

中山大学计算机学院本科生实验报告

(2021 学年第 1 学期)

课程名称: Data structures and algorithms

任课教师: 张子臻

年级	20 级	专业 (方向)	软件工程
学号	20337270	姓名	钟海财
电话	13397996670	Email	2940599563@qq.com
开始日期	2021/12/30	完成日期	2022/1/6

1. 实验题目

- 1) All pairs shortest path
- 2) Robot

2. 实验目的

- 1) 给出图中存在的所有边及其长度, 求出任意两点间的最短路径长度, 存在则输出长度, 不存在则输出-1.
- 2) 给出网格中所有方格的燃料权重, 求机器人从起点到终点的最少燃料总数。

3. 程序设计

1) All pairs shortest path

设计思路:

由于这是求任意两点间的最短路径, 所以我选择使用 Floyd 算法。

同时由于距离全为正值, 且要求存在最短路径时输出, 不存在最短路径时输出-1, 所以我用-1 表示两点间不存在路径, 初始化条件改为任意两点间距离

为-1，自身到自身距离为 0:

```
1. int d[N+1][N+1]; //邻接距离矩阵, d[i][j]==-1 表示不存在从 i 到 j 的路径
2. memset(d, -1, sizeof(d)); //初始化为-1
3. for (int i = 1; i <= N; i++){
4.     d[i][i]=0;
5. }
```

同时路径更新条件改为（存在路径或存在更短路径则更新）:

```
1. if ((d[i][j] > d[i][k] + d[k][j] || d[i][j]==-1)
2.     && d[i][k]!=-1 && d[k][j]!=-1){ //存在路径或存在更短路径则更新
3.     d[i][j] = d[i][k] + d[k][j];
4. }
```

通过 Floyd 算法算出最短路径矩阵后，则只需通过以下输出所求最短路径:

```
1. for(int i=0; i<Q; ++i){
2.     int x, y;
3.     cin >> x >> y;
4.     cout << d[x][y] << endl;
5. }
```

代码:

```
1. #include<iostream>
2. #include<cstring>
3. using namespace std;
4. //Floyd 算法
5. int main(){
6.     int N, K, Q; //城市数量, 道路数量, 查询数量
7.     cin >> N >> K >> Q;
8.     int d[N+1][N+1];
9.     //邻接距离矩阵, d[i][j]==-1 表示不存在从 i 到 j 的路径
10.    memset(d, -1, sizeof(d)); //初始化为-1
11.    for (int i = 1; i <= N; i++){
12.        d[i][i]=0;
13.    }
14.    for(int t=0; t<K; ++t){ //读入邻接距离矩阵
15.        int i, j, k;
16.        cin >> i >> j >> k;
17.        if (k < d[i][j] || d[i][j]==-1){
18.            d[i][j] = k;
19.        }
20.    }
21.    for (int k = 1; k <= N; k++){ //Floyd 算法
22.        for (int i = 1; i <= N; i++){
```

```

23.         for (int j = 1; j <= N; j++){
24.             if ((d[i][j] > d[i][k] + d[k][j]
25.                 || d[i][j]==-1)&& d[i][k]!=-1&& d[k][j]!=-1){
26.                 //存在路径或存在更短路径则更新
27.                 d[i][j] = d[i][k] + d[k][j];
28.             }
29.         }
30.     }
31. }
32. for(int i=0;i<Q;++i){//输出所求的最短路径
33.     int x,y;
34.     cin >> x >> y;
35.     cout << d[x][y] << endl;
36. }
37. }

```

2) Robot

设计思路:

由于只需求从一个块到另一个块的最小燃料总数，即等价于求从一个点到另一个点的最短路径距离（即权重在点上），所以我采用 Dijkstra 算法：

- 1.初始化：S 只包含源点，U 包含除 V 中 S 以外的其他点；
- 2.选择：从 U 中选取一个距离源点最近的顶点 u 加入 S；
- 3.更新：修改 u 的后继顶点的最短路径长度；
- 4.重复步骤 2 和 3 直到所有顶点都包含在 S 中。

由于这里是方格的权重（即权重在点上），所以对 Dijkstra 算法进行修改：

- 1.初始化：记录起点（方格）并对起点进行更新；
- 2.选择：从未更新中选取一个距离起点（方格）最近的顶点 u（方格）记录；
- 3.更新：修改 u 的后继方格（周围的四个方格中不越界的方格）的最短路径长度；
- 4.重复步骤 2 和 3 直到 u==终点（即未更新的距离起点最近的点是终点）。

代码:

```
1. #include <iostream>
2. using namespace std;
3.
4. int dir[4][2] = {0, 1, 0, -1, 1, 0, -1, 0};
5. int row, col;
6. int cost[105][105]; //记录每个格子的燃料消耗
7. int best[105][105]; //记录从起点到该点的最小燃料消耗
8. bool visited[105][105]; //判断该方格是否更新过
9.
10. void refresh(int r, int c) { //遍历周围的方格并更新最小距离
11.     int tr, tc; //周围的方格的坐标
12.     for (int i = 0; i < 4; ++i) { //遍历周围的方格
13.         tr = r + dir[i][0];
14.         tc = c + dir[i][1];
15.         if (tr < 1 || tr > row || tc < 1 || tc > col) continue; //越界
16.         if (best[tr][tc] > best[r][c] + cost[tr][tc]) { //更新最小距离
17.             best[tr][tc] = best[r][c] + cost[tr][tc];
18.         }
19.     }
20. }
21.
22. int main() {
23.     int times;
24.     cin >> times;
25.     while (times-- > 0) {
26.         cin >> row >> col;
27.         for (int i = 1; i <= row; ++i) { //初始化
28.             for (int j = 1; j <= col; ++j) {
29.                 cin >> cost[i][j];
30.                 best[i][j] = 9999;
31.                 visited[i][j] = false;
32.             }
33.         }
34.         int x0, y0, x1, y1; //起点(x0,y0), 终点(x1,y1)
35.         cin >> x0 >> y0 >> x1 >> y1;
36.         if (x0 == x1 && y0 == y1) { //起点==终点
37.             cout << cost[x0][y0] << endl;
38.             continue;
39.         }
40.         visited[x0][y0] = true;
41.         best[x0][y0] = cost[x0][y0];
42.         refresh(x0, y0); //遍历起点周围的方格并更新最小距离
43.         while (true) { //Dijkstra 算法
44.             int inr, inc, min = 9999;
```

```

45.         for (int i = 1; i <= row; ++i) {
46.             for (int j = 1; j <= col; ++j) {
47.                 if (!visited[i][j] && min > best[i][j]) {
48.                     min = best[i][j]; //找到当前的最近的未更新的方格
49.                     inr = i;
50.                     inc = j;
51.                 }
52.             }
53.         }
54.         if (inr == x1 && inc == y1) break; //最近点是终点是退出
55.         visited[inr][inc] = true; //将该最近点标记为已经遍历过
56.         refresh(inr, inc); //对周围点的距离进行更新
57.     }
58.     cout << best[x1][y1] << endl; //输出最小燃料总数
59. }
60. }

```

4.程序运行与测试

1) All pairs shortest path

测试输入 1

```

1. 4 3 2
2. 1 2 3
3. 2 3 4
4. 2 4 3
5. 1 3
6. 2 4

```

输出 1 (通过)

```

1. 7
2. 3

```

测试输入 2

```

1. 4 4 2
2. 1 2 3
3. 1 3 4
4. 2 3 5

```

```
5. 1 4 1
6. 3 2
7. 1 4
```

输出 2 (通过)

```
1. -1
2. 1
```

测试输入 3

```
1. 4 4 2
2. 1 2 1
3. 1 2 2
4. 1 3 3
5. 1 3 2
6. 1 2
7. 1 3
```

输出 3 (通过)

```
1. 1
2. 2
```

2) Robot

测试输入 1

```
1. 3
2. 5 5
3. 1 1 5 3 2
4. 4 1 4 2 6
5. 3 1 1 3 3
6. 5 2 3 1 2
7. 2 1 1 1 1
8. 1 1 5 5
9. 5 4
10. 2 2 15 1
11. 5 1 15 1
12. 5 3 10 1
13. 5 2 1 1
14. 8 13 2 15
15. 1 1 1 4
16. 10 10
17. 1 1 1 1 1 1 1 1 1 1
18. 1 1 1 1 1 1 1 1 1 1
```

```
19. 1 1 1 1 1 1 1 1 1 1
20. 1 1 1 1 1 1 1 1 1 1
21. 1 1 1 1 1 1 1 1 1 1
22. 1 1 1 1 1 1 1 1 1 1
23. 1 1 1 1 1 1 1 1 1 1
24. 1 1 1 1 1 1 1 1 1 1
25. 1 1 1 1 1 1 1 1 1 1
26. 1 1 1 1 1 1 1 1 1 1
27. 1 1 10 10
```

输出 1（通过）

```
1. 10
2. 15
3. 19
```

测试输入 2

```
1. 7
2. 1 1
3. 1
4. 1 1 1 1
5. 10 10
6. 1 1 2 2 2 2 2 2 2 2
7. 1 1 2 2 2 2 2 2 2 2
8. 2 2 2 2 2 2 2 2 2 2
9. 2 2 2 2 2 2 2 2 2 2
10. 2 2 2 2 2 2 2 2 2 2
11. 2 2 2 2 2 2 2 2 2 2
12. 2 2 2 2 2 2 2 2 2 2
13. 2 2 2 2 2 2 2 2 2 2
14. 2 2 2 2 2 2 2 2 2 2
15. 2 2 2 2 2 2 2 2 2 2
16. 5 5 10 1
17. 10 10
18. 1 1 1 1 1 100 1 1 1 1
19. 1 1 1 1 1 100 1 100 1 1
20. 1 1 1 1 1 100 1 100 1 1
21. 1 1 1 1 1 100 1 100 1 1
22. 1 1 1 1 1 1 1 100 1 1
23. 1 1 1 1 1 1 1 1 1 1
24. 1 1 1 1 1 1 1 1 1 1
25. 1 1 1 1 1 1 1 1 1 1
26. 1 1 1 1 1 1 1 1 1 1
27. 1 1 1 1 1 1 1 1 1 1
28. 5 5 1 10
```

```

29. 4 4
30. 1 1 1 1
31. 1 1 1 1
32. 1 1 1 1
33. 1 1 1 1
34. 1 1 1 4
35. 5 5
36. 1 1 5 3 2
37. 4 1 4 2 6
38. 3 1 1 3 3
39. 5 2 3 1 2
40. 2 1 1 1 1
41. 1 1 5 5
42. 5 4
43. 2 2 15 1
44. 5 1 15 1
45. 5 3 10 1
46. 5 2 1 1
47. 8 13 2 15
48. 1 1 1 4
49. 10 10
50. 1 1 1 1 1 1 1 1 1 1
51. 1 1 1 1 1 1 1 1 1 1
52. 1 1 1 1 1 1 1 1 1 1
53. 1 1 1 1 1 1 1 1 1 1
54. 1 1 1 1 1 1 1 1 1 1
55. 1 1 1 1 1 1 1 1 1 1
56. 1 1 1 1 1 1 1 1 1 1
57. 1 1 1 1 1 1 1 1 1 1
58. 1 1 1 1 1 1 1 1 1 1
59. 1 1 1 1 1 1 1 1 1 1
60. 1 1 10 10

```

输出 2 (通过)

```

1. 1
2. 20
3. 10
4. 4
5. 10
6. 15
7. 19

```


5.实验总结与心得

Dijkstra 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。所以本次实验 2) Robot 采用该方法。

Floyd 算法是多源最短路径算法，用于计算任意两点间的最短路径。所以本次实验 1) All pairs shortest path 采用该方法。

Dijkstra 算法是基于贪心算法的思想，从起点出发逐步找到通向终点的最短距离；而 Floyd 算法是基于动态规划的思路，通过循环迭代的方法同时找出任意两个顶点间的最短距离。

附录、提交文件清单