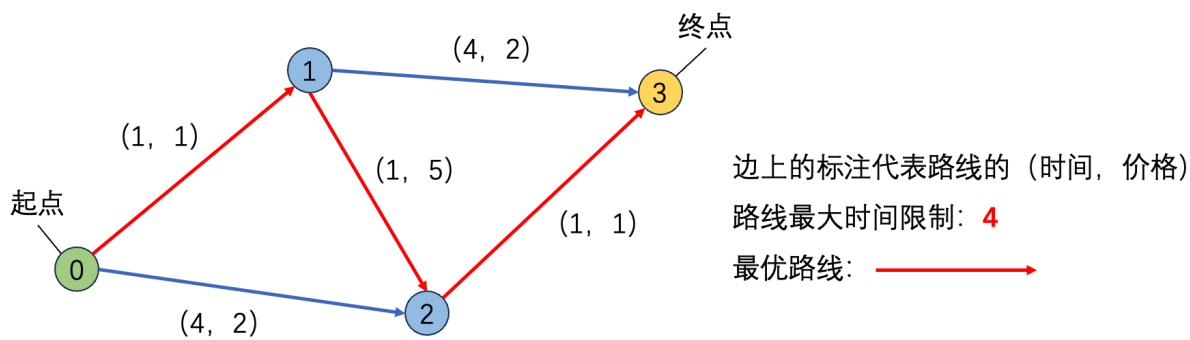


OJ9: 小明的火车旅行计划

Yanxu Chen, December 25th, 2023

【Description】

小明计划乘坐火车去远方的城市旅游。铁路系统可以被抽象为一个有向图，其中每个节点代表一个城市的火车站，边表示不同城市之间的火车线路。每条边都有两个权重，分别表示乘坐该线路所需的时间和费用。小明需要在一定的时间之内到达目的地，并且希望尽量减少花费。请问，你能帮助小明计算在满足时间要求的情况下，到达目的地所需的最低费用是多少吗？



【Input】

输入共M+2行

第1行：图的节点数、边数（N，M）

第2到M+1行：代表N条路径的信息，每行包括4个整数，分别代表路线起点、路线终点、路线时间、路线价格（其中时间和价格为正整数）

第M+2行：包括3个整数，分别代表小明的起点、目的地、路线最大时间限制

【Output】

一个整数，代表满足时间条件的路线中价格最小路线的价格。如果没有符合要求的路线，请输出-1。

【Example】

Input:

```
4 5
0 1 1 1
0 2 4 2
1 2 1 5
1 3 4 2
2 3 1 1
0 3 4
```

Output:

7

【Hints】

图节点数 $N < 2^{16}$

图边数 $M < 2^{20}$

每条边对应的时间和费用 $< 2^{16}$ ，但最终的输出可能大于这个数

【Restrictions】

Time: 1000ms

Memory: 20000KB

【Ideas】

1. 可以考虑使用动态规划。（未成功）
2. 考虑使用蒙特卡洛方法。（未成功）
3. 递归算法。

【Code】

动态规划：（未成功，最后两个测试点超时）

```
1  #include <cstdio>
2  #include <vector>
3  #include <cstdlib>
4  // #include <algorithm>
5  // #include <cmath>
6
7  struct Node
8  {
9      int Start;
10     int Time;
11     int Price;
12 };
13
14 int main()
15 {
16     int N, M; // N个节点, M条边
17     scanf("%d%d", &N, &M);
18     int START, END, TIME; // 小明的起点和终点, 时间限制
19     std::vector<std::vector<Node>> graph(N);
20     for (int i = 0; i < M; i++)
21     {
22         int start, end, time, price;
23         scanf("%d%d%d%d", &start, &end, &time, &price);
24         graph[end].push_back({start, time, price});
```

```

25     }
26     scanf("%d%d%d", &START, &END, &TIME);
27
28     // std::vector<std::vector<int>> dp(N, std::vector<int>(TIME + 1, -1));
29     // int dp[N][TIME + 1];
30     // for (int i = 0; i < N; i++)
31     // {
32     //     for (int j = 0; j <= TIME; j++)
33     //     {
34     //         dp[i][j] = -1;
35     //     }
36     // }
37     int **dp = (int **)malloc(N * sizeof(int *));
38     for (int i = 0; i < N; ++i)
39     {
40         dp[i] = (int *)malloc((TIME + 1) * sizeof(int));
41     }
42
43     // 初始化数组
44     for (int i = 0; i < N; i++)
45     {
46         for (int j = 0; j <= TIME; j++)
47         {
48             dp[i][j] = -1;
49         }
50     }
51
52     // dp[end][time]表示在时间不超过time的前提下，到达节点end的最小花费
53     // dp[i][t]=min(dp[i][t],min(dp[k][t-time[i][k]]+cost[i][k])), k是能到达i的
    所有点
54     dp[START][0] = 0;
55     for (int t = 1; t <= TIME; t++)
56     {
57         for (int i = 0; i < N; i++)
58         {
59             dp[i][t] = dp[i][t - 1];
60             for (int j = 0; j < graph[i].size(); j++)
61             {
62                 if (t >= graph[i][j].Time && dp[graph[i][j].Start][t -
graph[i][j].Time] != -1)
63                 {
64                     if (dp[i][t] == -1)
65                     {
66                         dp[i][t] = dp[graph[i][j].Start][t - graph[i]
[j].Time] + graph[i][j].Price;
67                     }
68                     else if (dp[i][t] > dp[graph[i][j].Start][t - graph[i]
[j].Time] + graph[i][j].Price)
69                         dp[i][t] = dp[graph[i][j].Start][t - graph[i]
[j].Time] + graph[i][j].Price;
70                 }
71             }
72         }
73     }
74     // for (int i = 0; i < N; i++)
75     // {

```

```

76     //     for (int j = 0; j <= TIME; j++)
77     //     {
78     //         printf("%d ", dp[i][j]);
79     //     }
80     //     printf("\n");
81     // }
82     printf("%d", dp[END][TIME]);
83     return 0;
84 }
85

```

DFS: (未成功, 几乎所有点超时)

```

1  #include <stdio>
2  #include <vector>
3  // #include <stdlib>
4  // #include <algorithm>
5  // #include <cmath>
6
7  struct Node
8  {
9      int End;
10     int Time;
11     int Price;
12 };
13
14 int main()
15 {
16     int N, M; // N个节点, M条边
17     scanf("%d%d", &N, &M);
18     int START, END, TIME; // 小明的起点和终点, 时间限制
19     std::vector<std::vector<Node>> graph(N);
20     for (int i = 0; i < M; i++)
21     {
22         int start, end, time, price;
23         scanf("%d%d%d%d", &start, &end, &time, &price);
24         graph[start].push_back({end, time, price});
25     }
26     scanf("%d%d%d", &START, &END, &TIME);
27
28     int min_price = 0x7fffffff; // 最小价格
29     std::vector<int> stk; // 存放节点的栈
30     std::vector<int> path; // 存放当前路径
31     std::vector<bool> visited(N, false); // 是否被访问过
32     std::vector<int> ready(N, 0); // 这一次该访问该节点的哪个邻接点
33
34     stk.push_back(START);
35     visited[START] = true;
36     while (!stk.empty())
37     {
38         // 当前节点的所有邻接点已经全部遍历完
39         if (ready[stk.back()] >= graph[stk.back()].size())
40         {

```

```

41         visited[stk.back()] = false;
42         ready[stk.back()] = 0;
43         stk.pop_back();
44         path.pop_back(); // 把终点从栈和路径中弹出，继续寻找
45     }
46     else
47     {
48         for (int i = ready[stk.back()]; i < graph[stk.back()].size();
i++)
49         {
50             ready[stk.back()] = i + 1;
51             if (visited[graph[stk.back()][i].End] == false)
52             {
53                 visited[graph[stk.back()][i].End] = true;
54                 stk.push_back(graph[stk.back()][i].End);
55                 path.push_back(i);
56                 break;
57             }
58         }
59         if (stk.back() == END)
60         {
61             // 找到了一条路径，计算总时间和总费用
62             int cur_time = 0, cur_price = 0;
63             int the_node = START;
64             for (int i = 0; i < path.size(); i++)
65             {
66                 // printf("%d ", the_node);
67                 cur_time += graph[the_node][path[i]].Time;
68                 if (cur_time > TIME)
69                 {
70                     break; // 已经超出了时间限制，提前终止循环
71                 }
72                 cur_price += graph[the_node][path[i]].Price;
73                 the_node = graph[the_node][path[i]].End;
74                 // printf("%d ", the_node);
75             }
76             // printf("%d %d\n", cur_time, cur_price);
77             if (cur_time <= TIME && cur_price < min_price)
78             {
79                 min_price = cur_price;
80             }
81             visited[stk.back()] = false;
82             ready[stk.back()] = 0;
83             stk.pop_back();
84             path.pop_back(); // 把终点从栈和路径中弹出，继续寻找
85         }
86     }
87 }
88
89 // 所有路径全部超时，输出-1
90 if (min_price == 0x7fffffff)
91 {
92     printf("%d", -1);
93 }
94 else
95 {

```

```

96     printf("%d", min_price);
97 }
98 return 0;
99 }
100

```

复杂版蒙特卡洛法，未成功（6、7、9、10难通过）

```

1  #include <stdio>
2  #include <vector>
3  #include <stdlib>
4  #include <ctime>
5  // #include <algorithm>
6  // #include <cmath>
7
8  struct Node
9  {
10     int End;
11     int Time;
12     int Price;
13 };
14
15 int main()
16 {
17     int N, M; // N个节点, M条边
18     scanf("%d%d", &N, &M);
19     int START, END, TIME; // 小明的起点和终点, 时间限制
20     std::vector<std::vector<Node>> graph(N);
21     for (int i = 0; i < M; i++)
22     {
23         int start, end, time, price;
24         scanf("%d%d%d", &start, &end, &time, &price);
25         graph[start].push_back({end, time, price});
26     }
27     scanf("%d%d%d", &START, &END, &TIME);
28
29     int min_price = 0x7fffffff;
30     int repeat = 720000; // 随机的重复次数
31
32     srand(static_cast<unsigned int>(time(NULL)));
33     for (int r = 0; r < repeat; r++)
34     {
35         // srand(static_cast<unsigned int>(time(NULL) + r));
36         std::vector<bool> visited(N, false);
37         visited[START] = true;
38         int cur_time = 0, cur_price = 0;
39         int current = START;
40         while (current != END)
41         {
42             std::vector<int> temp;
43             for (int i = 0; i < graph[current].size(); i++)
44             {
45                 if (visited[graph[current][i].End] == false)
46                 {
47                     temp.push_back(i);

```

```

48         }
49     }
50     if (temp.empty())
51     {
52         break;
53     }
54     int random = rand() % temp.size();
55     visited[graph[current][temp[random]].End] = true;
56     cur_time += graph[current][temp[random]].Time;
57     if (cur_time > TIME)
58     {
59         break;
60     }
61     cur_price += graph[current][temp[random]].Price;
62     current = graph[current][temp[random]].End;
63 }
64 if (current == END && cur_time <= TIME && cur_price < min_price)
65 {
66     min_price = cur_price;
67 }
68 }
69
70 // 所有路径全部超时，输出-1
71 if (min_price == 0x7fffffff)
72 {
73     printf("%d", -1);
74 }
75 else
76 {
77     printf("%d", min_price);
78 }
79 return 0;
80 }
81

```

简单版蒙特卡洛，未成功（6、7、9、10未通过）

```

1  #include <stdio>
2  #include <vector>
3  #include <stdlib>
4  #include <ctime>
5  // #include <algorithm>
6  // #include <cmath>
7
8  struct Node
9  {
10     int End;
11     int Time;
12     int Price;
13 };
14
15 int main()
16 {
17     int N, M; // N个节点，M条边
18     scanf("%d%d", &N, &M);

```

```

19     int START, END, TIME; // 小明的起点和终点, 时间限制
20     std::vector<std::vector<Node>> graph(N);
21     for (int i = 0; i < M; i++)
22     {
23         int start, end, time, price;
24         scanf("%d%d%d", &start, &end, &time, &price);
25         graph[start].push_back({end, time, price});
26     }
27     scanf("%d%d%d", &START, &END, &TIME);
28
29     long long min_price = 0x7fffffffffffffff;
30     int repeat = 1600000; // 随机的重复次数
31
32     srand(static_cast<unsigned int>(time(NULL)));
33     for (int r = 0; r < repeat; r++)
34     {
35         // srand(static_cast<unsigned int>(time(NULL) + r));
36         std::vector<bool> visited(N, false);
37         visited[START] = true;
38         long long cur_time = 0, cur_price = 0;
39         int current = START;
40         while (current != END)
41         {
42             if (graph[current].size() > 1)
43             {
44                 int random = rand() % graph[current].size();
45                 cur_time += graph[current][random].Time;
46                 if (cur_time > TIME)
47                 {
48                     break;
49                 }
50                 cur_price += graph[current][random].Price;
51                 current = graph[current][random].End;
52             }
53             else
54             {
55                 break;
56             }
57             // std::vector<int> temp;
58             // for (int i = 0; i < graph[current].size(); i++)
59             // {
60             //     if (visited[graph[current][i].End] == false)
61             //     {
62             //         temp.push_back(i);
63             //     }
64             // }
65             // if (temp.empty())
66             // {
67             //     break;
68             // }
69             // int random = rand() % temp.size();
70             // visited[graph[current][temp[random]].End] = true;
71             // cur_time += graph[current][temp[random]].Time;
72             // if (cur_time > TIME)
73             // {
74             //     break;

```



```

75         // }
76         // cur_price += graph[current][temp[random]].Price;
77         // current = graph[current][temp[random]].End;
78     }
79     if (current == END && cur_time <= TIME && cur_price < min_price)
80     {
81         min_price = cur_price;
82     }
83 }
84
85 // 所有路径全部超时，输出-1
86 if (min_price == 0x7fffffffffffffff)
87 {
88     printf("%d", -1);
89 }
90 else
91 {
92     printf("%d", min_price);
93 }
94 return 0;
95 }
96

```

递归算法，成功，时间性能较好：（从终点往起点找能通过，从起点往终点找10会超时）

```

1  #include <cstdio>
2  #include <vector>
3  // #include <cstdlib>
4  // #include <algorithm>
5  // #include <cmath>
6
7  struct Node
8  {
9      int Start;
10     int Time;
11     int Cost;
12 };
13
14 int searchPath(int current, int START, int cur_time, int cur_cost, int TIME,
std::vector<std::vector<Node>> &graph, std::vector<bool> &visited)
15 {
16     // 在当前节点处，在时间限制下，能够到达终点的最低价格。如果不能到达终点或者时间超出限制，返回无穷大
17     if (current == START && cur_time <= TIME)
18     {
19         return cur_cost;
20     }
21     if (graph[current].empty())
22     {
23         return 0x7fffffff;
24     }
25     int flag = 0;
26     int min_cost = 0x7fffffff;
27     for (const Node &neighbor : graph[current])
28     {

```

```

29         if (visited[neighbor.Start] == false)
30         {
31             if (cur_time + neighbor.Time > TIME)
32             {
33                 continue; // 已经超时，提前退出
34             }
35             else
36             {
37                 visited[neighbor.Start] = true;
38                 int cost = searchPath(neighbor.Start, START, cur_time +
neighbor.Time, cur_cost + neighbor.Cost, TIME, graph, visited);
39                 if (cost < min_cost)
40                 {
41                     min_cost = cost;
42                 }
43                 visited[neighbor.Start] = false;
44                 flag = 1;
45             }
46         }
47     }
48     if (flag == 0)
49     {
50         return 0x7fffffff; // 没有可以访问的节点
51     }
52     return min_cost;
53 }
54
55 int main()
56 {
57     int N, M; // N个节点, M条边
58     scanf("%d%d", &N, &M);
59     int START, END, TIME; // 小明的起点和终点, 时间限制
60     std::vector<std::vector<Node>> graph(N);
61     for (int i = 0; i < M; i++)
62     {
63         int start, end, time, cost;
64         scanf("%d%d%d%d", &start, &end, &time, &cost);
65         graph[end].push_back({start, time, cost});
66     }
67     scanf("%d%d%d", &START, &END, &TIME);
68
69     int cur_time = 0, cur_cost = 0;
70     std::vector<bool> visited(N, false);
71     int min_cost = searchPath(END, START, cur_time, cur_cost, TIME, graph,
visited);
72
73     if (min_cost == 0x7fffffff)
74     {
75         printf("%d", -1);
76     }
77     else
78     {
79         printf("%d", min_cost);
80     }
81     return 0;
82 }

```

最后附上通过照片：

运行结果			分数	100.00
#	状态	时间	内存	
1	Accepted	0 ms	920 KB	
2	Accepted	0 ms	916 KB	
3	Accepted	0 ms	936 KB	
4	Accepted	0 ms	996 KB	
5	Accepted	4 ms	964 KB	
6	Accepted	4 ms	1184 KB	
7	Accepted	20 ms	1804 KB	
8	Accepted	16 ms	1844 KB	
9	Accepted	16 ms	1844 KB	
10	Accepted	64 ms	4748 KB	