北京大学
PEKING UNIVERSITY

北京大学暑期课《ACM/ICPC竞赛训练》

北京大学信息学院 郭炜
guo_wei@PKU.EDU.CN
http://weibo.com/guoweiofpku

课程网页：http://acm.pku.edu.cn/summerschool/pku_acm_train.htm

深度优先搜索

寻路问题

ROADS （POJ1724）

N个城市，编号1到N。城市间有R条单向道路。
每条道路连接两个城市，有长度和过路费两个属性。
Bob只有K块钱，他想从城市1走到城市N。问最短共需要走多长的路。如果到不了N，输出-1

2<=N<=100
0<=K<=10000
1<=R<=10000
每条路的长度 L，1 <= L <= 100
每条路的过路费T ， 0 <= T <= 100

输入：
K
N
R
s_1 e_1 L_1 T_1
s_1 e_2 L_2 T_2
...
s_R e_R L_R T_R
s e是路起点和终点

解题思路

从城市 1开始深度优先遍历整个图，找到所有能过到达 N 的走法，选一个最优的。

解题思路

从城市 1开始深度优先遍历整个图，找到所有能过到达 N 的走法，
选一个最优的。

最优性剪枝：

1）如果当前已经找到的最优路径长度为L ，那么在继续搜索的过程中，总长度已经大于L的走法，就可以直接放弃，不用走到底了

解题思路

从城市 1开始深度优先遍历整个图，找到所有能到达 N 的走法，
选一个最优的。

最优性剪枝：

1）如果当前已经找到的最优路径长度为L ，那么在继续搜索的过程中，总长度已经大于L的走法，就可以直接放弃，不用走到底了

保存中间计算结果用于最优性剪枝：

2）用midL[k][m] 表示：走到城市k时总过路费为m的条件下，最优路径的长度。若在后续的搜索中，再次走到k时，如果总路费恰好为m，且此时的路径长度已经超过midL[k][m]，则不必再走下去了。

解题思路

另一种通用的最优性剪枝思想 ---保存中间计算结果用于最优性剪枝：

2） 如果到达某个状态A时，发现前面曾经也到达过A，且前面那次到达A所花代价更少，则剪枝。这要求保存到达状态A的到目前为止的最少代价。

用midL[k][m] 表示：走到城市k时总过路费为m的条件下，最优路径的长度。若在后续的搜索中，再次走到k时，如果总路费恰好为m，且此时的路径长度已经超过midL[k][m],则不必再走下去了。

```
#include <iostream>
#include <vector>
#include <cstring>
using namespace std;
int K,N,R;
struct Road {
    int d,L,t;
};
vector<vector<Road> > cityMap(110); //邻接表。cityMap[i]是从点i有路连到的城市集合
int minLen = 1 << 30;  //当前找到的最优路径的长度
int totalLen;   //正在走的路径的长度
int totalCost ; //正在走的路径的花销
int visited[110];  //城市是否已经走过的标记
int minL[110][10100]; //minL[i][j]表示从1到i点的，花销为j的最短路的长度
```

```
void Dfs(int s) //从 s开始向N行走
{
        if( s == N ) {
                minLen = min(minLen,totalLen);
                return ;
        }
        for( int i = 0 ;i < cityMap[s].size(); ++i ) {
                int d =  cityMap[s][i].d;  //s 有路连到d
                if(! visited[d] ) {
                    int cost = totalCost +  cityMap[s][i].t;
                    if( cost > K)
                                continue;
                    if( totalLen +  cityMap[s][i].L >= minLen ||
                      totalLen + cityMap[s][i].L >= minL[d][cost])
                                continue;
```

```
                    totalLen += cityMap[s][i].L;
                    totalCost +=  cityMap[s][i].t;
                    minL[d][cost] = totalLen;
                    visited[d] = 1;
                    Dfs(d);
                    visited[d] = 0;
                    totalCost -=  cityMap[s][i].t;
                    totalLen -= cityMap[s][i].L;
                }
        }
}
```

```
int main()
{
	cin >>K >> N >> R;
	for( int i = 0;i < R; ++ i) {
		int s;
		Road r;
		cin >> s >> r.d >> r.L >> r.t;
		if( s != r.d )
			cityMap[s].push_back(r);
	}
	for( int i = 0;i < 110; ++i )
		for( int j = 0; j < 10100; ++ j )
			minL[i][j] = 1 << 30;
	memset(visited,0,sizeof(visited));
	totalLen = 0;
	totalCost = 0;
	visited[1] = 1;
```

```
        minLen = 1 << 30;
        Dfs(1);
        if( minLen < (1 << 30))
                cout << minLen << endl;
        else
                cout << "-1" << endl;
}
```

海贼王之伟大航路

N个城市，编号1到N。起点是1，终点是N(N<=16)。

任意两个城市间都有路，A->B和B->A的路可能不一样长。

已知所有路的长度，问从1出发到达N且经每个城市恰好一次的最短路径的长度

样例输入：
4
0 10 20 999
5 0 90 30
99 50 0 10
999 1 2 0
样例输出：
100

最优性剪枝总结

1. 从初始状态到当前状态的代价已经不小于目前找到的最优解，则剪枝

最优性剪枝总结

1. 从初始状态到当前状态的代价已经不小于目前找到的最优解，则剪枝

2. 预测一下从当前状态到解的状态至少要花的代价W(可以很粗略很乐观，小于真实的最小代价)，如果W加上到达当前状态时已经花费的代价，必然不小于目前找到的最优解,则剪枝

最优性剪枝总结

1. 从初始状态到当前状态的代价已经不小于目前找到的最优解，则剪枝

2. 预测一下从当前状态到解的状态至少要花的代价W(可以很粗略很乐观，小于真实的最小代价)，如果W加上到达当前状态时已经花费的代价，必然不小于目前找到的最优解,则剪枝

3. 如果到达某个状态A时，发现前面曾经也到达过A，且前面那次到达A所花代价更少，则剪枝。这要求记录到目前为止到达状态A时所能取得的最小代价。

海贼王的最优性剪枝

- 怎么表示“状态”？
- 怎么预测从当前状态到解状态的至少代价？
- 如何存储到某个状态的“目前最小代价”？

海贼王的最优性剪枝

- 怎么表示“状态”？

状态，由两部分构成：

1） 已经走过的城市（除最后一个城市外，其他城市顺序不重要）
2） 走过的最后一个城市

海贼王的最优性剪枝

- 怎么预测从当前状态到解状态的至少代价？

海贼王的最优性剪枝

- 怎么预测从当前状态到解状态的至少代价？

  假设还有 c1, c2…. N 这k个城市还没有走到，则接下来一定要走k段两个城市间的道路，且这k段路终点分别是c1, c2 . . . N

  取终点为c1的路中最短的那条，再取终点为c2的路中最短的那条 . . . . 取终点为N的路中最短的那条。它们的长度之和，一定不大于把这k个城市都走到的最短路径的长度。

海贼王的最优性剪枝

- 如何存储到某个状态的“目前最小代价”？

海贼王的最优性剪枝

- 如何存储到某个状态的“目前最小代价”？

  状态，由两部分构成：

  1）已经走过的城市（除最后一个城市外，其他城市顺序不重要）
  2）走过的最后一个城市

  除解状态以外的状态总数上限：$14*2^{14}$

  可以用一个的$14*2^{14}$二维数组存放到达某个状态的“目前为止最小代价”

最优性剪枝总结

1. 从初始状态到当前状态的代价已经不小于目前找到的最优解，则剪枝

2. 预测一下从当前状态到解的状态至少要花的代价W，如果W加上到达当前状态时已经花费的代价，必然不小于目前找到的最优解,则剪枝

3. 如果到达某个状态A时，发现前面曾经也到达过A，且前面那次到达A所花代价更少，则剪枝。这要求记录到目前为止到达状态A时所能取得的最小代价。