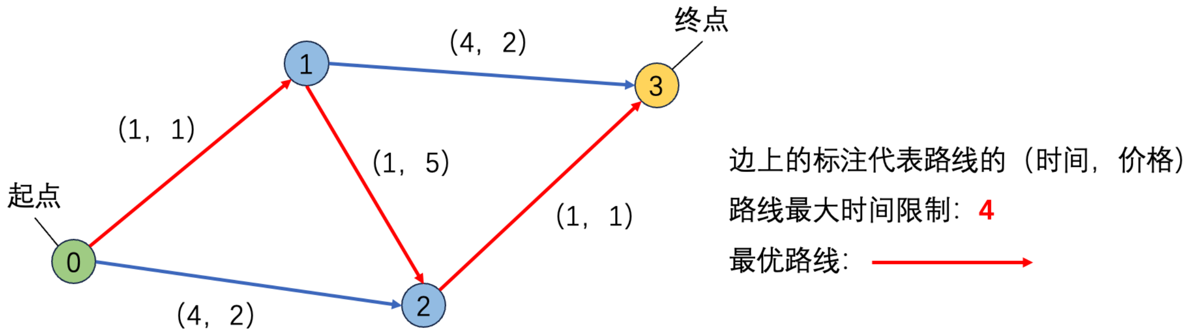
OJ9：小明的火车旅行计划

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）

```
#include <cstdio>
#include <vector>
#include <cstdlib>
// #include <algorithm>
// #include <cmath>

struct Node
{
    int Start;
    int Time;
    int Price;
};

int main()
{
    int N, M; // N个节点，M条边
    scanf("%d%d", &N, &M);
    int START, END, TIME; // 小明的起点和终点，时间限制
    std::vector<std::vector<Node>> graph(N);
    for (int i = 0; i < M; i++)
    {
        int start, end, time, price;
        scanf("%d%d%d%d", &start, &end, &time, &price);
        graph[end].push_back({start, time, price});
    }
    scanf("%d%d%d", &START, &END, &TIME);

    // std::vector<std::vector<int>> dp(N, std::vector<int>(TIME + 1, -1));
    // int dp[N][TIME + 1];
    // for (int i = 0; i < N; i++)
    // {
    //     for (int j = 0; j <= TIME; j++)
    //     {
    //         dp[i][j] = -1;
    //     }
    // }
    int **dp = (int **)malloc(N * sizeof(int *));
    for (int i = 0; i < N; ++i)
    {
        dp[i] = (int *)malloc((TIME + 1) * sizeof(int));
    }

    // 初始化数组
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j <= TIME; j++)
        {
            dp[i][j] = -1;
        }
    }

    // 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
    // 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
    dp[START][0] = 0;
    for (int t = 1; t <= TIME; t++)
    {
        for (int i = 0; i < N; i++)
        {
            dp[i][t] = dp[i][t - 1];
            for (int j = 0; j < graph[i].size(); j++)
            {
                if (t >= graph[i][j].Time && dp[graph[i][j].Start][t - graph[i][j].Time] != -1)
                {
                    if (dp[i][t] == -1)
                    {
                        dp[i][t] = dp[graph[i][j].Start][t - graph[i][j].Time] + graph[i][j].Price;
                    }
                    else if (dp[i][t] > dp[graph[i][j].Start][t - graph[i][j].Time] + graph[i][j].Price)
                        dp[i][t] = dp[graph[i][j].Start][t - graph[i][j].Time] + graph[i][j].Price;
                }
            }
        }
    }
    // for (int i = 0; i < N; i++)
    // {
```

```
    //     for (int j = 0; j <= TIME; j++)
    //     {
    //         printf("%d ", dp[i][j]);
    //     }
    //     printf("\n");
    // }
    printf("%d", dp[END][TIME]);
    return 0;
}

```

DFS：（未成功，几乎所有点超时）

```
#include <cstdio>
#include <vector>
// #include <cstdlib>
// #include <algorithm>
// #include <cmath>

struct Node
{
    int End;
    int Time;
    int Price;
};

int main()
{
    int N, M; // N个节点，M条边
    scanf("%d%d", &N, &M);
    int START, END, TIME; // 小明的起点和终点，时间限制
    std::vector<std::vector<Node>> graph(N);
    for (int i = 0; i < M; i++)
    {
        int start, end, time, price;
        scanf("%d%d%d%d", &start, &end, &time, &price);
        graph[start].push_back({end, time, price});
    }
    scanf("%d%d%d", &START, &END, &TIME);

    int min_price = 0x7fffffff;          // 最小价格
    std::vector<int> stk;                // 存放节点的栈
    std::vector<int> path;               // 存放当前路径
    std::vector<bool> visited(N, false); // 是否被访问过
    std::vector<int> ready(N, 0);        // 这一次该访问该节点的哪个邻接点

    stk.push_back(START);
    visited[START] = true;
    while (!stk.empty())
    {
        // 当前节点的所有邻接点已经全部遍历完
        if (ready[stk.back()] >= graph[stk.back()].size())
        {
            visited[stk.back()] = false;
            ready[stk.back()] = 0;
            stk.pop_back();
            path.pop_back(); // 把终点从栈和路径中弹出，继续寻找
        }
        else
        {
            for (int i = ready[stk.back()]; i < graph[stk.back()].size(); i++)
            {
                ready[stk.back()] = i + 1;
                if (visited[graph[stk.back()][i].End] == false)
                {
                    visited[graph[stk.back()][i].End] = true;
                    stk.push_back(graph[stk.back()][i].End);
                    path.push_back(i);
                    break;
                }
            }
            if (stk.back() == END)
            {
                // 找到了一条路径，计算总时间和总费用
                int cur_time = 0, cur_price = 0;
                int the_node = START;
                for (int i = 0; i < path.size(); i++)
                {
                    // printf("%d ", the_node);
                    cur_time += graph[the_node][path[i]].Time;
                    if (cur_time > TIME)
                    {
                        break; // 已经超出了时间限制，提前终止循环
                    }
                    cur_price += graph[the_node][path[i]].Price;
                    the_node = graph[the_node][path[i]].End;
                    // printf("%d ", the_node);
                }
                // printf("%d %d\n", cur_time, cur_price);
                if (cur_time <= TIME && cur_price < min_price)
                {
                    min_price = cur_price;
                }
                visited[stk.back()] = false;
                ready[stk.back()] = 0;
                stk.pop_back();
                path.pop_back(); // 把终点从栈和路径中弹出，继续寻找
            }
        }
    }

    // 所有路径全部超时，输出-1
    if (min_price == 0x7fffffff)
    {
        printf("%d", -1);
    }
    else
    {
        printf("%d", min_price);
    }
    return 0;
}

```

复杂版蒙特卡洛法，未成功（6、7、9、10难通过）

```
#include <cstdio>
#include <vector>
#include <cstdlib>
#include <ctime>
// #include <algorithm>
// #include <cmath>

struct Node
{
    int End;
    int Time;
    int Price;
};

int main()
{
    int N, M; // N个节点，M条边
    scanf("%d%d", &N, &M);
    int START, END, TIME; // 小明的起点和终点，时间限制
    std::vector<std::vector<Node>> graph(N);
    for (int i = 0; i < M; i++)
    {
        int start, end, time, price;
        scanf("%d%d%d%d", &start, &end, &time, &price);
        graph[start].push_back({end, time, price});
    }
    scanf("%d%d%d", &START, &END, &TIME);

    int min_price = 0x7fffffff;
    int repeat = 720000; // 随机的重复次数

    srand(static_cast<unsigned int>(time(NULL)));
    for (int r = 0; r < repeat; r++)
    {
        // srand(static_cast<unsigned int>(time(NULL) + r));
        std::vector<bool> visited(N, false);
        visited[START] = true;
        int cur_time = 0, cur_price = 0;
        int current = START;
        while (current != END)
        {
            std::vector<int> temp;
            for (int i = 0; i < graph[current].size(); i++)
            {
                if (visited[graph[current][i].End] == false)
                {
                    temp.push_back(i);
                }
            }
            if (temp.empty())
            {
                break;
            }
            int random = rand() % temp.size();
            visited[graph[current][temp[random]].End] = true;
            cur_time += graph[current][temp[random]].Time;
            if (cur_time > TIME)
            {
                break;
            }
            cur_price += graph[current][temp[random]].Price;
            current = graph[current][temp[random]].End;
        }
        if (current == END && cur_time <= TIME && cur_price < min_price)
        {
            min_price = cur_price;
        }
    }

    // 所有路径全部超时，输出-1
    if (min_price == 0x7fffffff)
    {
        printf("%d", -1);
    }
    else
    {
        printf("%d", min_price);
    }
    return 0;
}

```

简单版蒙特卡洛，未成功（6、7、9、10未通过）

```
#include <cstdio>
#include <vector>
#include <cstdlib>
#include <ctime>
// #include <algorithm>
// #include <cmath>

struct Node
{
    int End;
    int Time;
    int Price;
};

int main()
{
    int N, M; // N个节点，M条边
    scanf("%d%d", &N, &M);
    int START, END, TIME; // 小明的起点和终点，时间限制
    std::vector<std::vector<Node>> graph(N);
    for (int i = 0; i < M; i++)
    {
        int start, end, time, price;
        scanf("%d%d%d%d", &start, &end, &time, &price);
        graph[start].push_back({end, time, price});
    }
    scanf("%d%d%d", &START, &END, &TIME);

    long long min_price = 0x7fffffffffffffff;
    int repeat = 1600000; // 随机的重复次数

    srand(static_cast<unsigned int>(time(NULL)));
    for (int r = 0; r < repeat; r++)
    {
        // srand(static_cast<unsigned int>(time(NULL) + r));
        std::vector<bool> visited(N, false);
        visited[START] = true;
        long long cur_time = 0, cur_price = 0;
        int current = START;
        while (current != END)
        {
            if (graph[current].size() > 1)
            {
                int random = rand() % graph[current].size();
                cur_time += graph[current][random].Time;
                if (cur_time > TIME)
                {
                    break;
                }
                cur_price += graph[current][random].Price;
                current = graph[current][random].End;
            }
            else
            {
                break;
            }
            // std::vector<int> temp;
            // for (int i = 0; i < graph[current].size(); i++)
            // {
            //     if (visited[graph[current][i].End] == false)
            //     {
            //         temp.push_back(i);
            //     }
            // }
            // if (temp.empty())
            // {
            //     break;
            // }
            // int random = rand() % temp.size();
            // visited[graph[current][temp[random]].End] = true;
            // cur_time += graph[current][temp[random]].Time;
            // if (cur_time > TIME)
            // {
            //     break;
            // }
            // cur_price += graph[current][temp[random]].Price;
            // current = graph[current][temp[random]].End;
        }
        if (current == END && cur_time <= TIME && cur_price < min_price)
        {
            min_price = cur_price;
        }
    }

    // 所有路径全部超时，输出-1
    if (min_price == 0x7fffffffffffffff)
    {
        printf("%d", -1);
    }
    else
    {
        printf("%d", min_price);
    }
    return 0;
}

```

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）

```
#include <cstdio>
#include <vector>
// #include <cstdlib>
// #include <algorithm>
// #include <cmath>

struct Node
{
    int Start;
    int Time;
    int Cost;
};

int searchPath(int current, int START, int cur_time, int cur_cost, int TIME, std::vector<std::vector<Node>> &graph, std::vector<bool> &visited)
{
    // 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
    if (current == START && cur_time <= TIME)
    {
        return cur_cost;
    }
    if (graph[current].empty())
    {
        return 0x7fffffff;
    }
    int flag = 0;
    int min_cost = 0x7fffffff;
    for (const Node &neighbor : graph[current])
    {
```

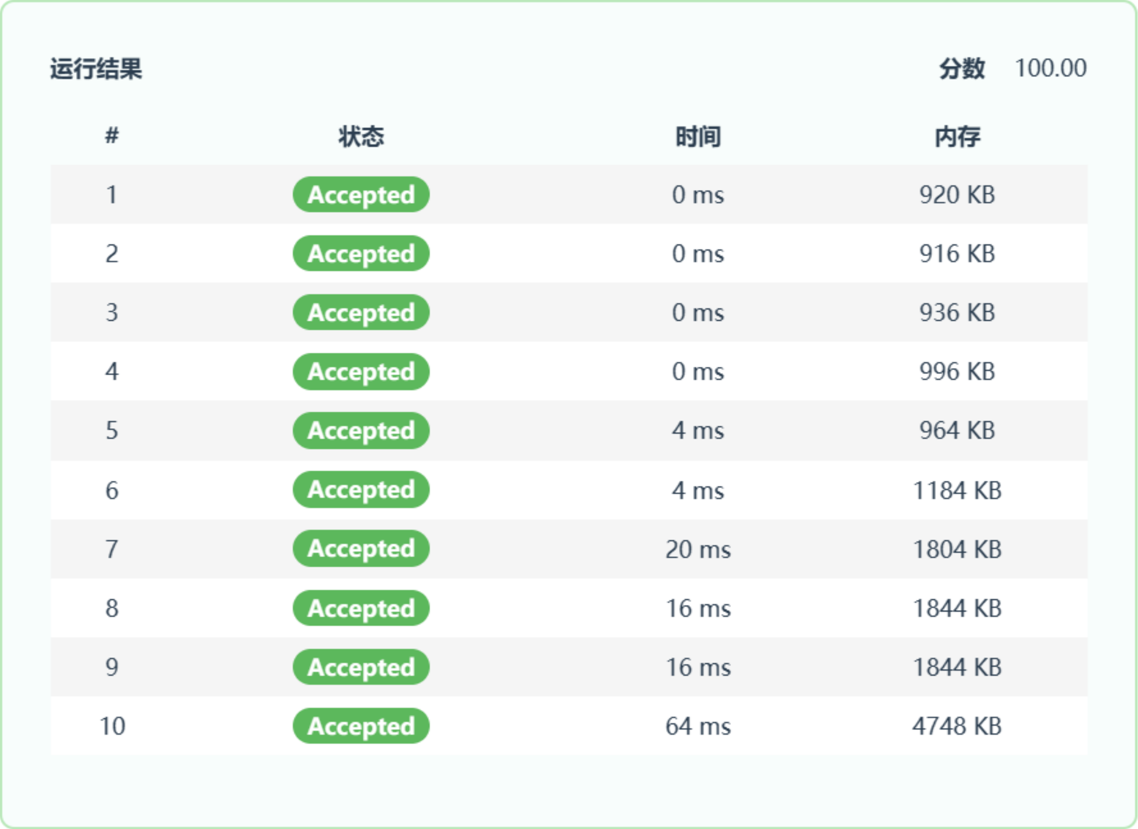
```cpp
        if (visited[neighbor.Start] == false)
        {
            if (cur_time + neighbor.Time > TIME)
            {
                continue; // 已经超时，提前退出
            }
            else
            {
                visited[neighbor.Start] = true;
                int cost = searchPath(neighbor.Start, START, cur_time + neighbor.Time, cur_cost + neighbor.Cost, TIME, graph, visited);
                if (cost < min_cost)
                {
                    min_cost = cost;
                }
                visited[neighbor.Start] = false;
                flag = 1;
            }
        }
    }
    if (flag == 0)
    {
        return 0x7fffffff; // 没有可以访问的节点
    }
    return min_cost;
}

int main()
{
    int N, M; // N个节点，M条边
    scanf("%d%d", &N, &M);
    int START, END, TIME; // 小明的起点和终点，时间限制
    std::vector<std::vector<Node>> graph(N);
    for (int i = 0; i < M; i++)
    {
        int start, end, time, cost;
        scanf("%d%d%d%d", &start, &end, &time, &cost);
        graph[end].push_back({start, time, cost});
    }
    scanf("%d%d%d", &START, &END, &TIME);

    int cur_time = 0, cur_cost = 0;
    std::vector<bool> visited(N, false);
    int min_cost = searchPath(END, START, cur_time, cur_cost, TIME, graph, visited);

    if (min_cost == 0x7fffffff)
    {
        printf("%d", -1);
    }
    else
    {
        printf("%d", min_cost);
    }
    return 0;
}
```

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