**CCF 2013-2016年第四题题解及源码**

[**CCF201312-4 有趣的数（100分）**](http://blog.csdn.net/tigerisland45/article/details/55270910)

|  |  |
| --- | --- |
| 试题编号： | 201312-4 |
| 试题名称： | 有趣的数 |
| 时间限制： | 1.0s |
| 内存限制： | 256.0MB |
| 问题描述： | **问题描述**  　　我们把一个数称为有趣的，当且仅当： 　　1. 它的数字只包含0, 1, 2, 3，且这四个数字都出现过至少一次。 　　2. 所有的0都出现在所有的1之前，而所有的2都出现在所有的3之前。 　　3. 最高位数字不为0。 　　因此，符合我们定义的最小的有趣的数是2013。除此以外，4位的有趣的数还有两个：2031和2301。 　　请计算恰好有n位的有趣的数的个数。由于答案可能非常大，只需要输出答案除以1000000007的余数。  **输入格式**  　　输入只有一行，包括恰好一个正整数n (4 ≤ n ≤ 1000)。  **输出格式**  　　输出只有一行，包括恰好n 位的整数中有趣的数的个数除以1000000007的余数。  **样例输入**  4  **样例输出**  3 |

问题链接：[CCF201312试题](http://blog.csdn.net/tigerisland45/article/details/54446795)。

**原题链接**：[有趣的数](http://115.28.138.223/view.page?gpid=T2)。

**问题描述**：参见上文。

**问题分析**：这是一个计算问题，关键在于找到一个递推式。只要找到一个递推式，问题就解决了。有时候这类问题也用DP（动态规划）来解决。

根据题意，有趣的数满足以下约束条件如下：

1.只包含数字0、1、2和3

2.0、1、2和3各自至少出现一次

3.所有的0都出现在1之前

4.所有的2都出现在3之前

5.最高位数字不为0

数（字符串）可以分为以下六种情况（或称为状态）：

1.只包含数字2，记为S1

2.只包含数字2和0（0开始的数0个，以此数为前缀的数均不是以0开始），记为S2

3.只包含数字2和3，记为S3

4.只包含数字2、0和1，并且满足所有0在1之前，记为S4

5.只包含数字2、0和3，并且满足所有2在3之前，记为S5

6.包含任意数字（包含0、1、2和3），满足所有0在1之前，满足所有2在3之前，记为S6

考虑递推式：

1.对于S1，考虑其长度l，定义f(l,S1)为长度l的S1数的数量，则f(l,S1)＝１。也就是说长度为l的只包含2的数只有1个。

2.对于S2，考虑其长度l，定义f(l,S2)为长度l的S2数的数量，当l=1则那么f(l,S2)=0，即f(1,S2)=0，当l>1则f(l,S2)=f(l-1,S2)\*2+f(l-1,S1)。这是因为，长度为l的S2数可以是由长度为l-1的S2的数加上2或0构成，例20为长度为2的S2数，那么202和200为长度为3的S2数；另外，长度为l的S2数可以是由长度为l-1的S1的数加上0构成，例如22为长度为2的S1数，那么220为长度为3的S2数。

3.对于S3，考虑其长度l，定义f(l,S3)为长度l的S3数的数量，当l=1则那么f(l,S3)=0，即那么f(1,S3)=0，当l>1则f(l,S3)=f(l-1,S3)\*2+f(l-1,S1)。这是因为，长度为l的S3数可以是由长度为l-1的S3的数加上2或3构成，例23为长度为2的S3数，那么232和233为长度为3的S3数；另外，长度为l的S3数可以是由长度为l-1的S1的数加上3构成，例如22为长度为2的S1数，那么223为长度为3的S3数。

4.对于S4，考虑其长度l，定义f(l,S4)为长度l的S4数的数量，当l=1则那么f(l,S4)=0，即f(1,S4)=0，当l>1则f(l,S4)=f(l-1,S4)\*2+f(l-1,S2)+f(l-1,S3)。这是因为，长度为l的S4数可以是由长度为l-1的S4的数加上2或1构成（满足所有0在1之前，不可以加上0），例201为长度为3的S4数，那么2012和2011为长度为4的S4数；另外，长度为l的S4数可以是由长度为l-1的S2的数加上1构成，例如20为长度为2的S2数，那么201为长度为3的S4数。

同理，对于S5和S6，可以得到相应的递推公式。详细参见以下的源程序。

有了递推式，程序就可以根据递推式，逐步进行计算。

**程序说明**：（略）。

提交后得100分的[**C++语言**](http://lib.csdn.net/base/c)程序如下：

**[cpp]** [view plain](http://blog.csdn.net/tigerisland45/article/details/55270910) [copy](http://blog.csdn.net/tigerisland45/article/details/55270910)

1. /\* CCF201312-4 有趣的数 \*/
3. #include <iostream>
4. #include <cstring>
6. **using** **namespace** std;
8. **const** **long** **long** MOD = 1000000007;
9. **const** **int** MAXN = 1000;
10. **const** **int** MAXS = 5;
11. **long** **long** status[MAXN+1][MAXS+1];

14. **int** main()
15. {
16. **int** n;
18. cin >> n;
20. memset(status, 0, **sizeof**(status));
22. // DP
23. status[1][0] = 1;
24. **for**(**int** i=2; i<=n; i++) {
25. status[i][0] = 1;
26. status[i][1] = (status[i - 1][1] \* 2 + status[i - 1] [0]) % MOD;
27. status[i][2] = (status[i - 1][2] + status[i - 1][0]) % MOD;
28. status[i][3] = (status[i - 1][3] \* 2 + status[i - 1][1] ) % MOD;
29. status[i][4] = (status[i - 1][4] \* 2 + status[i - 1][1] + status[i - 1][2]) % MOD;
30. status[i][5] = (status[i - 1][5] \* 2 + status[i - 1][3] + status[i - 1][4]) % MOD;
31. }
33. cout << status[n][5] << endl;
35. **return** 0;
36. }

[**CCF201403-4 无线网络（100分)**](http://blog.csdn.net/tigerisland45/article/details/54954364)

|  |  |
| --- | --- |
| 试题编号： | 201403-4 |
| 试题名称： | 无线网络 |
| 时间限制： | 1.0s |
| 内存限制： | 256.0MB |
| 问题描述： | **问题描述**  　　目前在一个很大的平面房间里有 n 个无线路由器,每个无线路由器都固定在某个点上。任何两个无线路由器只要距离不超过 r 就能互相建立网络连接。 　　除此以外,另有 m 个可以摆放无线路由器的位置。你可以在这些位置中选择至多 k 个增设新的路由器。 　　你的目标是使得第 1 个路由器和第 2 个路由器之间的网络连接经过尽量少的中转路由器。请问在最优方案下中转路由器的最少个数是多少?  **输入格式**  　　第一行包含四个正整数 n,m,k,r。(2 ≤ n ≤ 100,1 ≤ k ≤ m ≤ 100, 1 ≤ r ≤ 108)。 　　接下来 n 行,每行包含两个整数 xi 和 yi,表示一个已经放置好的无线 路由器在 (xi, yi) 点处。输入数据保证第 1 和第 2 个路由器在仅有这 n 个路由器的情况下已经可以互相连接(经过一系列的中转路由器)。 　　接下来 m 行,每行包含两个整数 xi 和 yi,表示 (xi, yi) 点处可以增设 一个路由器。 　　输入中所有的坐标的绝对值不超过 108,保证输入中的坐标各不相同。  **输出格式**  　　输出只有一个数,即在指定的位置中增设 k 个路由器后,从第 1 个路 由器到第 2 个路由器最少经过的中转路由器的个数。  **样例输入**  5 3 1 3 0 0 5 5 0 3 0 5 3 5 3 3 4 4 3 0  **样例输出**  2 |

**问题链接：**[CCF201403试题](http://blog.csdn.net/tigerisland45/article/details/54430251)。

**问题描述**：（参见上文）。

**问题分析**：这是一个求最优问题，通常用BFS（广度优先搜索）来实现。据称，该问题还可以用SPFA算法来实现。

**程序说明**：数组visited[]用于标记访问过的坐标。函数bfs()的参数是为程序通用性而设置的，就本问题而言，可以使用常量。

提交后得100分的[**C++语言**](http://lib.csdn.net/base/c)程序如下：

**[cpp]** [view plain](http://blog.csdn.net/tigerisland45/article/details/54954364) [copy](http://blog.csdn.net/tigerisland45/article/details/54954364)

1. /\* CCF201403-4 无线网络 \*/
3. #include <iostream>
4. #include <cstring>
5. #include <queue>
7. **using** **namespace** std;
9. **const** **int** N = 100 + 100;
11. **struct** {
12. **long** **long** x, y;
13. } coord[N+1];
15. **struct** node {
16. **long** **long** x, y;
17. **int** step;
18. };
20. **bool** visited[N+1];
22. **int** bfs(**int** n, **int** begin, **int** end, **long** **long** r)
23. {
24. // 变量初始化
25. memset(visited, **false**, **sizeof**(visited));
27. // 设置根结点
28. node start, front, v;
29. start.x = coord[begin].x;
30. start.y = coord[begin].y;
31. start.step = 0;
32. queue<node> q;
33. q.push(start);
35. // 设置根结点为已经访问过
36. visited[begin] = **true**;
38. **while**(!q.empty()) {
39. front = q.front();
40. q.pop();
42. // 到达终点则结束
43. **if**(front.x == coord[end].x && front.y == coord[end].y)
44. **return** front.step - 1;
46. // 搜索可以连接的路由器
47. **for**(**int** i=0; i<n; i++) {
48. // 访问过的坐标则跳过
49. **if**(visited[i])
50. **continue**;
52. // 判定下一个路由器的坐标是否在半径r之内, 不在半径之内则跳过，在半径之内则继续搜索
53. **if**((front.x - coord[i].x) \* (front.x - coord[i].x) + (front.y - coord[i].y) \* (front.y - coord[i].y) > r \* r)
54. **continue**;
55. **else** {
56. // 第i个路由器设为已经访问过
57. visited[i] = **true**;
59. // 计算步数，并且将第i个路由器加入队列
60. v.x = coord[i].x;
61. v.y = coord[i].y;
62. v.step = front.step + 1;
63. q.push(v);
64. }
65. }
66. }
68. **return** 0;
69. }
71. **int** main()
72. {
73. **int** n, m, k;
74. **long** **long** r;
76. // 输入数据
77. cin >> n >> m >> k >> r;
78. **for**(**int** i=0; i<n+m; i++)       // n个路由器的位置＋可以增设的m个路由器的位置
79. cin >> coord[i].x >> coord[i].y;
81. // BFS
82. **int** ans = bfs(n + m, 0, 1, r);
84. // 输出结果
85. cout << ans << endl;
87. **return** 0;
88. }
90. /\*
91. 测试数据：
93. 5 3 1 3
94. 0 0
95. 5 5
96. 0 3
97. 0 5
98. 3 5
99. 3 3
100. 4 4
101. 3 0
102. 2
104. 10 1 1 2
105. 0 0
106. 3 1
107. -2 0
108. -2 2
109. -2 4
110. -2 6
111. 0 6
112. 2 6
113. 2 4
114. 2 2
115. 2 0
116. 1
118. 10 1 1 2
119. 0 0
120. 3 1
121. -2 0
122. -2 2
123. -2 4
124. -2 6
125. 0 6
126. 2 6
127. 2 4
128. 2 2
129. 3 0
130. 8
132. 6 3 2 50000000
133. 0 0
134. 50000000 100000000
135. 100000000 100000000
136. 100000000 0
137. 100000000 50000000
138. 50000000 0
139. -100000000 50000000
140. 0 50000000
141. 0 100000000
142. 2
143. \*/

[**CCF201409-4 最优配餐（100分）**](http://blog.csdn.net/tigerisland45/article/details/54945267)

|  |  |
| --- | --- |
| 试题编号： | 201409-4 |
| 试题名称： | 最优配餐 |
| 时间限制： | 1.0s |
| 内存限制： | 256.0MB |
| 问题描述： | **问题描述**  　　栋栋最近开了一家餐饮连锁店，提供外卖服务。随着连锁店越来越多，怎么合理的给客户送餐成为了一个急需解决的问题。 　　栋栋的连锁店所在的区域可以看成是一个n×n的方格图（如下图所示），方格的格点上的位置上可能包含栋栋的分店（绿色标注）或者客户（蓝色标注），有一些格点是不能经过的（红色标注）。 　　方格图中的线表示可以行走的道路，相邻两个格点的距离为1。栋栋要送餐必须走可以行走的道路，而且不能经过红色标注的点。  　　送餐的主要成本体现在路上所花的时间，每一份餐每走一个单位的距离需要花费1块钱。每个客户的需求都可以由栋栋的任意分店配送，每个分店没有配送总量的限制。 　　现在你得到了栋栋的客户的需求，请问在最优的送餐方式下，送这些餐需要花费多大的成本。  C:\Users\XIAOQI~1\AppData\Local\Temp\Center.png  **输入格式**  　　输入的第一行包含四个整数n, m, k, d，分别表示方格图的大小、栋栋的分店数量、客户的数量，以及不能经过的点的数量。 　　接下来m行，每行两个整数xi, yi，表示栋栋的一个分店在方格图中的横坐标和纵坐标。 　　接下来k行，每行三个整数xi, yi, ci，分别表示每个客户在方格图中的横坐标、纵坐标和订餐的量。（注意，可能有多个客户在方格图中的同一个位置） 　　接下来d行，每行两个整数，分别表示每个不能经过的点的横坐标和纵坐标。  **输出格式**  　　输出一个整数，表示最优送餐方式下所需要花费的成本。  **样例输入**  10 2 3 3 1 1 8 8 1 5 1 2 3 3 6 7 2 1 2 2 2 6 8  **样例输出**  29  **评测用例规模与约定**  　　前30%的评测用例满足：1<=n <=20。 　　前60%的评测用例满足：1<=n<=100。 　　所有评测用例都满足：1<=n<=1000，1<=m, k, d<=n^2。可能有多个客户在同一个格点上。每个客户的订餐量不超过1000，每个客户所需要的餐都能被送到。 |

**问题链接：**[CCF201609试题](http://blog.csdn.net/tigerisland45/article/details/54446624)。

**问题描述**：（参照上文）。

**问题分析**：这是一个求最优问题，通常用BFS（广度优先搜索）来实现。

**程序说明**：数组visited[][]中，除了标记访问过的点之外，不可经过的点和分店也用它来标记，程序逻辑就会变得简洁。开始时统计订餐点的数量，以便用作结束条件。不同客户在同一点时，需要合计他们的订餐数量。其他都是套路。

提交后得100分的[**C++语言**](http://lib.csdn.net/base/c)程序如下：

**[cpp]** [view plain](http://blog.csdn.net/tigerisland45/article/details/54945267) [copy](http://blog.csdn.net/tigerisland45/article/details/54945267)

1. /\* CCF201409-4 最优配餐 \*/
3. #include <iostream>
4. #include <cstring>
5. #include <queue>
7. **using** **namespace** std;
9. **const** **int** N = 1000;
10. **const** **int** TRUE = 1;
12. **const** **int** DIRECTSIZE = 4;
13. **struct** direct {
14. **int** drow, dcol;
15. } direct[DIRECTSIZE] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
17. **int** buyer[N+1][N+1];
18. **int** visited[N+1][N+1];
20. **struct** node {
21. **int** row, col, step;
22. node(){}
23. node(**int** r, **int** c, **int** s){row=r, col=c, step=s;}
24. };
26. queue<node> q;
27. **int** buyercount = 0;
28. **long** **long** ans = 0;
30. **void** bfs(**int** n)
31. {
32. node front, v;
34. **while**(!q.empty()) {
35. front = q.front();
36. q.pop();
38. **for**(**int** i=0; i<DIRECTSIZE; i++) {
39. // 移动一格
40. v.row = front.row + direct[i].drow;
41. v.col = front.col + direct[i].dcol;
42. v.step = front.step + 1;
44. // 行列越界则跳过
45. **if**(v.row < 1 || v.row > n || v.col < 1 || v.col > n)
46. **continue**;
48. // 已经访问过的点不再访问
49. **if**(visited[v.row][v.col])
50. **continue**;
52. // 如果是订餐点，则计算成本并且累加
53. **if**(buyer[v.row][v.col] > 0) {
54. visited[v.row][v.col] = 1;
55. ans += buyer[v.row][v.col] \* v.step;
56. **if**(--buyercount == 0)
57. **return**;
58. }
60. // 向前搜索：标记v点为已经访问过，v点加入队列中
61. visited[v.row][v.col] = 1;
62. q.push(v);
63. }
64. }
65. }
67. **int** main()
68. {
69. **int** n, m, k, d, x, y, c;
71. // 变量初始化
72. memset(buyer, 0, **sizeof**(buyer));
73. memset(visited, 0, **sizeof**(visited));
75. // 输入数据
76. cin >> n >> m >> k >> d;
77. **for**(**int** i=1; i<=m; i++) {
78. cin >> x >> y;
79. q.push(node(x, y, 0));
80. visited[x][y] = TRUE;      // 各个分店搜索时，需要跳过
81. }
82. **for**(**int** i=1; i<=k; i++) {
83. cin >> x >> y;
84. cin >> c;
85. **if**(buyer[x][y] == 0)    // 统计客户所在地点数量：多个客户可能在同一地点
86. buyercount++;
87. buyer[x][y] += c;       // 统计某个地点的订单数量
88. }
89. **for**(**int** i=1; i<=d; i++) {
90. cin >> x >> y;
91. visited[x][y] = TRUE;
92. }
94. // BFS
95. bfs(n);
97. // 输出结果
98. cout << ans << endl;
100. **return** 0;
101. }

[**CCF201412-4 最优灌溉（80分）**](http://blog.csdn.net/tigerisland45/article/details/54896169)

|  |  |
| --- | --- |
| 试题编号： | 201412-4 |
| 试题名称： | 最优灌溉 |
| 时间限制： | 1.0s |
| 内存限制： | 256.0MB |
| 问题描述： | **问题描述**  　　雷雷承包了很多片麦田，为了灌溉这些麦田，雷雷在第一个麦田挖了一口很深的水井，所有的麦田都从这口井来引水灌溉。 　　为了灌溉，雷雷需要建立一些水渠，以连接水井和麦田，雷雷也可以利用部分麦田作为“中转站”，利用水渠连接不同的麦田，这样只要一片麦田能被灌溉，则与其连接的麦田也能被灌溉。 　　现在雷雷知道哪些麦田之间可以建设水渠和建设每个水渠所需要的费用（注意不是所有麦田之间都可以建立水渠）。请问灌溉所有麦田最少需要多少费用来修建水渠。  **输入格式**  　　输入的第一行包含两个正整数n, m，分别表示麦田的片数和雷雷可以建立的水渠的数量。麦田使用1, 2, 3, ……依次标号。 　　接下来m行，每行包含三个整数ai, bi, ci，表示第ai片麦田与第bi片麦田之间可以建立一条水渠，所需要的费用为ci。  **输出格式**  　　输出一行，包含一个整数，表示灌溉所有麦田所需要的最小费用。  **样例输入**  4 4 1 2 1 2 3 4 2 4 2 3 4 3  **样例输出**  6  **样例说明**  　　建立以下三条水渠：麦田1与麦田2、麦田2与麦田4、麦田4与麦田3。  **评测用例规模与约定**  　　前20%的评测用例满足：n≤5。 　　前40%的评测用例满足：n≤20。 　　前60%的评测用例满足：n≤100。 　　所有评测用例都满足：1≤n≤1000，1≤m≤100,000，1≤ci≤10,000。 |

**问题链接：**[CCF201412试题](http://blog.csdn.net/tigerisland45/article/details/54446794)。

**问题描述**：（参见上文）。

**问题分析**：这是一个最小生成树的为问题，解决的[**算法**](http://lib.csdn.net/base/datastructure)有Kruskal（克鲁斯卡尔）算法和Prim（普里姆） 算法。

**程序说明**：本程序使用Prim算法实现，也许是算法复杂度的问题，，时间上超时了，只得了80分。希望有人能够帮助改进一下。

有关最小生成树的问题也许使用克鲁斯卡尔算法，实现上更具有优势，只需要对所有的边进行排序后处理一遍即可。

**参考链接**：[Prim**算法**的C语言程序](http://blog.csdn.net/tigerisland45/article/details/54866951)。

提交后得80分的[**C++语言**](http://lib.csdn.net/base/c)程序如下：

**[cpp]** [view plain](http://blog.csdn.net/tigerisland45/article/details/54896169) [copy](http://blog.csdn.net/tigerisland45/article/details/54896169)

1. /\* CCF201412-4 最优灌溉 \*/
3. #include <iostream>
4. #include <cstring>
5. #include <climits>
7. **using** **namespace** std;
9. **int** main()
10. {
11. **int** n, m, ans=0;
13. // 输入数据
14. cin >> n >> m;
16. unsigned **int** visited[n+1], cost[n+1][n+1];
18. memset(visited, 0, **sizeof**(visited));
19. memset(cost, INT\_MAX, **sizeof**(cost));
21. **int** src, dest;
22. **for**(**int** i=1; i<=m; i++) {
23. cin >> src >> dest;
24. cin >> cost[src][dest];
25. cost[dest][src] = cost[src][dest];
26. }
28. // Prim算法
29. unsigned **int** min;
30. **int** next = 1, u, v;
31. visited[1]=1;
32. **while**(next < n) {
33. min = INT\_MAX;
34. **for**(**int** i=1; i<=n; i++)
35. **if**(visited[i] != 0)
36. **for**(**int** j=1; j<=n; j++)
37. **if**(cost[i][j] < min) {
38. min = cost[i][j];
39. u = i;
40. v = j;
41. }
43. **if**(visited[u]==0 || visited[v]==0) {
44. next++;
45. ans += min;
46. visited[v] = 1;
47. }
49. cost[u][v] = cost[v][u] = INT\_MAX;
50. }
52. // 输出结果
53. cout << ans << endl;
55. **return** 0;
56. }

另外一个提交后得80分的C++程序：

**[cpp]** [view plain](http://blog.csdn.net/tigerisland45/article/details/54896169) [copy](http://blog.csdn.net/tigerisland45/article/details/54896169)

1. /\* CCF201412-4 最优灌溉 \*/
3. #include <iostream>
4. #include <cstring>
5. #include <climits>
7. **using** **namespace** std;
9. **const** **int** TRUE = 1;
10. **const** **int** FALSE = 0;
11. **const** **int** N = 1000;
13. unsigned **int** cost[N+1][N+1];
14. **int** s\_set[N+1], s\_count;
15. **int** vs\_set[N+1], vs\_count;
17. **int** n, m, ans = 0;
19. // Prim算法
20. **void** prim(**int** n)
21. {
22. **int** i, j, pj;
23. unsigned **int** minval;
25. **for**(; vs\_count > 0;) {
26. minval = INT\_MAX;
27. **for**(i=1; i<=n; i++) {
28. **if**(s\_set[i])
29. **for**(j=1; j<=n; j++)
30. **if**(i!=j && vs\_set[j])
31. **if**(cost[i][j] < minval) {
32. minval = cost[i][j];
33. pj = j;
34. }
35. }
36. s\_set[pj] = TRUE;
37. s\_count++;
38. vs\_set[pj] = FALSE;
39. vs\_count--;
41. ans += minval;
42. }
43. }
45. **int** main()
46. {
47. // 变量初始化
48. memset(cost, INT\_MAX, **sizeof**(cost));
49. memset(s\_set, FALSE, **sizeof**(s\_set));
50. memset(vs\_set, TRUE, **sizeof**(vs\_set));
52. // 输入数据
53. cin >> n >> m;
54. **int** src, dest;
55. **for**(**int** i=1; i<=m; i++) {
56. cin >> src >> dest;
57. cin >> cost[src][dest];
58. cost[dest][src] = cost[src][dest];
59. }
61. // Prim算法
62. s\_set[1] = TRUE;
63. s\_count = 1;
64. vs\_set[1] = FALSE;
65. vs\_count = n - 1;
66. prim(n);
68. // 输出结果
69. cout << ans << endl;
71. **return** 0;
72. }

[**CCF201412-4 最优灌溉（解法二）（100分）**](http://blog.csdn.net/tigerisland45/article/details/54917446)

|  |  |
| --- | --- |
| 试题编号： | 201412-4 |
| 试题名称： | 最优灌溉 |
| 时间限制： | 1.0s |
| 内存限制： | 256.0MB |
| 问题描述： | **问题描述**  　　雷雷承包了很多片麦田，为了灌溉这些麦田，雷雷在第一个麦田挖了一口很深的水井，所有的麦田都从这口井来引水灌溉。 　　为了灌溉，雷雷需要建立一些水渠，以连接水井和麦田，雷雷也可以利用部分麦田作为“中转站”，利用水渠连接不同的麦田，这样只要一片麦田能被灌溉，则与其连接的麦田也能被灌溉。 　　现在雷雷知道哪些麦田之间可以建设水渠和建设每个水渠所需要的费用（注意不是所有麦田之间都可以建立水渠）。请问灌溉所有麦田最少需要多少费用来修建水渠。  **输入格式**  　　输入的第一行包含两个正整数n, m，分别表示麦田的片数和雷雷可以建立的水渠的数量。麦田使用1, 2, 3, ……依次标号。 　　接下来m行，每行包含三个整数ai, bi, ci，表示第ai片麦田与第bi片麦田之间可以建立一条水渠，所需要的费用为ci。  **输出格式**  　　输出一行，包含一个整数，表示灌溉所有麦田所需要的最小费用。  **样例输入**  4 4 1 2 1 2 3 4 2 4 2 3 4 3  **样例输出**  6  **样例说明**  　　建立以下三条水渠：麦田1与麦田2、麦田2与麦田4、麦田4与麦田3。  **评测用例规模与约定**  　　前20%的评测用例满足：n≤5。 　　前40%的评测用例满足：n≤20。 　　前60%的评测用例满足：n≤100。 　　所有评测用例都满足：1≤n≤1000，1≤m≤100,000，1≤ci≤10,000。 |

**问题链接：**[CCF201412试题](http://blog.csdn.net/tigerisland45/article/details/54446794)。

**问题描述**：（参见上文）。

**问题分析**：这是一个最小生成树的为问题，解决的[**算法**](http://lib.csdn.net/base/datastructure)有Kruskal（克鲁斯卡尔）[**算法**](http://lib.csdn.net/base/datastructure)和Prim（普里姆） 算法。

**程序说明**：本程序使用Kruskal算法实现。有关最小生成树的问题,使用克鲁斯卡尔算法更具有优势，只需要对所有的边进行排序后处理一遍即可。程序中使用了并查集，用来判定加入一条边后会不会产生循环。n个结点的图，其最小生成树应该是n-1条边，这个作为程序处理的结束条件。这个程序实现Kruskal算法部分的逻辑和代码都是否简洁易懂。程序中，图采用边列表的方式存储，按边的权从小到大顺序放在优先队列中，省去了排序。

提交后得100分的[**C++语言**](http://lib.csdn.net/base/c)程序如下：

**[cpp]** [view plain](http://blog.csdn.net/tigerisland45/article/details/54917446) [copy](http://blog.csdn.net/tigerisland45/article/details/54917446)

1. /\* CCF201412-4 最优灌溉 \*/
3. #include <iostream>
4. #include <vector>
5. #include <queue>
7. **using** **namespace** std;
9. // 并查集类
10. **class** UF {
11. **private**:
12. vector<**int**> v;
13. **public**:
14. UF(**int** n) {
15. **for**(**int** i=0; i<=n; i++)
16. v.push\_back(i);
17. }
19. **int** Find(**int** x) {
20. **for**(;;) {
21. **if**(v[x] != x)
22. x = v[x];
23. **else**
24. **return** x;
25. }
26. }
28. **bool** Union(**int** x, **int** y) {
29. x = Find(x);
30. y = Find(y);
31. **if**(x == y)
32. **return** **false**;
33. **else** {
34. v[x] = y;
35. **return** **true**;
36. }
37. }
38. };
40. **struct** edge {
41. **int** src, dest, cost;
42. **bool** operator < (**const** edge& n) **const** {
43. **return** cost > n.cost;
44. }
45. };
47. **int** main()
48. {
49. priority\_queue<edge> q;     // 优先队列，用于存储边列表
50. edge e;
52. // 输入数据
53. **int** n, m;
54. cin >> n >> m;
55. **for**(**int** i=1; i<=m; i++) {
56. cin >> e.src >> e.dest >> e.cost;
57. q.push(e);
58. }
60. // Kruskal算法
61. UF uf(n);
62. **int** ans=0, count=0;
63. **for**(;;) {
64. e = q.top();
65. q.pop();
67. **if**(uf.Find(e.src) != uf.Find(e.dest)) {
68. uf.Union(e.src, e.dest);
70. ans += e.cost;
72. **if**(++count == n -1)
73. **break**;
74. }
75. }
77. // 输出结果
78. cout << ans << endl;
80. **return** 0;
81. }
83. /\*
84. 测试数据与结果两组：
85. 6 10
86. 1 2 3
87. 1 3 1
88. 1 4 6
89. 2 3 5
90. 2 5 3
91. 3 4 5
92. 3 5 6
93. 3 6 4
94. 4 6 2
95. 5 6 6
96. 13
98. 4 4
99. 1 2 1
100. 2 3 4
101. 2 4 2
102. 3 4 3
103. 6
104. \*/

[**CCF201503-4 网络延时（100分）**](http://blog.csdn.net/tigerisland45/article/details/54881069)

|  |  |
| --- | --- |
| 试题编号： | 201503-4 |
| 试题名称： | 网络延时 |
| 时间限制： | 1.0s |
| 内存限制： | 256.0MB |
| 问题描述： | **问题描述**  　　给定一个公司的网络，由n台交换机和m台终端电脑组成，交换机与交换机、交换机与电脑之间使用网络连接。交换机按层级设置，编号为1的交换机为根交换机，层级为1。其他的交换机都连接到一台比自己上一层的交换机上，其层级为对应交换机的层级加1。所有的终端电脑都直接连接到交换机上。 　　当信息在电脑、交换机之间传递时，每一步只能通过自己传递到自己所连接的另一台电脑或交换机。请问，电脑与电脑之间传递消息、或者电脑与交换机之间传递消息、或者交换机与交换机之间传递消息最多需要多少步。  **输入格式**  　　输入的第一行包含两个整数n, m，分别表示交换机的台数和终端电脑的台数。 　　第二行包含n - 1个整数，分别表示第2、3、……、n台交换机所连接的比自己上一层的交换机的编号。第i台交换机所连接的上一层的交换机编号一定比自己的编号小。 　　第三行包含m个整数，分别表示第1、2、……、m台终端电脑所连接的交换机的编号。  **输出格式**  　　输出一个整数，表示消息传递最多需要的步数。  **样例输入**  4 2 1 1 3 2 1  **样例输出**  4  **样例说明**  　　样例的网络连接模式如下，其中圆圈表示交换机，方框表示电脑：  　　其中电脑1与交换机4之间的消息传递花费的时间最长，为4个单位时间。  C:\Users\XIAOQI~1\AppData\Local\Temp\Center.png  **样例输入**  4 4 1 2 2 3 4 4 4  **样例输出**  4  **样例说明**  　　样例的网络连接模式如下：  　　其中电脑1与电脑4之间的消息传递花费的时间最长，为4个单位时间。  C:\Users\XIAOQI~1\AppData\Local\Temp\Center.png  **评测用例规模与约定**  　　前30%的评测用例满足：n ≤ 5, m ≤ 5。 　　前50%的评测用例满足：n ≤ 20, m ≤ 20。 　　前70%的评测用例满足：n ≤ 100, m ≤ 100。 　　所有评测用例都满足：1 ≤ n ≤ 10000，1 ≤ m ≤ 10000。 |

**问题链接：**[CCF201503试题](http://blog.csdn.net/tigerisland45/article/details/54446795)。

**问题描述**：（参见上文）。

**问题分析**：这是一个树的问题，求树的直径，即在树中找出两个结点，使得这两个结点间的距离最长，这个最长距离称为直径。一般可以用两次DFS或BFS来实现，在树上任意选取1个结点s，先用DFS或BFS找到距离s距离最远的结点start，然后再从结点start开始，再次用DFS或BFS找到距离s距离最远的结点，得到结果。

**程序说明**：树用邻接结点来存储，使用STL的向量数组vector<int> tree[]来表示，tree[i]中的存储从结点i能够到达的各个结点。其他说明参见源程序。

用整数表示结点，结点号是不允许重复的。终端电脑的变化从n+1开始，依次类推。

**参考链接**：[HDU4607 Park Visit（解法二）](http://blog.csdn.net/tigerisland45/article/details/52268881)。

提交后得100分的[**C++语言**](http://lib.csdn.net/base/c)程序如下：

**[cpp]** [view plain](http://blog.csdn.net/tigerisland45/article/details/54881069) [copy](http://blog.csdn.net/tigerisland45/article/details/54881069)

1. /\* CCF201503-4 网络延时 \*/
3. #include <iostream>
4. #include <vector>
5. #include <cstring>
7. **using** **namespace** std;
9. // 深度优先搜索：计算结点now到各个结点的距离，结果放入数组d[]中
10. **void** dfs(**int** now, **int** last, **int** d[], vector<**int**> tree[])
11. {
12. **int** u, size;
13. size = tree[now].size();
14. **for**(**int** i=0; i<size; i++)
15. **if** ((u = tree[now][i]) != last) {
16. d[u] = d[now] + 1;
17. dfs(u, now, d, tree);
18. }
19. }
21. **int** main()
22. {
23. **int** n, m, t;
25. // 输入数据，构建树（邻接图）
26. cin >> n >> m;
28. vector<**int**> tree[n+m+2];
29. **int** dist[n+m+2];
31. **for**(**int** i=2; i<=n; i++) {
32. cin >> t;
34. tree[i].push\_back(t);
35. tree[t].push\_back(i);
36. }
37. **for**(**int** i=1; i<=m; i++) {
38. cin >> t;
40. tree[n+i].push\_back(t);
41. tree[t].push\_back(n+i);
42. }
44. // 求结点1到各个结点的距离：距离放在数组dist[]中，dist[i]中存放结点1到结点i的距离
45. memset(dist, 0, **sizeof**(dist));
46. dfs(1, 0, dist, tree);
48. // 找出距离结点1最远的结点start
49. **int** start = 0;
50. dist[start] = 0;
51. **for**(**int** i=1; i<n+m+2; i++)
52. **if**(dist[i] > dist[start])
53. start = i;
55. // 求start结点到各个结点的距离：距离放在数组dist[]中，dist[i]中存放结点start到结点i的距离
56. memset(dist, 0, **sizeof**(dist));
57. dfs(start, 0, dist, tree);
59. // 找出距离结点start最远的结点target
60. **int** target = 0;
61. **for** (**int** i=1; i<n+m+2; i++)
62. **if**(dist[i] > dist[target])
63. target = i;
65. // 输出结果
66. cout << dist[target] << endl;
68. **return** 0;
69. }

[**CCF201509-4 高速公路（100分）**](http://blog.csdn.net/tigerisland45/article/details/54865933)

|  |  |
| --- | --- |
| 试题编号： | 201509-4 |
| 试题名称： | 高速公路 |
| 时间限制： | 1.0s |
| 内存限制： | 256.0MB |
| 问题描述： | **问题描述**  　　某国有n个城市，为了使得城市间的交通更便利，该国国王打算在城市之间修一些高速公路，由于经费限制，国王打算第一阶段先在部分城市之间修一些单向的高速公路。 　　现在，大臣们帮国王拟了一个修高速公路的计划。看了计划后，国王发现，有些城市之间可以通过高速公路直接（不经过其他城市）或间接（经过一个或多个其他城市）到达，而有的却不能。如果城市A可以通过高速公路到达城市B，而且城市B也可以通过高速公路到达城市A，则这两个城市被称为便利城市对。 　　国王想知道，在大臣们给他的计划中，有多少个便利城市对。  **输入格式**  　　输入的第一行包含两个整数n, m，分别表示城市和单向高速公路的数量。 　　接下来m行，每行两个整数a, b，表示城市a有一条单向的高速公路连向城市b。  **输出格式**  　　输出一行，包含一个整数，表示便利城市对的数量。  **样例输入**  5 5 1 2 2 3 3 4 4 2 3 5  **样例输出**  3  **样例说明**  　　城市间的连接如图所示。有3个便利城市对，它们分别是(2, 3), (2, 4), (3, 4)，请注意(2, 3)和(3, 2)看成同一个便利城市对。  C:\Users\XIAOQI~1\AppData\Local\Temp\Center.png  **评测用例规模与约定**  　　前30%的评测用例满足1 ≤ n ≤ 100, 1 ≤ m ≤ 1000； 　　前60%的评测用例满足1 ≤ n ≤ 1000, 1 ≤ m ≤ 10000； 　　所有评测用例满足1 ≤ n ≤ 10000, 1 ≤ m ≤ 100000。 |

**问题链接：**[CCF201509试题](http://blog.csdn.net/tigerisland45/article/details/54446801)。

**问题描述**：（参见上文）。

**问题分析**：这是一个强联通图的问题，用Tarjan[**算法**](http://lib.csdn.net/base/datastructure)来解决。另外一个算法是kosaraju算法，也用于解决强联通图问题。

**程序说明**：本程序采用Tarjan算法。主函数main()中，创建图对象是参数本应该用n，但是提交后出现了运行错误，所有改成n+1。程序通过使用Tarjan算法类（参见以下链接）来实现，做了简单修改，使用变量ans来存储结果，其中增加了中间变量count。

求得强联通子图后，对于每一个强联通子图如果有k个结点，若k>1则强联通对结点的数量为k\*(k-1)/2，若k=1则为0。

**相关链接**：[Tarjan算法查找强联通组件](http://blog.csdn.net/tigerisland45/article/details/54864931)。

提交后得100分的[**C++语言**](http://lib.csdn.net/base/c)程序如下：

**[cpp]** [view plain](http://blog.csdn.net/tigerisland45/article/details/54865933) [copy](http://blog.csdn.net/tigerisland45/article/details/54865933)

1. /\* CCF201509-4 高速公路  \*/
3. #include <iostream>
4. #include <list>
5. #include <stack>
7. **using** **namespace** std;
9. **const** **int** NIL = -1;
11. **int** ans  = 0;
13. // A class that represents an directed graph
14. **class** Graph
15. {
16. **int** V;    // No. of vertices
17. list<**int**> \*adj;    // A dynamic array of adjacency lists
19. // A Recursive DFS based function used by SCC()
20. **void** SCCUtil(**int** u, **int** disc[], **int** low[],
21. stack<**int**> \*st, **bool** stackMember[]);
22. **public**:
23. Graph(**int** V);   // Constructor
24. **void** addEdge(**int** v, **int** w);   // function to add an edge to graph
25. **void** SCC();    // prints strongly connected components
26. };
28. Graph::Graph(**int** V)
29. {
30. **this**->V = V;
31. adj = **new** list<**int**>[V];
32. }
34. **void** Graph::addEdge(**int** v, **int** w)
35. {
36. adj[v].push\_back(w);
37. }
39. // A recursive function that finds and prints strongly connected
40. // components using DFS traversal
41. // u --> The vertex to be visited next
42. // disc[] --> Stores discovery times of visited vertices
43. // low[] -- >> earliest visited vertex (the vertex with minimum
44. //             discovery time) that can be reached from subtree
45. //             rooted with current vertex
46. // \*st -- >> To store all the connected ancestors (could be part
47. //           of SCC)
48. // stackMember[] --> bit/index array for faster check whether
49. //                  a node is in stack
50. **void** Graph::SCCUtil(**int** u, **int** disc[], **int** low[], stack<**int**> \*st,
51. **bool** stackMember[])
52. {
53. // A static variable is used for simplicity, we can avoid use
54. // of static variable by passing a pointer.
55. **static** **int** time = 0;
57. // Initialize discovery time and low value
58. disc[u] = low[u] = ++time;
59. st->push(u);
60. stackMember[u] = **true**;
62. // Go through all vertices adjacent to this
63. list<**int**>::iterator i;
64. **for** (i = adj[u].begin(); i != adj[u].end(); ++i)
65. {
66. **int** v = \*i;  // v is current adjacent of 'u'
68. // If v is not visited yet, then recur for it
69. **if** (disc[v] == -1)
70. {
71. SCCUtil(v, disc, low, st, stackMember);
73. // Check if the subtree rooted with 'v' has a
74. // connection to one of the ancestors of 'u'
75. // Case 1 (per above discussion on Disc and Low value)
76. low[u]  = min(low[u], low[v]);
77. }
79. // Update low value of 'u' only of 'v' is still in stack
80. // (i.e. it's a back edge, not cross edge).
81. // Case 2 (per above discussion on Disc and Low value)
82. **else** **if** (stackMember[v] == **true**)
83. low[u]  = min(low[u], disc[v]);
84. }
86. // head node found, pop the stack and print an SCC
87. **int** w = 0;  // To store stack extracted vertices
88. **int** count = 0;
89. **if** (low[u] == disc[u])
90. {
91. **while** (st->top() != u)
92. {
93. w = (**int**) st->top();
94. //            cout << w << " ";
95. count++;
96. stackMember[w] = **false**;
97. st->pop();
98. }
99. w = (**int**) st->top();
100. //        cout << w << "\n";
101. count++;
102. stackMember[w] = **false**;
103. st->pop();
104. }
105. **if**(count > 1)
106. ans += count \* (count -1) / 2;
107. }
109. // The function to do DFS traversal. It uses SCCUtil()
110. **void** Graph::SCC()
111. {
112. **int** \*disc = **new** **int**[V];
113. **int** \*low = **new** **int**[V];
114. **bool** \*stackMember = **new** **bool**[V];
115. stack<**int**> \*st = **new** stack<**int**>();
117. // Initialize disc and low, and stackMember arrays
118. **for** (**int** i = 0; i < V; i++)
119. {
120. disc[i] = NIL;
121. low[i] = NIL;
122. stackMember[i] = **false**;
123. }
125. // Call the recursive helper function to find strongly
126. // connected components in DFS tree with vertex 'i'
127. **for** (**int** i = 0; i < V; i++)
128. **if** (disc[i] == NIL)
129. SCCUtil(i, disc, low, st, stackMember);
130. }
132. **int** main()
133. {
134. **int** n, m, src, dest;
136. cin >> n >> m;
138. Graph g(n+1);
140. **for**(**int** i=1; i<=m; i++) {
141. cin >> src >> dest;
143. g.addEdge(src, dest);
144. }
146. g.SCC();
148. cout << ans << endl;
150. **return** 0;
151. }

[**CCF201512-4 送货（100分）**](http://blog.csdn.net/tigerisland45/article/details/56664423)

|  |  |
| --- | --- |
| 试题编号： | 201512-4 |
| 试题名称： | 送货 |
| 时间限制： | 1.0s |
| 内存限制： | 256.0MB |
| 问题描述： | **问题描述**  　　为了增加公司收入，F公司新开设了物流业务。由于F公司在业界的良好口碑，物流业务一开通即受到了消费者的欢迎，物流业务马上遍及了城市的每条街道。然而，F公司现在只安排了小明一个人负责所有街道的服务。 　　任务虽然繁重，但是小明有足够的信心，他拿到了城市的地图，准备研究最好的方案。城市中有n个交叉路口，m条街道连接在这些交叉路口之间，每条街道的首尾都正好连接着一个交叉路口。除开街道的首尾端点，街道不会在其他位置与其他街道相交。每个交叉路口都至少连接着一条街道，有的交叉路口可能只连接着一条或两条街道。 　　小明希望设计一个方案，从编号为1的交叉路口出发，每次必须沿街道去往街道另一端的路口，再从新的路口出发去往下一个路口，直到所有的街道都经过了正好一次。  **输入格式**  　　输入的第一行包含两个整数n, m，表示交叉路口的数量和街道的数量，交叉路口从1到n标号。 　　接下来m行，每行两个整数a, b，表示和标号为a的交叉路口和标号为b的交叉路口之间有一条街道，街道是双向的，小明可以从任意一端走向另一端。两个路口之间最多有一条街道。  **输出格式**  　　如果小明可以经过每条街道正好一次，则输出一行包含m+1个整数p1, p2, p3, ..., pm+1，表示小明经过的路口的顺序，相邻两个整数之间用一个空格分隔。如果有多种方案满足条件，则输出字典序最小的一种方案，即首先保证p1最小，p1最小的前提下再保证p2最小，依此类推。 　　如果不存在方案使得小明经过每条街道正好一次，则输出一个整数-1。  **样例输入**  4 5 1 2 1 3 1 4 2 4 3 4  **样例输出**  1 2 4 1 3 4  **样例说明**  　　城市的地图和小明的路径如下图所示。  C:\Users\XIAOQI~1\AppData\Local\Temp\Center.png  **样例输入**  4 6 1 2 1 3 1 4 2 4 3 4 2 3  **样例输出**  -1  **样例说明**  　　城市的地图如下图所示，不存在满足条件的路径。  C:\Users\XIAOQI~1\AppData\Local\Temp\Center.png  **评测用例规模与约定**  　　前30%的评测用例满足：1 ≤ n ≤ 10, n-1 ≤ m ≤ 20。 　　前50%的评测用例满足：1 ≤ n ≤ 100, n-1 ≤ m ≤ 10000。 　　所有评测用例满足：1 ≤ n ≤ 10000，n-1 ≤ m ≤ 100000。 |

**问题链接：**[CCF201512试题](http://blog.csdn.net/tigerisland45/article/details/54446853)。

**问题描述**：参见上文，求字典顺序最小的解。

**问题分析**：这个问题是一笔画问题，或者说是欧拉路径问题。所有需要判定图是否是连通图，然后再判定是否存在欧拉路径。判定是否是连通图，可以使用并查集来实现。判定是否存在欧拉路径的条件是：无向图的所有结点的出入度均为偶数，或者有2个出入度为奇数的结点。满足这个条件的图，必然能够找到欧拉路径。由于是从结点1出发，如果有2个出入度为奇数的结点，1的出入度必须为奇数。

**程序说明**：程序中，使用邻接表表示图，而且使用的是集合set。集合类set有自然排序的特征，构建好集合后不用专门排序。这样做,在用DFS寻找欧拉路径时，找到的第1个解即为字典顺序最小解。

并查集类需要是压缩的[**算法**](http://lib.csdn.net/base/datastructure)，不然时间上会超时。

其他都是套路。

提交后得100分的[**C++语言**](http://lib.csdn.net/base/c)程序如下：

**[cpp]** [view plain](http://blog.csdn.net/tigerisland45/article/details/56664423) [copy](http://blog.csdn.net/tigerisland45/article/details/56664423)

1. /\* CCF201512-4 送货 \*/
3. #include <iostream>
4. #include <cstring>
5. #include <set>
6. #include <vector>
7. #include <stack>
9. **using** **namespace** std;
11. **const** **int** N = 10000;
13. // 并查集类
14. **int** v[N+1];
15. **class** UF {
16. **private**:
17. **int** length;
18. **public**:
19. UF(**int** n) {
20. length = n;
21. **for**(**int** i=0; i<=n; i++)
22. v[i] = i;
23. }
25. // 压缩
26. **int** Find(**int** x) {
27. **if**(x == v[x])
28. **return** x;
29. **else**
30. **return** v[x] = Find(v[x]);
31. }
33. **bool** Union(**int** x, **int** y) {
34. x = Find(x);
35. y = Find(y);
36. **if**(x == y) {
37. **return** **false**;
38. } **else** {
39. v[x] = y;
40. **return** **true**;
41. }
42. }
43. };
45. set<**int**> g[N+1];          // 用邻接表存储图，用集合进行排序
46. stack<**int**> path;          // 用于保存欧拉路径
47. **bool** visited[N+1][N+1];
48. **int** nopathflag;
50. **int** n, m;
52. // 深度优先搜索
53. **void** dfs(**int** node)
54. {
55. **for**(set<**int**>::iterator it=g[node].begin(); it!=g[node].end(); it++) {
56. **if**(!visited[node][\*it]) {
57. visited[node][\*it] = **true**;
58. visited[\*it][node] = **true**;
60. dfs(\*it);
61. }
62. }
64. path.push(node);
65. }
67. **int** main()
68. {
69. **int** src, dest;
71. // 输入数据
72. cin >> n >> m;
73. UF uf(n);
74. **for**(**int** i=0; i<m; i++) {
75. cin >> src >> dest;
77. g[src].insert(dest);
78. g[dest].insert(src);
80. uf.Union(src, dest);
81. }
83. // 判断图的联通性
84. nopathflag = **false**;
85. **int** root = uf.Find(1);
86. **for**(**int** i=2; i<=n; i++)
87. **if**(uf.Find(i) != root) {
88. nopathflag = **true**;
89. **break**;
90. }
92. // 判定是否存在欧拉路径：根据出入度进行计算
93. **if**(!nopathflag) {
94. // 计算出入度
95. **int** count = 0;
96. **for**(**int** i=1; i<=n; i++)
97. **if**(g[i].size() % 2 == 1)
98. count++;
99. **if**(!(count == 0 || (count == 2 && g[1].size() % 2 == 1)))
100. nopathflag = **true**;
101. }
103. **if**(!nopathflag) {
104. // 计算路径：从结点1开始深度优先搜索
105. memset(visited, **false**, **sizeof**(visited));
106. dfs(1);
108. // 输出结果
109. **int** t;
110. **while**(!path.empty()) {
111. t = path.top();
112. path.pop();
114. cout << t << ' ';
115. }
116. cout << endl;
117. } **else**
118. // 输出结果：未找到路径
119. cout << -1 << endl;
121. **return** 0;
122. }

[**CCF201604-4 游戏（100分）**](http://blog.csdn.net/tigerisland45/article/details/54934916)

|  |  |
| --- | --- |
| 试题编号： | 201604-4 |
| 试题名称： | 游戏 |
| 时间限制： | 1.0s |
| 内存限制： | 256.0MB |
| 问题描述： | **问题描述**  　　小明在玩一个电脑游戏，游戏在一个n×m的方格图上进行，小明控制的角色开始的时候站在第一行第一列，目标是前往第n行第m列。 　　方格图上有一些方格是始终安全的，有一些在一段时间是危险的，如果小明控制的角色到达一个方格的时候方格是危险的，则小明输掉了游戏，如果小明的角色到达了第n行第m列，则小明过关。第一行第一列和第n行第m列永远都是安全的。 　　每个单位时间，小明的角色必须向上下左右四个方向相邻的方格中的一个移动一格。 　　经过很多次尝试，小明掌握了方格图的安全和危险的规律：每一个方格出现危险的时间一定是连续的。并且，小明还掌握了每个方格在哪段时间是危险的。 　　现在，小明想知道，自己最快经过几个时间单位可以达到第n行第m列过关。  **输入格式**  　　输入的第一行包含三个整数n, m, t，用一个空格分隔，表示方格图的行数n、列数m，以及方格图中有危险的方格数量。 　　接下来t行，每行4个整数r, c, a, b，表示第r行第c列的方格在第a个时刻到第b个时刻之间是危险的，包括a和b。游戏开始时的时刻为0。输入数据保证r和c不同时为1，而且当r为n时c不为m。一个方格只有一段时间是危险的（或者说不会出现两行拥有相同的r和c）。  **输出格式**  　　输出一个整数，表示小明最快经过几个时间单位可以过关。输入数据保证小明一定可以过关。  **样例输入**  3 3 3 2 1 1 1 1 3 2 10 2 2 2 10  **样例输出**  6  **样例说明**  　　第2行第1列时刻1是危险的，因此第一步必须走到第1行第2列。 　　第二步可以走到第1行第1列，第三步走到第2行第1列，后面经过第3行第1列、第3行第2列到达第3行第3列。  **评测用例规模与约定**  　　前30%的评测用例满足：0 < n, m ≤ 10，0 ≤ t < 99。 　　所有评测用例满足：0 < n, m ≤ 100，0 ≤ t < 9999，1 ≤ r ≤ n，1 ≤ c ≤ m，0 ≤ a ≤ b ≤ 100。 |

**问题链接：**[CCF201604试题](http://blog.csdn.net/tigerisland45/article/details/54447017)。

**问题描述**：（参见上文）。

**问题分析**：这是一个求最短路径的问题，即求最优问题，通常用BFS（广度优先搜索）来实现。本题也用BFS来实现，比较难以考虑到的是，需要一个三维的标志来避免重复搜索。除了行列坐标外，还需要考虑时间因素，所以是三维的。因为一些格在某个时间范围是危险的，不可进入，但是这个时间范围之外，是可以随意进入的。所以有时候需要在一些地方踱步，等过了这段时间再前行，就不能简单地限制为进入过的格不能再进入。

**程序说明**：把格的危险时间范围存储在数组visited[][][]中，使得程序逻辑变得更加简洁，同时也节省了存储。其他都是套路。

提交后得100分的[**C++语言**](http://lib.csdn.net/base/c)程序如下：

**[cpp]** [view plain](http://blog.csdn.net/tigerisland45/article/details/54934916) [copy](http://blog.csdn.net/tigerisland45/article/details/54934916)

1. /\* CCF201604-4 游戏 \*/
3. #include <iostream>
4. #include <cstring>
5. #include <queue>
7. **using** **namespace** std;
9. **const** **int** N = 100;
11. **const** **int** DIRECTSIZE = 4;
12. **struct** direct {
13. **int** drow, dcol;
14. } direct[DIRECTSIZE] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
16. **int** visited[N+1][N+1][300+1];
18. **struct** node {
19. **int** row, col, level;
20. };
22. **int** bfs(**int** n, **int** m)
23. {
24. node start, front, v;
25. start.row = 1;
26. start.col = 1;
27. start.level = 0;
29. queue<node> q;
30. q.push(start);
32. **while**(!q.empty()) {
33. front = q.front();
34. q.pop();
36. // 到达终点则结束
37. **if**(front.row == n && front.col == m)
38. **return** front.level;
40. **for**(**int** i=0; i<DIRECTSIZE; i++) {
41. // 移动一格
42. v.row = front.row + direct[i].drow;
43. v.col = front.col + direct[i].dcol;
44. v.level = front.level + 1;
46. // 行列越界则跳过
47. **if**(v.row < 1 || v.row > n || v.col < 1 || v.col > m)
48. **continue**;
50. // 已经访问过的点不再访问
51. **if**(visited[v.row][v.col][v.level])
52. **continue**;
54. // 向前搜索：标记v点为已经访问过，v点加入队列中
55. visited[v.row][v.col][v.level] = 1;
56. q.push(v);
57. }
58. }
60. **return** 0;
61. }
63. **int** main()
64. {
65. **int** n, m, t, r, c, a, b;
67. // 变量初始化
68. memset(visited, 0, **sizeof**(visited));
70. // 输入数据
71. cin >> n >> m >> t;
72. **for**(**int** i=1; i<=t; i++) {
73. cin >> r >> c >> a >> b;
75. // 设置方格危险时间，使之那些时间不可进入
76. **for**(**int** j=a; j<=b; j++)
77. visited[r][c][j] = 1;
78. }
80. // BFS搜索
81. **int** ans = bfs(n, m);
83. // 输出结果
84. cout << ans << endl;
86. **return** 0;
87. }

[**CCF201609-4 交通规划（100分）**](http://blog.csdn.net/tigerisland45/article/details/60084015)

|  |  |
| --- | --- |
| 试题编号： | 201609-4 |
| 试题名称： | 交通规划 |
| 时间限制： | 1.0s |
| 内存限制： | 256.0MB |
| 问题描述： | **问题描述**  　　G国国王来中国参观后，被中国的高速铁路深深的震撼，决定为自己的国家也建设一个高速铁路系统。 　　建设高速铁路投入非常大，为了节约建设成本，G国国王决定不新建铁路，而是将已有的铁路改造成高速铁路。现在，请你为G国国王提供一个方案，将现有的一部分铁路改造成高速铁路，使得任何两个城市间都可以通过高速铁路到达，而且从所有城市乘坐高速铁路到首都的最短路程和原来一样长。请你告诉G国国王在这些条件下最少要改造多长的铁路。  **输入格式**  　　输入的第一行包含两个整数n, m，分别表示G国城市的数量和城市间铁路的数量。所有的城市由1到n编号，首都为1号。 　　接下来m行，每行三个整数a, b, c，表示城市a和城市b之间有一条长度为c的双向铁路。这条铁路不会经过a和b以外的城市。  **输出格式**  　　输出一行，表示在满足条件的情况下最少要改造的铁路长度。  **样例输入**  4 5 1 2 4 1 3 5 2 3 2 2 4 3 3 4 2  **样例输出**  11  **评测用例规模与约定**  　　对于20%的评测用例，1 ≤ n ≤ 10，1 ≤ m ≤ 50； 　　对于50%的评测用例，1 ≤ n ≤ 100，1 ≤ m ≤ 5000； 　　对于80%的评测用例，1 ≤ n ≤ 1000，1 ≤ m ≤ 50000； 　　对于100%的评测用例，1 ≤ n ≤ 10000，1 ≤ m ≤ 100000，1 ≤ a, b ≤ n，1 ≤ c ≤ 1000。输入保证每个城市都可以通过铁路达到首都。 |

**问题链接：**[CCF201609试题](http://blog.csdn.net/tigerisland45/article/details/54447023)。

**问题描述**：参见上文。

**问题分析**：这是一个最优化的问题，也是一个单源最短路径问题，所有要用Dijkstra[**算法**](http://lib.csdn.net/base/datastructure)。题目要求在“所有城市乘坐高速铁路到首都的最短路程和原来一样长”的前提下，计算出“最少要改造多少铁路”？

**程序说明**：图的表示主要有三种形式，一是邻接表，二是邻接矩阵，三是边列表。邻接矩阵对于结点多和边少的情况都不理想。程序中用邻接表存储图，即g[]，是一种动态的存储。数组dist[]中存储单源（首都，结点1）到各个结点（城市）的最短距离。优先队列q按照边的权值从小到大排队，便于计算最短路径。

对于n个结点的城市，要连通起来，最少有n-1条道路就够了。

数组cost[i]用于存储要到达结点i，并且满足单源最短路径的条件，需要改造的铁路的长度。这是使用Dijkstra算法解决本问题需要增加的。程序中的72行就是增加的逻辑。

另外一个问题，从单源出发到达某个结点，最短路径有两条以上，并且路径长度相等时，需要选一个代价小的。例如，[**测试**](http://lib.csdn.net/base/softwaretest)实例中，结点1到4有两条路径，1-2-4和1-3-4，其距离都是7，边1-2和1-3是必选的，边2-4和3-4是可选的，由于边2-4的权为3，而边3-4的权为2，所以为了到达结点4选择小的权2。程序中，这个逻辑体现在75行。

提交后得100分的[**C++语言**](http://lib.csdn.net/base/c)程序如下：

**[cpp]** [view plain](http://blog.csdn.net/tigerisland45/article/details/60084015) [copy](http://blog.csdn.net/tigerisland45/article/details/60084015)

1. /\* CCF201609-4 交通规划 \*/
3. #include <iostream>
4. #include <vector>
5. #include <queue>
7. **using** **namespace** std;
9. //#define DEBUG
11. **const** **int** INT\_MAX2 = ((unsigned **int**)(-1) >> 1);
12. **const** **int** MAXN = 10000;
14. // 边
15. **struct** \_edge {
16. **int** v, cost;
17. \_edge(**int** v2, **int** c){v=v2; cost=c;}
18. };
20. // 结点
21. **struct** \_node {
22. **int** u, cost;
23. \_node(){}
24. \_node(**int** u2, **int** c){u=u2; cost=c;}
26. **bool** operator<(**const** \_node n) **const** {
27. **return** cost > n.cost;
28. }
29. };
31. vector<\_edge> g[MAXN+1];
32. priority\_queue<\_node> q;
33. **int** dist[MAXN+1];
34. **int** cost[MAXN+1];
35. **bool** visited[MAXN+1];
37. **void** dijkstra\_add(**int** start, **int** n)
38. {
39. **for**(**int** i=0; i<=n; i++) {
40. dist[i] = INT\_MAX2;
41. cost[i] = INT\_MAX2;
42. visited[i] = **false**;
43. }

46. dist[start] = 0;
47. cost[start] = 0;
49. q.push(\_node(start, 0));
51. \_node f;
52. **while**(!q.empty()) {
53. f = q.top();
54. q.pop();
56. **int** u = f.u;
57. **if**(!visited[u]) {
58. visited[u] = **true**;
60. **int** len = g[u].size();
61. **for**(**int** i=0; i<len; i++) {
62. **int** v2 = g[u][i].v;
64. **if**(visited[v2])
65. **continue**;
67. **int** tempcost = g[u][i].cost;
68. **int** nextdist = dist[u] + tempcost;
70. **if**(dist[v2] > nextdist) {
71. dist[v2] = nextdist;
72. cost[v2] = tempcost;                      // add code
73. q.push(\_node(v2, dist[v2]));
74. } **else** **if**(dist[v2] == nextdist)
75. cost[v2] = min(cost[v2], tempcost); // add code
76. }
77. }
78. }
79. }
81. **int** main()
82. {
83. **int** n, m, src, dest, cost2;
85. // 输入数据，构建图
86. cin >> n >> m;
87. **for**(**int** i=1; i<=m; i++) {
88. cin >> src >> dest >> cost2;
90. g[src].push\_back(\_edge(dest, cost2));
91. g[dest].push\_back(\_edge(src, cost2));
92. }
94. // 改进的Dijkstra算法
95. dijkstra\_add(1, n);
97. #ifdef DEBUG
98. cout << "dist : ";
99. **for**(**int** i=1; i<=n; i++)
100. cout << dist[i] << " ";
101. cout << endl;
103. cout << "cost : ";
104. **for**(**int** i=1; i<=n; i++)
105. cout << cost[i] << " ";
106. cout << endl;
107. #endif
109. // 统计边的权重
110. **int** ans=0;
111. **for**(**int** i=2; i<=n; i++)
112. ans += cost[i];
113. cout << ans << endl;
115. **return** 0;
116. }

**[CCF201612-4 压缩编码（100分）](http://blog.csdn.net/tigerisland45/article/details/61622152)**

试题编号：    201612-4  
试题名称：    压缩编码  
时间限制：    3.0s  
内存限制：    256.0MB  
问题描述：      
问题描述  
　　给定一段文字，已知单词a1, a2, …, an出现的频率分别t1, t2, …, tn。可以用01串给这些单词编码，即将每个单词与一个01串对应，使得任何一个单词的编码（对应的01串）不是另一个单词编码的前缀，这种编码称为前缀码。  
　　使用前缀码编码一段文字是指将这段文字中的每个单词依次对应到其编码。一段文字经过前缀编码后的长度为：  
　　L=a1的编码长度×t1+a2的编码长度×t2+…+ an的编码长度×tn。  
　　定义一个前缀编码为字典序编码，指对于1 ≤ i < n，ai的编码（对应的01串）的字典序在ai+1编码之前，即a1, a2, …, an的编码是按字典序升序排列的。  
　　例如，文字E A E C D E B C C E C B D B E中， 5个单词A、B、C、D、E出现的频率分别为1, 3, 4, 2, 5，则一种可行的编码方案是A:000, B:001, C:01, D:10, E:11，对应的编码后的01串为1100011011011001010111010011000111，对应的长度L为3×1+3×3+2×4+2×2+2×5=34。  
　　在这个例子中，如果使用哈夫曼(Huffman)编码，对应的编码方案是A:000, B:01, C:10, D:001, E:11，虽然最终文字编码后的总长度只有33，但是这个编码不满足字典序编码的性质，比如C的编码的字典序不在D的编码之前。  
　　在这个例子中，有些人可能会想的另一个字典序编码是A:000, B:001, C:010, D:011, E:1，编码后的文字长度为35。  
　　请找出一个字典序编码，使得文字经过编码后的长度L最小。在输出时，你只需要输出最小的长度L，而不需要输出具体的方案。在上面的例子中，最小的长度L为34。  
输入格式  
　　输入的第一行包含一个整数n，表示单词的数量。  
　　第二行包含n个整数，用空格分隔，分别表示a1, a2, …, an出现的频率，即t1, t2, …, tn。请注意a1, a2, …, an具体是什么单词并不影响本题的解，所以没有输入a1, a2, …, an。  
输出格式  
　　输出一个整数，表示文字经过编码后的长度L的最小值。  
样例输入  
5  
1 3 4 2 5  
样例输出  
34  
样例说明  
　　这个样例就是问题描述中的例子。如果你得到了35，说明你算得有问题，请自行检查自己的[**算法**](http://lib.csdn.net/base/datastructure)而不要怀疑是样例输出写错了。  
评测用例规模与约定  
　　对于30%的评测用例，1 ≤ n ≤ 10，1 ≤ ti ≤ 20；  
　　对于60%的评测用例，1 ≤ n ≤ 100，1 ≤ ti ≤ 100；  
　　对于100%的评测用例，1 ≤ n ≤ 1000，1 ≤ ti ≤ 10000。

**问题链接：**[CCF201612试题](http://blog.csdn.net/tigerisland45/article/details/54456551)。

**问题描述**：（参见上文）。

**问题分析**：

解决本问题，首先需要知道哈夫曼编码。参见：[哈夫曼编码\_百度百科](http://baike.baidu.com/item/%E5%93%88%E5%A4%AB%E6%9B%BC%E7%BC%96%E7%A0%81)。

这是一个编码问题，似乎可以用哈夫曼编码来解决，但是略有不同的地方在于“每个字符的编码按照字典序排列后的顺序与原先顺序一样”。

所以无法每次取出权值最小的两个节点，而只能选择相邻的节点，到底选择哪两个相邻节点，这便是[石子问题](http://http/blog.csdn.net/acdreamers/article/details/18039073)

设dp[i][j]表示第i到第j堆石子合并的最优值，sum[i][j]表示第i到第j堆石子的总数量。那么就有状态转移公式：

*1、dp[i][j]=0 (i==j)*

*2、dp[i][j]=min(dp[i][k]+dp[k][j])+sum[i][j] (i!=j)*

此时[**算法**](http://lib.csdn.net/base/datastructure)复杂为O(n^3)。

这里可以利用[平行四边形优化](http://baike.baidu.com/view/1985058.htm?fr=aladdin)降为O(n^2)：

由上面的方程式可知我们每次求dp[i][j]的关键是找到合适的k值，设p[i][j]为dp[i][j]的这个合适的k值，根据平行四边形规则有以下不等式：p[i][j-1]<=p[i][j]<=p[i+1][j]。

那么求解dp[i][i+L](L为长度)的复杂度就为：

(p[2,L+1]-p[1,L])+(p[3,L+2]-p[2,L+1])…+(p[n-L+1,n]-p[n-L,n-1])=p[n-L+1,n]-p[1,L]≤n。

复杂度为O(n)。然后L从1循环至n，总复杂度就为O(n^2)。

**程序说明**。：程序中，INT\_MAX2值是个将就的做法，并不是整数最大值。

这里给出了两个代码程序，后一种做了优化，速度快了非常多。前一个程序运行时间是秒级的，后一个程序运行时间是毫秒级的。

提交后得100分的[**C++语言**](http://lib.csdn.net/base/c)程序如下：

**[cpp]** [view plain](http://blog.csdn.net/tigerisland45/article/details/61622152) [copy](http://blog.csdn.net/tigerisland45/article/details/61622152)

1. /\* CCF201612-4 压缩编码 \*/
3. #include <iostream>
4. #include <cstring>
6. **using** **namespace** std;
8. **const** **int** INT\_MAX2 = 0x7F7F7F7F;
9. **const** **int** N = 1000;
11. **int** v[N+1];
12. **int** sum[N+1];
13. **int** dp[N+1][N+1];
15. **int** solve(**int** l, **int** r)
16. {
17. **if**(dp[l][r] == INT\_MAX2)
18. **for**(**int** i = l; i < r; i++)
19. dp[l][r] = min(dp[l][r], solve(l, i) + solve(i+1, r) + sum[r] - sum[l-1]);
21. **return** dp[l][r];
22. }
24. **int** main()
25. {
26. **int** n;
28. // 变量初始化：置为最大值
29. memset(dp, INT\_MAX2, **sizeof**(dp));
31. // 输入数据
32. cin >> n;
33. sum[0] = 0;
34. **for**(**int** i=1; i<=n; i++) {
35. cin >> v[i];
37. sum[i] = sum[i-1] + v[i];
38. dp[i][i] = 0;
39. }
41. // DP计算
42. solve(1, n);
44. // 输出结果
45. cout << dp[1][n] << endl;
47. **return** 0;
48. }

以上这个代码是由博主[YueLing's Blog](http://blog.csdn.net/cfarmerreally)提供的，这里只是做了简单修改，核心代码（函数solve()）的逻辑没有改变。

提交后得100分的[**C++语言**](http://lib.csdn.net/base/c)程序如下：

**[cpp]** [view plain](http://blog.csdn.net/tigerisland45/article/details/61622152) [copy](http://blog.csdn.net/tigerisland45/article/details/61622152)

1. /\* CCF201612-4 压缩编码 \*/
3. #include <iostream>
4. #include <cstring>
6. **using** **namespace** std;
8. **const** **int** INT\_MAX2 = 0x7F7F7F7F;
9. **const** **int** N = 1000;
11. **int** v[N+1];
12. **int** sum[N+1];
13. **int** dp[N+1][N+1];
14. **int** p[N+1][N+1];
16. **int** main()
17. {
18. **int** n;
20. // 变量初始化：置为最大值
21. memset(dp, INT\_MAX2, **sizeof**(dp));
23. // 输入数据
24. cin >> n;
25. sum[0] = 0;
26. **for**(**int** i=1; i<=n; i++) {
27. cin >> v[i];
29. sum[i] = sum[i-1] + v[i];
30. dp[i][i] = 0;
31. p[i][i] = i;
32. }
34. // DP计算
35. **for**(**int** len=2; len<=n; len++)
36. **for**(**int** i=1; i+len-1<=n; i++) {
37. **int** j = i + len - 1;
39. **for**(**int** k=p[i][j-1]; k<=p[i+1][j]; k++) {
40. **int** val = dp[i][k] + dp[k+1][j] + sum[j] - sum[i-1];
41. **if**(dp[i][j]>val) {
42. dp[i][j] = val;
43. p[i][j] = k;
44. }
45. }
46. }
48. // 输出结果
49. cout << dp[1][n] << endl;
51. **return** 0;
52. }

后一个程序，核心代码来自《[CCF 压缩编码](http://blog.csdn.net/qq_35877664/article/details/54708161)》，部分代码略做修改。

[**CCF201312-5 I’m stuck!（100分）**](http://blog.csdn.net/tigerisland45/article/details/55060929)

|  |  |
| --- | --- |
| 试题编号： | 201312-5 |
| 试题名称： | I’m stuck! |
| 时间限制： | 1.0s |
| 内存限制： | 256.0MB |
| 问题描述： | **问题描述**  　　给定一个R行C列的地图，地图的每一个方格可能是'#', '+', '-', '|', '.', 'S', 'T'七个字符中的一个，分别表示如下意思： 　　'#': 任何时候玩家都不能移动到此方格； 　　'+': 当玩家到达这一方格后，下一步可以向上下左右四个方向相邻的任意一个非'#'方格移动一格； 　　'-': 当玩家到达这一方格后，下一步可以向左右两个方向相邻的一个非'#'方格移动一格； 　　'|': 当玩家到达这一方格后，下一步可以向上下两个方向相邻的一个非'#'方格移动一格； 　　'.': 当玩家到达这一方格后，下一步只能向下移动一格。如果下面相邻的方格为'#'，则玩家不能再移动； 　　'S': 玩家的初始位置，地图中只会有一个初始位置。玩家到达这一方格后，下一步可以向上下左右四个方向相邻的任意一个非'#'方格移动一格； 　　'T': 玩家的目标位置，地图中只会有一个目标位置。玩家到达这一方格后，可以选择完成任务，也可以选择不完成任务继续移动。如果继续移动下一步可以向上下左右四个方向相邻的任意一个非'#'方格移动一格。 　　此外，玩家不能移动出地图。 　　请找出满足下面两个性质的方格个数： 　　1. 玩家可以从初始位置移动到此方格； 　　2. 玩家不可以从此方格移动到目标位置。  **输入格式**  　　输入的第一行包括两个整数R 和C，分别表示地图的行和列数。(1 ≤ R, C ≤ 50)。 　　接下来的R行每行都包含C个字符。它们表示地图的格子。地图上恰好有一个'S'和一个'T'。  **输出格式**  　　如果玩家在初始位置就已经不能到达终点了，就输出“I'm stuck!”（不含双引号）。否则的话，输出满足性质的方格的个数。  **样例输入**  5 5 --+-+ ..|#. ..|## S-+-T ####.  **样例输出**  2  **样例说明**  　　如果把满足性质的方格在地图上用'X'标记出来的话，地图如下所示： 　　--+-+ 　　..|#X 　　..|## 　　S-+-T 　　####X |

**问题链接：**[CCF201312试题](http://blog.csdn.net/tigerisland45/article/details/54429702)。

**原题链接**：[I’m stuck!](http://118.190.20.162/view.page?gpid=T1)。

**问题描述**：参见上文。

**问题分析**：这个问题可以用DFS（深度优先搜索）来解决，需要两次DFS。先从“S”点开始搜索，找出其可以到达的点。这是如果从“S”点不可以到达“T”点，则输出"I'm stuck!"，否则从“S”点可以到达点开始，逐个搜索其可到达的点，统计那些不可到达“T”点的数量即可。

**程序说明**：（略）。

提交后得100分的[**C++语言**](http://lib.csdn.net/base/c)程序如下：

**[cpp]** [view plain](http://blog.csdn.net/tigerisland45/article/details/55060929) [copy](http://blog.csdn.net/tigerisland45/article/details/55060929)

1. /\* CCF201312-5 I’m stuck! \*/
3. #include <iostream>
4. #include <cstring>
6. **using** **namespace** std;
8. **const** **int** N = 50;
9. **const** **int** DIRECTSIZE = 4;
10. **struct** \_direct {
11. **int** dr, dc;
12. } direct[DIRECTSIZE] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
14. **char** grid[N][N+1];
15. **int** visited[N][N], visited2[N][N];
17. **int** R, C;
19. // 判断坐标是否合法或可移动到
20. **inline** **bool** islegal(**int** r, **int** c)
21. {
22. **if**(0 <= r && r < R && 0 <= c && c < C && !visited[r][c] && grid[r][c] != '#')
23. **return** **true**;
24. **else**
25. **return** **false**;
26. }
28. // 深度优先搜索
29. **void** dfs(**int** r, **int** c)
30. {
31. **int** nextr, nextc;
33. visited[r][c] = 1;
35. **if**(grid[r][c] == '+' || grid[r][c] == 'S' || grid[r][c] == 'T') {
36. **for**(**int** i=0; i<DIRECTSIZE; i++) {
37. nextr = r + direct[i].dr;
38. nextc = c + direct[i].dc;
40. **if**(islegal(nextr, nextc))
41. dfs(nextr, nextc);
42. }
43. } **else** **if**(grid[r][c] == '-') {
44. **for**(**int** i=2; i<DIRECTSIZE; i++) {
45. nextr = r + direct[i].dr;
46. nextc = c + direct[i].dc;
48. **if**(islegal(nextr, nextc))
49. dfs(nextr, nextc);
50. }
51. } **else** **if**(grid[r][c] == '|') {
52. **for**(**int** i=0; i<2; i++) {
53. nextr = r + direct[i].dr;
54. nextc = c + direct[i].dc;
56. **if**(islegal(nextr, nextc))
57. dfs(nextr, nextc);
58. }
59. } **else** **if**(grid[r][c] == '.') {
60. nextr = r + direct[1].dr;
61. nextc = c + direct[1].dc;
63. **if**(islegal(nextr, nextc))
64. dfs(nextr, nextc);
65. }
66. }
68. **int** main()
69. {
70. **int** sr, sc, tr, tc;
72. // 输入数据
73. cin >> R >> C;
74. **for**(**int** i=0; i<R; i++)
75. cin >> grid[i];
77. // 找到起点和终点坐标
78. **for**(**int** i=0; i<R; i++)
79. **for**(**int** j=0; j<C; j++)
80. **if**(grid[i][j] == 'S')
81. sr = i, sc = j;
82. **else** **if**(grid[i][j] == 'T')
83. tr = i, tc = j;
85. // bfs：标记从"S"点可以到达的点
86. memset(visited, 0, **sizeof**(visited));
87. dfs(sr, sc);
88. memcpy(visited2, visited, **sizeof**(visited));
90. **if**(visited2[tr][tc]) {
91. **int** count = 0;
93. // 统计"S"点可以到达、而不可到达"T"点的数量
94. **for**(**int** i=0; i<R; i++)
95. **for**(**int** j=0; j<C; j++) {
96. **if**(visited2[i][j]) {    // "S"点可以到达的<i,j>点
97. // bfs：标记从<i,j>点开始可以到达的点，如果不能到达"T"点则计数
98. memset(visited, 0, **sizeof**(visited));
99. dfs(i, j);
101. **if**(!visited[tr][tc])
102. count++;
103. }
104. }
106. // 输出结果
107. cout << count << endl;
108. } **else**
109. // 从"S"点不可以到达"T"点
110. cout << "I'm stuck!" << endl;
112. **return** 0;
113. }