行重要性排序，使结果优先满足重要性高的条件。并逐步增加条件来修改 Dijkstra 算法，将原问题分解成几个易于用 Dijkstra 算法求解的子问题，先对各个子问题逐一求局部最优解，再在此基础上求全局最优解，得出一个满足尽量多的条件下的结果

最后针对计算结果中的误差，验证估计结果是否正确。

求解得出总路线最短、总花费最少且相对均衡的最优组路线，路线如下：

0→2→4→5→6→7→8→14→13→12→16→17

具体路线如图所示：



关键字：带权无向图模型　两点之间最短路径　条件重要性排序　编程　Dijkstra 算法　花费最少

蚂蚁的最优路径分析

一、问题重述

最强大脑中的收官蜂巢迷宫变态级挑战，相信大家都叹为观止！最强大脑收官战打响后，收视率节节攀升。在动物世界中，称得上活地图的，除了蜜蜂，蚂蚁当仁不让。在复杂多变的蚁巢中，蚂蚁总是能以最快、最高效的方式游历在各个储藏间（存储食物）。小蚁同学现需要通过蚁巢贮藏间取得玉米库的玉米，水果库的水果，还有要满足若干要求，如下：

1. 最少的花费拿到食物（图中路线上的数值表示每两个储物间的花费）；

2. 最多只能经过 9 个储藏间拿到食物（包含起止两个节点，多次通过同一节点按重复次数计算）；

3. 必须经过玉米间，水果间（附件图中标绿色节点）；

4. 食蚁兽也在路上活动（附件图中标红色路段），一旦与食蚁兽相遇，会遭遇生命危险；

5. 有两段路有蚁后准备的神秘礼物(附件图中标绿色路段)，是必须经过的。

请设计一种通用的路径搜索算法，来应对各种搜索限制条件，找到一条最优路径，顺利完成蚁后布置的任务。

注：

1、蚁巢，有若干个储藏间（图中圆圈表示），储藏间之间有诸多路可以到达；

2、节点本身通行无花费；

3、该图为无向图，可以正反两方向通行，两方向都会计费，并且花费相同；

4、起止节点分别为附件图中 S 点和 E 点。

5、最优路径：即满足限制条件的路径。

是能以最快、最高效的方式游历在各个储藏间（存储食物）。蚁巢有若干个储藏间（下图中圆圈表示），储藏间之间有诸多路可以到达(各储藏间及路径如图所示)。

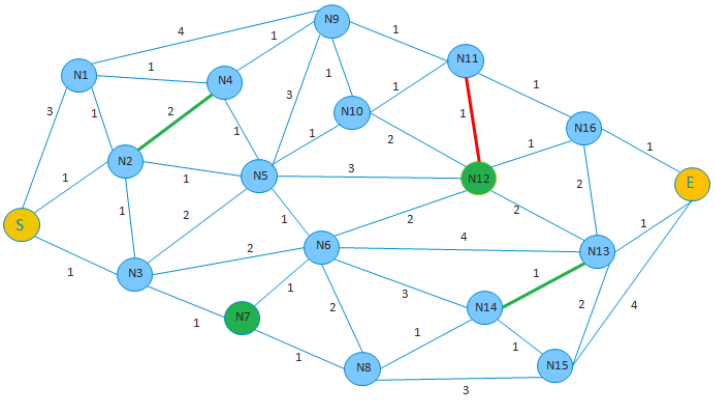


图 1.1　蚁巢中储藏间及路线图

二、模型假设与符号说明

2.1 模型假设

1. 假定不同普通储物间（非玉米间、水果间）除位置外其他条件都是相同的；

2. 假定每两个储物间之间的花费为图中两点之间边的权值；

3.假定第一个条件求最短路径设定为解决此问题的大前提；

4.根据题意，假定不同的条件对最终结果的影响程度是不同的，是可以按照对结果的影响大小（条件的重要性）对条件进行分级排列的；

5.经分析论证，本题的所有条件不可能全部实现（分析论证见下文的条件分析），假定将本题的最优路径含义理解为是最接近于结果的一条路径（即题中所说蚂蚁总是能以最快、最高效的方式游历在各个储藏间）。

2.2 符号说明

1. 将点 N1、N2...N16 设定为点 1、2...16，将 S 点和 E 点设定为点 0 和点 17，如下表所示：

符号	意义	符号	意义	符号	意义
0	点 S	6	点 N6	12	点 N12
1	点 N1	7	点 N7	13	点 N13
2	点 N2	8	点 N8	14	点 N14
3	点 N3	9	点 N9	15	点 N15
4	点 N4	10	点 N10	16	点 N16
5	点 N5	11	点 N11	17	点 E

2. I 表示条件对问题结果的影响程度，即条件的重要性。按数值大小重要性递增。如条件 2 的重要性为 I_2

3.d(j)表示 j 点到起点的距离

4.a_{jk}表示 j，k 两点之间的距离

5.INF 代表无穷大的值

6.ans 代表结果

三、问题分析

3.1 条件分析

1.以最少的花费拿到食物（附件图中路线上的数值表示每两个储物间的花费）；

***分析**：在本题的分析、建模、求解过程中，最少的花费是这个问题的核心约束条件。我们将其作为解决这个问题的大前提，不与其他 4 个条件并列分析。

2.最多只能经过 9 个储藏间拿到食物（包含起止两个节点）；

***分析**：在本题中，若满足题目要求同时过玉米仓、水果仓、两条绿色路径、起点、终点，则已经需要经过 8 个固定的点，而小蚁同学最多只能经过 9 个储藏间拿到食物（包含起止两个节点，多次通过同一节点按重复次数计算），显然不能成立。所以本条件是已经确定不能成立的了，所以将其重要性设定为 I_2=1。

3.必须经过玉米间，水果间（附件图中标绿色节点）；

***分析**：蚁后分配的任务就是要求小蚁通过蚁巢贮藏间取得玉米间的玉米、水果间的水果。显然，与蚂蚁路径的长短相比，完成任务是更重要的。所以，我们将其重要性设定为 I_3=3。

4. **食蚁兽也在路上活动，一旦与食蚁兽相遇，会遭遇生命危险；（附件图中标红**

色路段）；

***分析：**红色路段为不可控因素，如果走红色路段，虽然不确定蚂蚁一定能和食蚁兽相遇，但一旦相遇，蚂蚁就会有生命危险，这样蚂蚁任务的其他的条件全都无法实现，在确定最优计划的实际问题中，我们应该尽可能的避免不可控因素以及其所造成的影响。所以，我们将其重要性设定为 $I_4=4$。

5.有两段路是必须经过的，那里有神秘礼物(附件图中标绿色路段)。

***分析：**在实现小蚁以最少花费拿到玉米和水果的过程中，神秘礼物对该目的的实现造成的影响不可预测，应尽量降低它对解决问题影响程度，所以将其重要性设定为 $I_5=2$

3.2 条件权重排序

通过对条件的逐条分析，确定权重排序：

1.食蚁兽也在路上活动，一旦与食蚁兽相遇，会遭遇生命危险；（附件图中标红色路段）；I_4

2.必须经过玉米间，水果间（附件图中标绿色节点）；I_3

3.有两段路是必须经过的，那里有神秘礼物(附件图中标绿色路段)。I_5

4.最多只能经过 9 个储藏间拿到食物（包含起止两个节点）；I_2

$$I_4=4>I_3=3>I_5=2>I_2=1$$

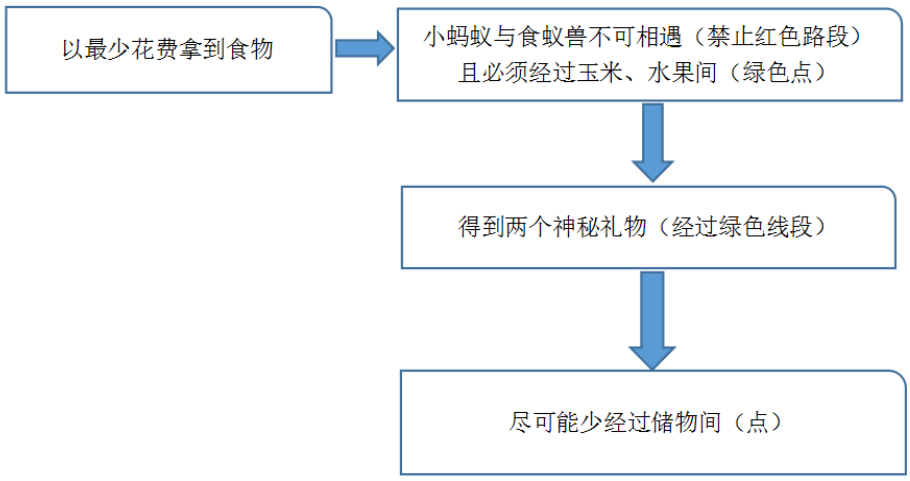


图 3.2.1 条件权重排序图

将图中各个储藏间及之间的路径抽象为带权无向图模型，将问题转化为求带权无向图中经过指定中间节点集的最短路径，按照 3.2 中条件重要性顺序设计算法求解满足题意的路径，确定最优解。

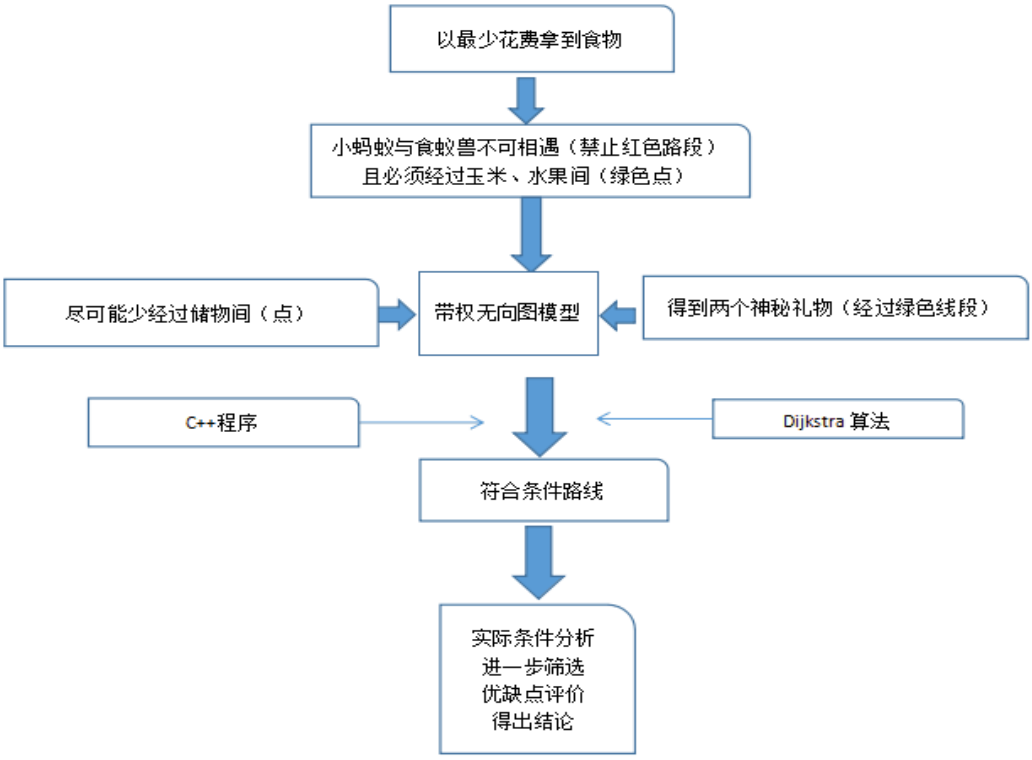


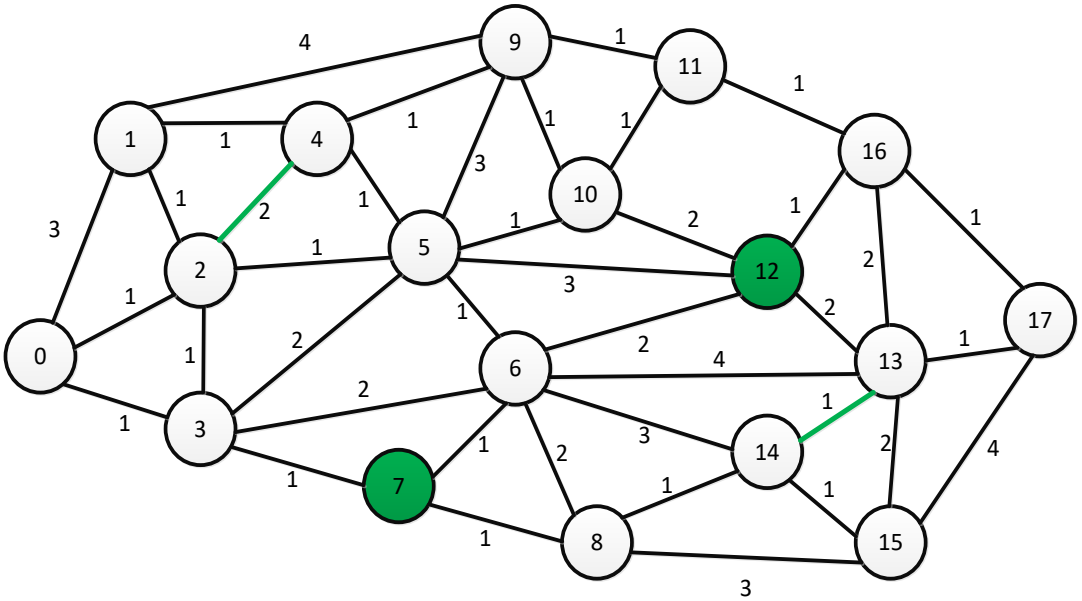
图 3.2.2 建模思路流程图

四、模型建立与求解

4.1 模型准备

将蚂蚁巢简化为一个带权无向图模型，每两个储物间之间的花费简化为边的权值。于是，问题转化成了寻找 S 点和 E 点之间的最短路径问题经过指定中间节点集和边集的最短路径问题。

首先，我们满足重要性最大的条件即条件 4：不与食蚁兽相遇。将题目附带的图整理简化为下图，直接将红色的边删除。



并将图转换为数据，以便程序读入存入邻接矩阵

0 1 3
0 2 1
0 3 1
1 4 1
1 2 1
1 9 4
2 4 2
4 5 1
5 9 3
4 9 1
2 5 1
5 10 1
9 10 1
2 3 1
3 5 2
3 6 2
5 6 1
3 7 1
6 7 1
6 8 2
7 8 1
5 12 3
10 12 2
10 11 1
9 11 1
11 16 1
11 12 1
6 12 2
6 13 4
12 13 2
12 16 1
13 16 2
13 14 1
6 14 3
8 14 1
8 15 3
14 15 1
13 14 1
13 15 2
16 17 1
13 17 1
15 17 4

4.2 模型的求解

（1） **Dijkstra 算法求带权无向图的单源最短路**

在重要性最高的第四个条件满足后，我们用深度优先搜索的方法来寻找所有可能的路径，共找到了 148895 条路径，因为数量太多，无法一一列举，搜索代码见附录。

之后我们运用 Dijkstra 算法来寻求最短路径

Dijkstra 算法的基本思路是：先将与起点有边直接相连的节点到起点的距离记为对应的边的权重值，将与起点无边直接相连的节点到起点的距离记为无穷大。然后以起点为中心向外层层扩展，计算所有节点到起点的最短距离。每次新扩展到一个距离最短的点后，更新与它有边直接相邻的节点到起点的最短距离。当所有点都扩展进来后，所有节点到起点的最短距离将不会再被改变，因而保证了算法的正确性

[1] Dijkstra 算法求解最短路径问题的基本步骤如下：

①设立 Y 和 N 两个集合， Y 用于保存所有等待访问的节点，N 记录所有已经访问过的节点。

②访问网络节点中距离起始节点最近且没有被访问过的节点，把这个节点放入 Y 中等待访问。

③从 Y 中找出距离起点最近的节点，放入 N 中，更新与这个节点有边直接相连的相邻节点到起始节点的最短距离，同时把这些相邻节点加入 Y 中。

$$d(j)=\min(d(j),\mathrm{a}_{jk}+d(\mathrm{k}))$$

④重复步骤（2）和（3），直到 Y 集合为空， N 集合为网络中所有节点为止。

[2]由于题中图节点较多，下面用该图来演示一下 Dijkstra 算法

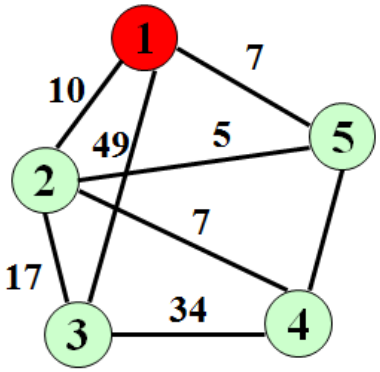
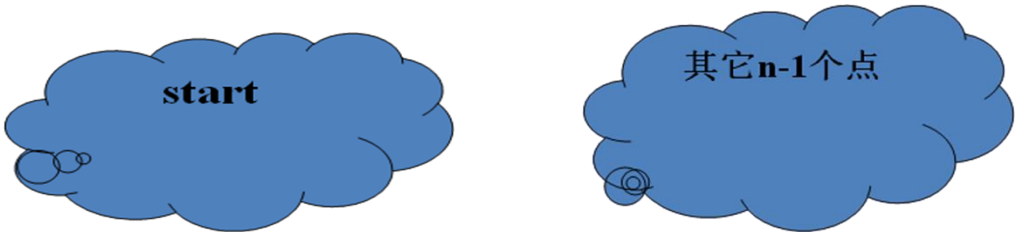


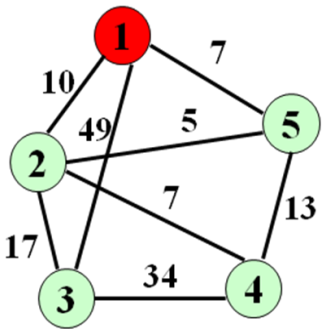
图 4.2 Dijkstra 算法演示样图

开始点（源点）：start
D[i]:顶点 i 到 start 的最短距离。
初始：
D[start]=0;
D[i]=a[start,i]　（无边设为 INF）

集合 1：已求点　　集合 2：未求点



算法步骤：
1、在集合 2 中找一个到 start 距离最近的顶点 k ，距离=d[k]
2、把顶点 k 加到集合 1 中，同时修改集合 2 中的剩余顶点 j 的 d[j]是否经过 k 后变短。如果变短修改 d[j]
If （ d[k]+a[k,j]<d[j]） → d[j]=d[k]+a[k,j]
3、重复 1，直至集合 2 空为止。




```
5
1
1 2 10
1 5 7
2 3 17
2 4 7
2 5 5
3 4 34
4 5 13
```

顶点	1	2	3	4	5
F[i]	T	F	F	F	F
D[i]	0	10	49	∞	7
Path[i]	1	1,2	1,3		1,5

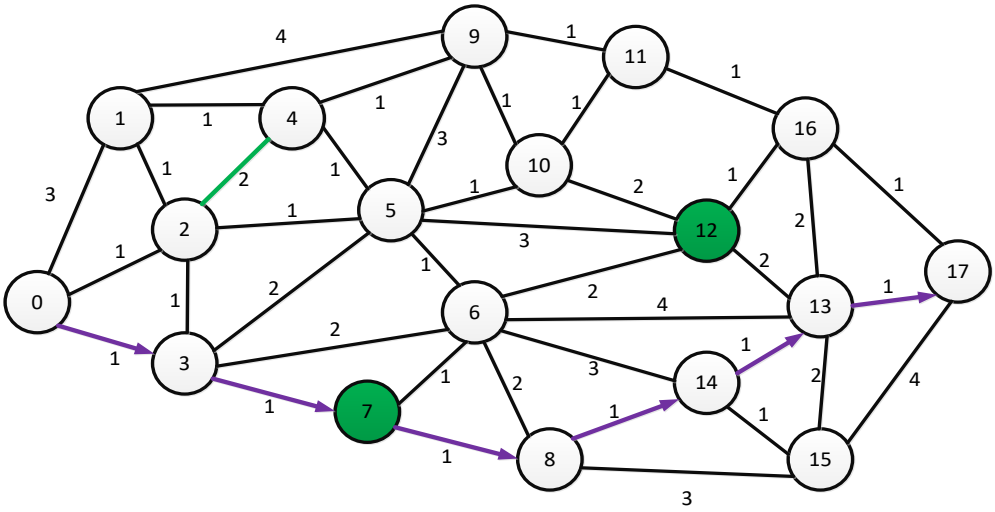
顶点	1	2	3	4	5
F[i]	T	F	F	F	T
D[i]	0	10	49	20	7
Path[i]	1	1,2	1,3	1,5,4	1,5

顶点	1	2	3	4	5
F[i]	T	T	F	F	T
D[i]	0	10	27	17	7
Path[i]	1	1,2	1,2,3	1, 2,4	1,5

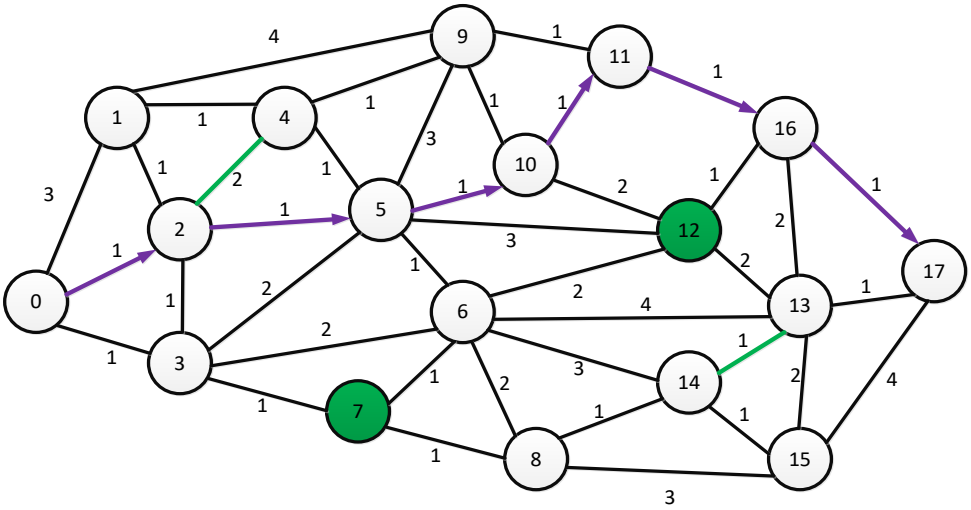
顶点	1	2	3	4	5
F[i]	T	T	T	T	T
D[i]	0	10	27	17	7
Path[i]	1	1,2	1,2,3	1, 2,4	1,5

顶点	1	2	3	4	5
F[i]	T	T	F	T	T
D[i]	0	10	27	17	7
Path[i]	1	1,2	1,2,3	1, 2,4	1,5

[3]至此，我们用可以用典型的 Dijkstra 算法可以找出满足重要性最大的条件 4 的最短路径（实现代码见附录）



路径①



路径②

路径①，②皆经过 7 个点，花费为 6

（2）基于 Dijkstra 算法的带权无向图经过指定边及节点集的单源最短路径模型

在重要性最高的两个条件 4、3 都满足的情况下（蚂蚁既不与食蚁兽相遇，又经过水果间和玉米间），我们用深度优先搜索的方法来寻找所有可能的路径，共找到了 57242 条路径，因为数量太多，无法一一列举，搜索代码见附录。

之后我们改进典型的 Dijkstra 算法来寻求最短路径

上文展示了 Dijkstra 算法的主要思想，但考虑本题中的限制条件后，典型 Dijkstra 算法就不再适用了，需要对其进行改进。我们再考虑满足重要性第二大的条件 3，即寻找包含 7、12 点的最短路径

1.经过指定节点集的改进

贪心算法，是一种通过分级处理某些最优解问题的方法。贪心算法，在求解过程的每一步中都采取在当前状态下看来是最好或最优的选择，从而希望最终的结果是最好或最优的。如果一个复杂的问题能够分解成几个子问题来解决，并且子问题的最优解能递推到最终问题的最优解，简言之，如果局部最优解能最终推导出全局最优解，那么贪心算法在解决这类问题中将会非常有效

贪心算法求解问题的基本步骤：

①把原问题分解成若干个易于求解的简化的子问题。

②对每一子问题分别求解，得到所有子问题的局部最优解。

③通过对全部子问题的某个局部最优解进行分析、聚合等处理，生成原问题的一个解。

④从上一步所求出的原问题的解集或某个初始解出发，通过筛选、迭代等手段求出原问题的解（全局最优解）。

我们算法的改进就是基于贪心算法思想的，将原问题分解成几个易于 Dijkstra 算法求解的子问题，先对各个子问题逐一求局部最优解，再在此基础上求全局最优解。具体说来，就是先将预处理后的网络拓扑图中的所有节点分为三个子集合，分别为包含起点的**起点集**，包含全部的需要经过的中间节点的**中间节点集**，包含终点的**终点集**。通过 Dijkstra 算法依次求起点集到中间节点集之间的局部最短路径，连通中间节点集中所有节点的局部最短路径，中间节点集到终点集之间的局部最短路径，以此求出一条从起点出发经过指定的所有中间节点后到达终点的待选全局最短路径。通过对有限的待选全局最短路径进行筛选，选出其中距离最短的路径，即为满足要求的全局最短路径

对本题来说

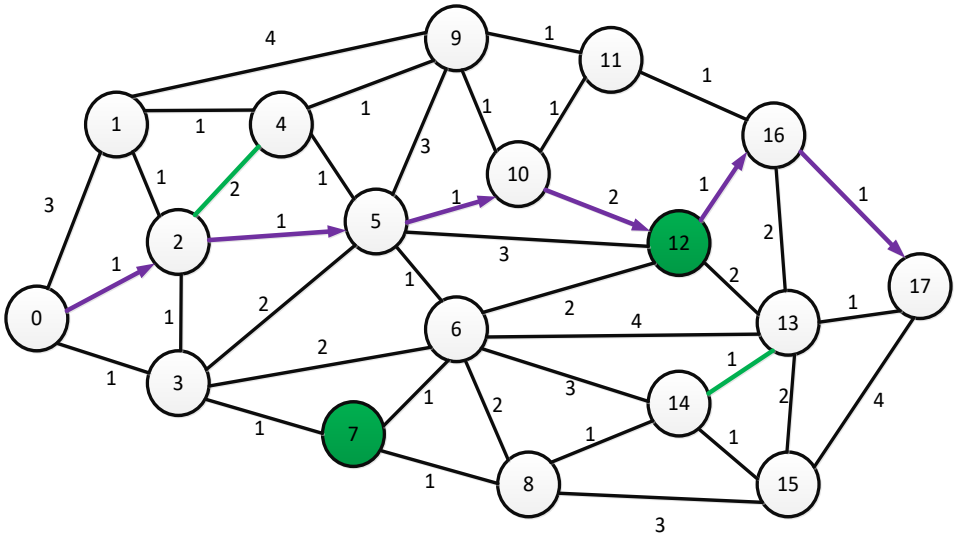
$$ans = \text{dijkstra}(0,7)+\text{dijkstra}(7,12)+\text{dijkstra}(12,17)$$

或

$$ans = \text{dijkstra}(0,12)+\text{dijkstra}(12,7)+\text{dijkstra}(7,17)$$

二者结果是一样的。

这样，我们可以找出满足条件 4、3 的最短路径(实现代码见附录)



2.经过指定节点集和边集的改进

在经过指定节点集的 Dijkstra 算法的基础上，将指定边的两节点加入到指定节点的集合中，并且该边的权值记录下来，图中的权值赋值为一个极大值，即不能重复走这条边

这样，指定节点集中就有了 6 个点

对这 6 个点的排列顺序进行全排列，对每一种组合进行判断，如果两条特殊边的两顶点不靠在一起，就舍弃这种排列

假设 E 中以 i，j 为起点终点的路径最短

$$ans = \text{dijkstra}(0,i)+\text{dijkstra}(i,j)+\text{dijkstra}(j,17)$$

这样，我们就只剩最后一个条件没有满足了。因为这个条件在本题中其他条件限制下是不能满足的，所以将其最多经过的储藏间个数扩大为 10 个、11 个、12 个·····通过深度优先搜索算法来寻找所有可能的路径

①扩大为 10 个时仍然是无解的

②扩大为 11 个点时有 1 条路径

0->2->4->5->12->6->7->8->14->13->17　　14（指总花费）

③扩大为 12 个点时共有 63 条路径

路径	总花费
0->1->2->4->5->12->6->7->8->14->13->17	17
0->1->4->2->3->7->6->12->13->14->15->17	19
0->1->4->2->3->7->6->14->13->12->16->17	17
0->1->4->2->3->7->8->14->13->12->16->17	15
0->1->4->2->5->12->6->7->8->14->13->17	17
0->2->4->5->3->7->6->12->13->14->15->17	18
0->2->4->5->3->7->6->14->13->12->16->17	16
0->2->4->5->3->7->8->14->13->12->16->17	14
0->2->4->5->6->7->8->14->13->12->16->17	13
0->2->4->5->10->12->6->7->8->14->13->17	14

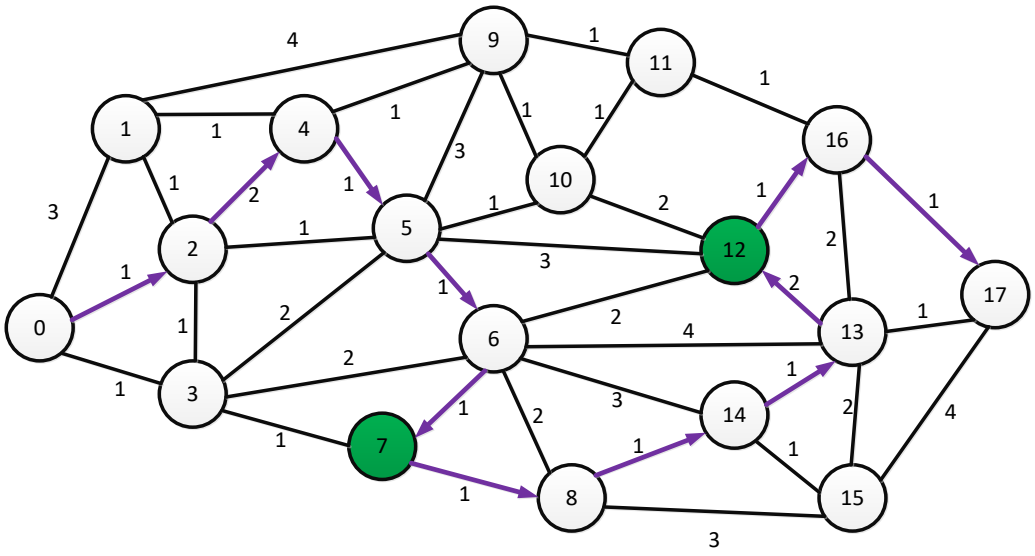
0->2->4->5->12->6->3->7->8->14->13->17	16
0->2->4->5->12->6->7->8->14->13->15->17	19
0->2->4->5->12->6->7->8->14->13->16->17	16
0->2->4->5->12->6->7->8->14->13->17	14
0->2->4->5->12->6->7->8->15->14->13->17	17
0->2->4->5->12->13->14->6->7->8->15->17	22
0->2->4->9->5->12->6->7->8->14->13->17	17
0->2->4->9->10->12->6->7->8->14->13->17	14
0->3->2->4->5->12->6->7->8->14->13->17	15
0->2->4->5->3->7->6->12->13->14->15->17	18
0->2->4->5->3->7->6->14->13->12->16->17	16
0->2->4->5->3->7->8->14->13->12->16->17	14
0->2->4->5->6->7->8->14->13->12->16->17	13
0->2->4->5->10->12->6->7->8->14->13->17	14
0->2->4->5->12->6->3->7->8->14->13->17	16
0->2->4->5->12->6->7->8->14->13->15->17	19
0->2->4->5->12->6->7->8->14->13->16->17	16
0->2->4->5->12->6->7->8->14->13->17	14
0->2->4->5->12->6->7->8->15->14->13->17	17
0->2->4->5->12->13->14->6->7->8->15->17	22
0->2->4->9->5->12->6->7->8->14->13->17	17
0->2->4->9->10->12->6->7->8->14->13->17	14
0->3->2->4->5->12->6->7->8->14->13->17	15
0->1->2->4->5->12->6->7->8->14->13->17	17
0->1->4->2->3->7->6->12->13->14->15->17	19
0->1->4->2->3->7->6->14->13->12->16->17	17
0->1->4->2->3->7->8->14->13->12->16->17	15
0->1->4->2->5->12->6->7->8->14->13->17	17
0->2->4->5->3->7->6->12->13->14->15->17	18
0->2->4->5->3->7->6->14->13->12->16->17	16
0->2->4->5->3->7->8->14->13->12->16->17	14
0->2->4->5->6->7->8->14->13->12->16->17	13
0->2->4->5->10->12->6->7->8->14->13->17	14
0->2->4->5->12->6->3->7->8->14->13->17	16
0->2->4->5->12->6->7->8->14->13->15->17	19
0->2->4->5->12->6->7->8->14->13->16->17	16
0->2->4->5->12->6->7->8->14->13->17	14
0->2->4->5->12->6->7->8->15->14->13->17	17
0->2->4->5->12->13->14->6->7->8->15->17	22
0->2->4->9->5->12->6->7->8->14->13->17	17
0->2->4->9->10->12->6->7->8->14->13->17	14
0->1->2->4->5->12->6->7->8->14->13->17	17
0->1->4->2->5->12->6->7->8->14->13->17	17

0->2->4->5->6->7->8->14->13->12->16->17	13
0->2->4->5->10->12->6->7->8->14->13->17	14
0->2->4->5->12->6->7->8->14->13->15->17	19
0->2->4->5->12->6->7->8->14->13->16->17	16
0->2->4->5->12->6->7->8->14->13->17	14
0->2->4->5->12->6->7->8->15->14->13->17	17
0->2->4->5->12->13->14->6->7->8->15->17	22
0->2->4->9->5->12->6->7->8->14->13->17	17
0->2->4->9->10->12->6->7->8->14->13->17	14
0->3->2->4->5->12->6->7->8->14->13->17	15

④扩大到 13 个点时有 331 条路径，因为路径太多，无法一一列举

⑤·····

用改进后的 Dijkstra 算法求得最优路线是

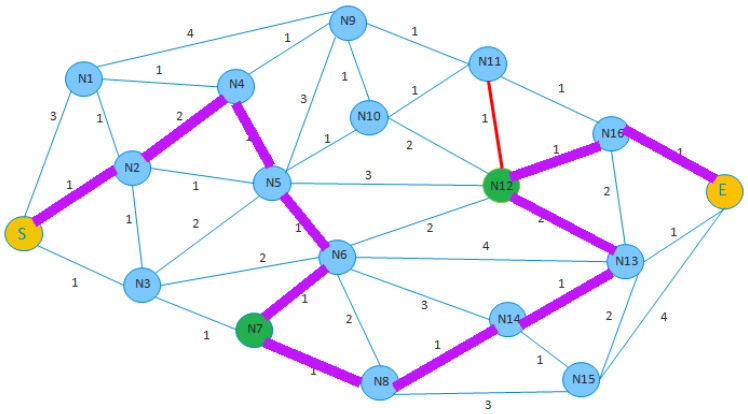


共经过 12 个点，总花费是 13

结合上述数据验证，这条路径的确是花费最少的路径

虽然经过的点不是最少的，但实现了其他条件下花费最少，又因为经过储藏间的个数是对结果影响最小的一个条件，所以，此道路是最优解。

所以，本题的最优解为



五、模型评价

5.1 模型优缺点

5.1.1 模型优点

(1)逐步求解方法可以保证模型中最后结果的精确性，使误差变得尽可能的小。

（2）抽象出的模型具有可转移性，当研究问题中的限制条件发生改变时，可更改所建模型中相关代码实现方法的通用应用。

(3)运用 C++语言编写程序，方便高效地完成求解带权无向模型。

5.1.2 模型缺点

(1)缺点:条件筛选所用的条件增加逐步求解方法，过程略微繁琐。

（2）在运行代码搜索路径的过程中，若有多种花费相同的符合条件可选择路径，只能输出其中一条。只能通过后续分析筛选出最优路径。

5.2 模型改进

（1）通过 lingo 的敏感分析（如分析已知量的变化，分析最优函数的反应）来进行更深层次的数据分析。

（2）可以用链表和结构体对算法进行优化。

推广：本文的求解方法可以适用于交通路线的选择等实际问题。

参考文献

[1] 黄书力，胡大裟，蒋玉明. 经过指定的中间节点集的最短路径算法. [计算机工程与应用] 41-45 2015，51（11）

[2] Thomas H.Cormen、Charles E.Leiserson 《算法导论》 北京 机械工业出版社 2014

[3] 刘汝佳 《算法竞赛入门经典》 北京 清华大学出版社 2015

附录：

① Dijstra 算法(满足不经过食蚁兽红边)

```
/*
程序会输出最优解和最优路径
*/
#include <iostream>
#include<stdio.h>
#include<cstring>
#include<stack>
#define maxn 536870912
#define up(x,y) for(int i=x;i<=y;i++)
#define mem(x) memset(x,0,sizeof(x))
using namespace std;
  int mapp[20][20];
      bool b[20];
      int d[20],fa[20];
stack<int>q;
int main()
{
      freopen("input.txt","r",stdin);
      int t,n,x,y,e,v,minn;
      n=17;
      t=42;
      up(0,n)
         for(int j=0;j<=n;j++)
            if(i==j)mapp[i][j]=0;else mapp[i][j]=maxn;
      while(t--)
      {
            cin>>x>>y>>e;
            mapp[x][y]=mapp[y][x]=e;

      }
      mem(b);mem(fa);
      up(0,n)
         {
               d[i]=mapp[0][i];
         }
      mapp[11][12]=mapp[12][11]=maxn;//不经过食蚁兽红边
      mapp[2][4]=mapp[4][2]=0;
      mapp[13][14]=mapp[14][13];
      up(1,n-1)//从每次找最小的边开始，用这条边松弛别的边
            {
            minn=maxn;
```

```
        for(int j=0;j<=n;j++)
           if(b[j]==0&&d[j]<minn)
           {
                minn=d[j];
                v=j;
           }
        b[v]=1;
        for(int j=0;j<=n;j++)
             if(b[j]==0&&d[j]>mapp[j][v]+d[v])//读入图的边，存入邻接矩阵
                {
                     d[j]=mapp[j][v]+d[v];
                     fa[j]=v;
                }
    }
    cout<<d[n]<<"\n";
    while(fa[n]!=0)
    {
        q.push(fa[n]);
        n=fa[n];
    }
    q.push(0);
    while(q.empty()==0)//输出最短路径
    {
        cout<<q.top()<<"->";
        q.pop();
    }
    cout<<"17\n";
   return 0;
}
```

② Dijstra 算法经过指定节点集的改进

```
    #include <iostream>
    #include<stdio.h>
    #include<cstring>
    #define maxn 536870912
    #define n 17
    #define t 42
    using namespace std;
    int mapp[20][20];
    bool b[20];
    int d[20],fa[20];
    int x,y,e,v,minn;
    int dijkstra(int s,int e)
    {
```

```
    memset(b,0,sizeof(b));
    memset(fa,0,sizeof(fa));
    for(int i=0;i<=n;i++)
      {
          d[i]=mapp[s][i];
      }
    for(int i=1;i<=n-1;i++)//从每次找最小的边开始，用这条边松弛别的边
        {
        minn=maxn;              for(int j=0;j<=n;j++)
          if(b[j]==0&&d[j]<minn)
          {
              minn=d[j];
              v=j;
          }
        b[v]=1;
        for(int j=0;j<=n;j++)
            if(b[j]==0&&d[j]>mapp[j][v]+d[v])
              {
                  d[j]=mapp[j][v]+d[v];
                  fa[j]=v;
              }
    }
  return d[e];

}
int main()
{
    freopen("input.txt","r",stdin);
    for(int i=0;i<=n;i++)
      for(int j=0;j<=n;j++)
        if(i==j)mapp[i][j]=0;else mapp[i][j]=maxn;//读入图的边，存入邻接矩阵
    for(int i=1;i<=t;i++)
    {
        cin>>x>>y>>e;
        mapp[x][y]=mapp[y][x]=e;
    }
    mapp[11][12]=mapp[12][11]=maxn;;//去掉食蚁兽红边
    cout<<dijkstra(0,7)+dijkstra(7,12)+dijkstra(12,17)<<endl;
  return 0;
}
```

③ Dijkstra 算法经过指定节点集和边集的改进

```
/*
程序最终会输出最优解，通过最优解的值可以从上面输出的路径中找出
最优解的路径的中间点集的顺序
输入数据即上文 4.1 模型准备中将图转换为的数据
*/
#include <iostream>
#include<stdio.h>
#include<cstring>
#include<algorithm>
#include<stack>
#define maxn 536870912
#define n 17
#define t 42
using namespace std;
int mapp[20][20];
int a[10]={2,4,7,13,14,12};
bool b[20];
int d[20],fa[20];
int x,y,e,v,minn;
stack<int>q;
stack<int>o;
int dijkstra(int s,int e)
{
      memset(b,0,sizeof(b));
      memset(fa,0,sizeof(fa));
      for(int i=0;i<=n;i++)
        {
            d[i]=mapp[s][i];
        }
      for(int i=1;i<=n-1;i++)//从每次找最小的边开始，用这条边松弛别的边
          {
          minn=maxn;//先吧 min 给赋初值
          for(int j=0;j<=n;j++)
            if(b[j]==0&&d[j]<minn)
            {
                  minn=d[j];
                  v=j;
            }
          b[v]=1;
          for(int j=0;j<=n;j++)
              if(b[j]==0&&d[j]>mapp[j][v]+d[v])
                  {
                        d[j]=mapp[j][v]+d[v];
```

```
                    fa[j]=v;
                }
        }
    return d[e];

}
int main()
{
    //freopen("input.txt","r",stdin);
    for(int i=0;i<=n;i++)
        for(int j=0;j<=n;j++)
            if(i==j)mapp[i][j]=0;else mapp[i][j]=maxn;//读入图的边，存入邻接矩阵
    for(int i=1;i<=t;i++)
    {
        cin>>x>>y>>e;
        mapp[x][y]=mapp[y][x]=e;
    }
    mapp[11][12]=mapp[12][11]=maxn;//去掉食蚁兽红边
    mapp[2][4]=mapp[4][2]=maxn;//绿色特殊便
    mapp[13][14]=mapp[14][13]=maxn;//绿色特殊边
    int sf,sum,cnt(100000);
    while(next_permutation(a,a+5))//中间节点集的 6 个点的全排列
    {
        sf=0;
        for(int i=0;i<=5;i++)
        {
            if(a[i]==4)//判断绿色特殊边
                if(a[i+1]==2||a[i-1]==2)
                    sf++;
            if(a[i]==2)
                if(a[i+1]==4||a[i-1]==4)
                    sf++;
            if(a[i]==14)
                if(a[i+1]==13||a[i-1]==13)
                    sf++;
            if(a[i]==13)
                if(a[i+1]==14||a[i-1]==14)
                    sf++;
        }
        if(sf==4)//如果两条特殊边都经过
        {
            cout<<0<<' ';
            for(int i=0;i<=5;i++)cout<<a[i]<<' ';cout<<17<<endl;
            sum=0;
```

```
                for(int i=0;i<=4;i++)
                {
                    if((a[i]==4&&a[i+1]==2)||(a[i]==2&&a[i+1]==4))
                    {
                        sum+=2;
                    }
                    else
                    if((a[i]==13&&a[i+1]==14)||(a[i]==14&&a[i+1]==13))
                    {
                        sum+=1;
                    }
                    else
                    {
                    sum+=dijkstra(a[i],a[i+1]);
                    }
                }
            sum+=dijkstra(0,a[0]);
            sum+=dijkstra(a[5],17);
            cout<<sum<<"----------------"<<endl;
            if(sum<cnt)
                cnt=sum;
        }
        }
        cout<<"\n 最优解："<<cnt<<endl;
    return 0;
}
```

④ 满足不经过食蚁兽红线的路径搜索代码

```
#include <iostream>
#include<stdio.h>
#include<cstring>
#include<stack>
#define maxn 536870912
using namespace std;
int mapp[20][20];
bool b[20];
int d[20],fa[20];
int t(42),n(17),x,y,e,v,minn;
void print(int x)
{
    stack<int>q;
    int flag(0);
    while(x!=0)
    {
        q.push(x);
```

```
            x=fa[x];
        }
        q.push(0);
        while(q.empty()==0)
        {
            cout<<q.top()<<"->";
            q.pop();
        }
        cout<<"17\n";
}
void dfs(int x)
{
        if(x==17)
        {
            print(fa[x]);
            return;
        }
        for(int i=0;i<=n;i++)
            if(b[i]==0&&mapp[x][i]<maxn)
            {
                b[i]=1;
                fa[i]=x;
                dfs(i);
                b[i]=0;
            }
}
int main()
{
        freopen("input.txt","r",stdin);
        freopen("output.txt","w",stdout);//若采用文件输入输出可以去掉注释
        for(int i=0;i<=n;i++)
           for(int j=0;j<=n;j++)
            if(i==j)mapp[i][j]=0;else mapp[i][j]=maxn;//读入图的边,存入邻接矩阵
        while(t--)
        {
            cin>>x>>y>>e;
            mapp[x][y]=mapp[y][x]=e;
        }
        mapp[11][12]=mapp[12][11]=maxn;
        memset(b,0,sizeof(b));
        memset(fa,0,sizeof(fa));
        dfs(0);
      return 0;
}
```

⑤ 满足不经过食蚁兽红线和经过玉米水果间的路径搜索代码

```
#include <iostream>
#include<stdio.h>
#include<cstring>
#include<stack>
#define maxn 536870912
using namespace std;
int mapp[20][20];
bool b[20];
int d[20],fa[20];
int t(42),n(17),x,y,e,v,minn;
void print(int x)
{
	stack<int>q;
	int flag(0);
	while(x!=0)
	{
		q.push(x);
		if(x==7||x==12)flag++;
		x=fa[x];
	}
	q.push(0);
	if(flag!=2)return ;
	while(q.empty()==0)
	{
		cout<<q.top()<<"->";
		q.pop();
	}
	cout<<"17\n";
}
void dfs(int x)
{
	if(x==17)
	{
		print(fa[x]);
		return;
	}
	for(int i=0;i<=n;i++)
		if(b[i]==0&&mapp[x][i]<maxn)
		{
			b[i]=1;
			fa[i]=x;
			dfs(i);
			b[i]=0;
```

```
            }
        }
        int main()
        {
            //freopen("input.txt","r",stdin);
            //freopen("output.txt","w",stdout);//若采用文件输入输出可以去掉注释
            for(int i=0;i<=n;i++)
               for(int j=0;j<=n;j++)
                  if(i==j)mapp[i][j]=0;else mapp[i][j]=maxn;//读入图的边,存入邻接矩阵
            while(t--)
            {
                cin>>x>>y>>e;
                mapp[x][y]=mapp[y][x]=e;
            }
            mapp[11][12]=mapp[12][11]=maxn;
            memset(b,0,sizeof(b));
            memset(fa,0,sizeof(fa));
            dfs(0);
           return 0;
        }
```

⑥储物间个数扩大的是搜索路径的代码

```
#include <iostream>
#include<stdio.h>
#include<cstring>
#include<stack>
#define maxn 536870912
using namespace std;
int mapp[20][20];
bool b[20];
int d[20],fa[20];
int t(42),n(17),x,y,e,v,minn;
void print(int x)
{
    stack<int>q;
    int flag(0);
    while(x!=0)
    {
        q.push(x);
        if(x==7||x==12)flag++;
        if((x==4&&fa[x]==2)||(x==2&&fa[x]==4))flag++;
        if((x==14&&fa[x]==13)||(x==13&&fa[x]==14))flag++;
        x=fa[x];
    }
    q.push(0);
```

```
    if(flag!=4||q.size()>11)return ;//要实现寻找 10、11、12 个点的路径，只需把 q.size()>后面的数字修改即可
    while(q.empty()==0)
    {
        cout<<q.top()<<"->";
        q.pop();
    }
    cout<<"17\n";
}
void dfs(int x)
{
    if(x==17)//如果搜索到终点就打印路径并输出
    {
        print(fa[x]);
        return;
    }
    for(int i=0;i<=n;i++)
        if(b[i]==0&&mapp[x][i]<maxn)
        {
            b[i]=1;//选择这个点的方向走
            fa[i]=x;//记录父节点
            dfs(i);
            b[i]=0;//不选这个点的情况
        }
}
int main()
{
    //freopen("input.txt","r",stdin);//若采用文件输入输出可以去掉注释
    //freopen("output.txt","w",stdout);
    for(int i=0;i<=n;i++)
       for(int j=0;j<=n;j++)
        if(i==j)mapp[i][j]=0;else mapp[i][j]=maxn;//读入图的边,存入邻接矩阵
    while(t--)
    {
        cin>>x>>y>>e;
        mapp[x][y]=mapp[y][x]=e;
    }
    mapp[11][12]=mapp[12][11]=maxn;
    memset(b,0,sizeof(b));
    memset(fa,0,sizeof(fa));
    dfs(0);

   return 0;
}
```