《算法分析与设计》

课后作业

作业编号 作业7

学 号

姓 名

专 业 软件工程

学 院 计算机与人工智能学院

二0二二年三月

第7章 分支限界法

总体要求：

1. 采用分支限界法实现下述题目要求。
2. 写出求解过程中的目标函数，限界函数以及约束函数。
3. 画出样例输入时的解空间树以及搜索空间树，对应结点的变量定义以及搜索空间树上结点的值，并画出求解过程中堆的变化。
4. 分析算法的时间复杂度。
5. 用C/C++语言编程实现题目要求。
6. 第（2）（3）（4）（5）部分以Word文档提交，不需要单独提交源程序。

题目1： 小李希望开车到全国各地旅游，他发现旅途中各个城市的汽油价格也不相同，显然如果能够采取合理的加油方式，将会节省整个旅途的费用。假设汽车**开始时油箱为空**，汽车每走一个单位的距离将耗费一个单位的汽油。请帮助小李找到一种最省钱的方式完成他的整个自驾游。

输入要求：

输入的第一行包含两个整数n （1 ≤ *n* ≤ 1000）和m（0 ≤ *m* ≤ 10000），分别表示城市的数量以及道路的数量。其后的一行包含n个整数，分别表示n个城市的汽油价格。其后的m行，每一行包含三个整数u,v,d，表示城市u，v之间的距离为d（1 ≤ *d* ≤ 100）。最后一行包含三个整数**c（ 1 ≤ *c* ≤ 100）,s,e,分别表示油箱的容量，出发的城市和最后到达的城市**。

输出要求：

如果能够帮助小李找到一个最省钱从s到e的路线，则输出对应的费用，否则输出“impossible”。

样例输入：

5 5

10 10 20 12 13

0 1 9

0 2 8

1 2 1

1 3 11

2 3 7

10 0 3

样例输出：

170

1. 写出求解过程中的目标函数，限界函数以及约束函数。

目标函数：minx，即获得最少花费数目

限界函数：

ful+1<= C && vis[v][ful + 1] == 0 && dp[v][ful + 1] > p + w[v]

此限界保证所加油量不超过油箱容量，到达下一个城市所需油量满足，加油花费保证大于到达下一个城市所需花费

*ful\_cost*<=*ful*&&vis[*u*][*ful*-*ful\_cost*]==0&&dp[*u*][*ful*-*ful\_cost*]>*p*

此限界保证剩余油量满足到达下一个城市所耗油量，到达下一个城市所需油量满足，到达下一个城市所需花费大于当前花费

约束函数：

if(v == E) *//是否到达终点*

1. 画出样例输入时的解空间树以及搜索空间树，对应结点的变量定义以及搜索空间树上结点的值，并画出求解过程中堆的变化。



1. 分析算法的时间复杂度。

输入时的时间复杂度为M+N；算法主体为bfs函数，两个限界函数check1与check2时间复杂度为1；bfs函数中while与城市个数N有关，其中内含一个规模为M的for循环，所以bfs函数时间复杂度为M\*N，所以算法时间复杂度O（n）=M\*N+N+M

1. 用C/C++语言编程实现题目要求。
2. #include<stdio.h>
3. #include<string.h>
4. #include<queue>
5. #include<vector>
6. #include<algorithm>
7. using namespace std;
8. typedef pair<int,int> PII;
9. typedef struct Node
10. {
11. int vex; *//到达城市编号*
12. int fuel; *//剩余油量*
13. int cost; *//花费*
14. Node(int *a*,int *b*,int *c*)
15. {
16. vex  = *a*;
17. fuel = *b*;
18. cost = *c*;
19. }
20. friend bool operator < (Node *x*,Node *y*)
21. {
22. return *x*.cost > *y*.cost;
23. }
24. }Node;
25. int N, M; *// 城市数量与道路数量*
26. int w[10]; *// n个城市的汽油价格*
27. int dp[10][10]; *// dp[vex][fuel]表示到达vex这个城市，剩余fuel油量的时候的最少花费*
28. int vis[10][10] = {0}; *// vis[vex][fuel]=0表示能到达vex这个城市*
29. vector<PII> Graph[100]; *// 无向图*
30. int C, S, E; *//油箱的容量，出发的城市和最后到达的城市*
31. int check1(Node *T*)
32. {
33. int v = *T*.vex;
34. int ful = *T*.fuel;
35. int p = *T*.cost;
36. if(ful + 1 <= C && vis[v][ful + 1] == 0 && dp[v][ful + 1] > p + w[v]) *//油量不超过，油量可到达，加油花费大于过该城市需要的花费*
37. return 1;
38. return 0;
39. }
40. int check2(int *u*,int *ful\_cost*,int *ful*,int *p*)
41. {
42. if(*ful\_cost* <= *ful* && vis[*u*][*ful* - *ful\_cost*] == 0 && dp[*u*][*ful* - *ful\_cost*] > *p*) *//耗费油量满足，油量可到达，花费满足*
43. return 1;
44. return 0;
45. }
46. int bfs()
47. {
48. priority\_queue<Node> Q; *//优先队列，{城市，油，花费}*
49. Q.push({S,0,0}); *//初始状态，队尾插入*
50. while(!Q.empty()) *//判断，empty()空返回true*
51. {
52. Node T = Q.top(); *//当前状态*
53. Q.pop(); *//最前位置出队*
54. int v = T.vex; *//当前城市*
55. int ful = T.fuel; *//当前剩余油量*
56. int p = T.cost; *//当前花费*
57. if(v == E) *//是否到达终点*
58. return p;
59. vis[v][ful] = 1;
60. if(check1(T) == 1)
61. {
62. dp[v][ful + 1] = p + w[v]; *//花费*
63. Q.push({v, ful + 1, dp[v][ful + 1]}); *//下一状态*
64. }
65. for (unsigned int i = 0; i < Graph[v].size(); i++)
66. {
67. int u = Graph[v][i].first; *//城市*
68. int ful\_cost = Graph[v][i].second;
69. if(check2(u,ful\_cost,ful,p))
70. {
71. dp[u][ful - ful\_cost] = p;
72. Q.push({u, ful - ful\_cost, p}); *//下一状态*
73. }
74. }
75. }
76. return -1;
77. }
78. int main()
79. {
80. scanf("%d%d", &N, &M); *//城市数量与道路数量*
81. for (int i = 1; i <= N; i++)
82. scanf("%d", &w[i]); *// n个城市的汽油价格*
83. for (int i = 1; i <= M; i++)
84. {
85. int u, v, d; *//城市u，v之间的距离为d*
86. scanf("%d%d%d",&u,&v,&d);
87. u++;
88. v++;
89. Graph[u].push\_back({v,d});
90. Graph[v].push\_back({u,d});
91. }
92. scanf("%d%d%d", &C, &S, &E); *//油箱的容量，出发的城市和最后到达的城市*
93. S++;
94. E++;
95. for (int i = 0; i < 10; i++) *//初始化*
96. for (int j = 0; j < 10; j++)
97. dp[i][j] = 999; *// dp[vex][fuel]表示到达vex这个城市，剩余fuel油量的时候的最少花费*
98. dp[S][0] = 0; *//初始化*
99. int minx = bfs();
100. if(minx == -1)
101. printf("impossible\n");
102. else
103. printf("%d\n",minx);
104. system("pause");
105. return 0;
106. }

题目2： 可能有不少的同学玩过“贪吃蛇”的游戏，游戏中蛇头带动整个蛇的移动，蛇身将沿蛇头移动过的位置进行移动。如果将整条蛇分成若干个方格，则可以表示为S1,S2,S3，…，SL，其中S1为蛇头，SL为蛇尾，中间则是蛇头到蛇尾之间的部分。蛇在某个区域内移动时，如果蛇头所在位置的上下左右四个方向没有其它的物体，则蛇头S1可以朝其中任何一个方格移动，蛇身则填补前面移动后的区域，也就是S2移动到S1所在区域，S3移动到S2所在区域……，依次类推，如下图所示。



备注：左图为蛇移动前的情况。右图为蛇向下移动一个方格后的情况。蓝色填充的方格表示有障碍物。

请判断蛇是否可以从它当前所在的位置移动到出口所在的位置，如果能，则给出最少需要移动的步数。如果不能，则输出“Impossible”。

输入要求：

输入的第1行包含三个整数 n, m（l<=n, m<=20）和L（2<=L<=8），分别表示蛇所在区域的大小（行，列方格数）以及蛇的长度。**其后的L行表示从蛇头到蛇尾依次占用的方格的行列值**。紧接着的1行包含一个整数K，表示该区域内障碍物占用的方格数。之后的K行，每行包含两个整数，分别表示障碍物所占用的方格的行列值。出口所在的位置为（1，1），且出口处没有障碍物。

输出要求：

输出1行，如果蛇能够到达出口，则输出蛇从当前位置移动到出口位置最少需要移动的方格数量，否则输出“Impossible”。

样例输入：

8 8 6

6 2

5 2

5 3

4 3

4 2

3 2

5

3 1

6 1

4 6

5 6

6 4

样例输出：

12

（2）写出求解过程中的目标函数，限界函数以及约束函数。

**目标函数：**

   return d + 1;

**bfs函数返回最小步数**

**限界函数**：

bool isSnake(int *findy*, int *findx*, Node *nd*)

判断该蛇是否符合条件，是否各部分连接

map[ny][nx]

判断是否在障碍物上

ny < 1 || ny > N || nx < 1 || nx > M

判断蛇在区域内

**约束函数：**

if (ny == 1 && nx == 1) //是否到达出口

1. 画出样例输入时的解空间树以及搜索空间树，对应结点的变量定义以及搜索空间树上结点的值，并画出求解过程中堆的变化。



1. 分析算法的时间复杂度。

验证蛇身函数isSnake，时间复杂度为L-1；bfs函数时间复杂度为常数，main函数中输入时间复杂度较大，蛇输入部分为L，障碍物输入部分为K，所以总体时间复杂度为O（n）=L+K

1. 用C/C++语言编程实现题目要求。

#include <stdio.h>

#include <Windows.h>

bool map[10][10], vis[10][10][100] = {0};

struct Node

{

    int x, y, st, d; *//坐标，，步数*

};

struct Queue

{

    Node nodepool[100000];

    Node \*tnode;

    int poolIndex, front, rear; *//目录，头，尾*

    void init()

    {

        poolIndex = -1;

        front = rear = 0;

    }

    void push(int *y*, int *x*, int *st*, int *d*)

    {

        tnode = &nodepool[rear++];

        tnode->y = *y*, tnode->x = *x*, tnode->st = *st*, tnode->d = *d*;

    }

    Node pop()

    {

        Node ret = nodepool[front++];

        return ret;

    }

    bool empty()

    {

        return rear == front;

    }

} q;

int dy[] = {1, 0, -1, 0}; *//下,左,上,右*

int dx[] = {0, -1, 0, 1};

int N, M, L, K;

bool isSnake(int *findy*, int *findx*, Node *nd*)

{

    int dir[10];

    for (int i = L - 1; i > 0; --i)

    {

        dir[i] = *nd*.st & 3; *//按位与，取低两位*

*nd*.st = *nd*.st / 4;

    }

    int y = *nd*.y, x = *nd*.x;

    for (int i = 1; i < L; ++i)

    {

        y += dy[dir[i]];

        x += dx[dir[i]];

        if (y == *findy* && x == *findx*)

            return true;

    }

    return false;

}

int bfs(int *hy*,int *hx*,int *hst*)

{

    if (*hy* == 1 && *hx* == 1) *//头在出口*

        return 0;

    q.init();

    q.push(*hy*, *hx*, *hst*, 0);

    vis[*hy*][*hx*][*hst*] = true;

    while (!q.empty())

    {

        Node in = q.pop();

        int y = in.y, x = in.x, st = in.st, d = in.d;

        for (int i = 0; i < 4; ++i)

        {

            int ny = y + dy[i], nx = x + dx[i];

            if (ny == 1 && nx == 1) *//是否到达出口*

                return d + 1;

            if (ny < 1 || ny > N || nx < 1 || nx > M || map[ny][nx] || isSnake(ny, nx, in))

                continue;

            int nst = (st / 4) + (((i + 2) % 4) << (2 \* (L - 2)));

            if (vis[ny][nx][nst])

                continue;

            printf("%d %d %d %d\n", ny, nx, nst, d + 1);

            q.push(ny, nx, nst, d + 1);

            vis[ny][nx][nst] = true;

        }

    }

    return -1;

}

int main()

{

    int heady, headx, headst = 0;

    while (scanf("%d%d%d", &N, &M, &L) != EOF && N)

    {

        scanf("%d%d", &heady, &headx); *//蛇头*

        int ty, tx, prey = heady, prex = headx;

        for (int i = 1; i < L; ++i) *//蛇身长度L-1*

        {

            scanf("%d%d", &ty, &tx); *//蛇身*

            for (int i = 0; i < 4; ++i)

                if (prey + dy[i] == ty && prex + dx[i] == tx) *//该块蛇身满足与之前相连*

                {

                    headst = headst \* 4 + i; *//最终742*

                    break;

                }

            prey = ty, prex = tx;

        }

        scanf("%d", &K);

        int block\_x, block\_y; *// 障碍物*

        for (block\_x = 0; block\_x < 10; ++block\_x)

            for (block\_y = 0; block\_y < 10; ++block\_y)

                map[block\_x][block\_y] = false;

        for (int i = 0; i < K; ++i)

        {

            scanf("%d%d", &block\_x, &block\_y);

            map[block\_x][block\_y] = true;

        }

        printf("%d\n", bfs(heady, headx, headst));

        break;

    }

    system("pause");

    return 0;

}

评分标准：（总分100分，每题按100分评分）

1. 采用分支限界法实现。（10分）
2. 算法求解过程中的目标函数，限界函数以及约束函数：（20分）
3. 目标函数描述准确。（10分）
4. 限界函数和约束条件描述准确。（10分）
5. 样例输入时的解空间树以及搜索空间树，结点对应变量的定义以及搜索空间树上结点的值，求解过程中堆的变化过程。（30分）
6. 结点对应的变量定义准确。（5分）
7. 解空间树结构正确。（5分）
8. 搜索空间树正确，对应结点的值正确。（10分）
9. 求解过程中堆的结构以及对应结点的值正确。（10分）
10. 算法时间复杂度分析要求（10分）
11. 算法时间复杂度分析方法及过程正确。（5分）
12. 算法时间复杂度分析结果正确。（5分）
13. C/C++程序要求（30分）
14. 程序编译、运行正确无误。（5分）
15. 程序书写规范，变量、函数定义符合规范。（5分）
16. 程序与算法求解过程一致，且符合题目要求（20分）

提交截止时间：2022年6月26日20：00，每延后一天扣20分。

提交方式：电子稿，命名规则：学号\_姓名\_作业7.doc(或docx)

邮件发送给：hyhuang@home.swjtu.edu.cn