



程序设计与算法(二)

算法基础

郭 炜

微博：<http://weibo.com/guoweiofpku>

学会程序和算法，走遍天下都不怕！

讲义照片均为郭炜拍摄



北京大学
PEKING UNIVERSITY

信息科学技术学院

配套教材：

高等教育出版社

《算法基础与在线实践》

刘家瑛 郭炜 李文新 编著

本讲义中所有例题，根据题目名称在
<http://openjudge.cn>
“百练”组进行搜索即可提交





深度优先搜索(二)



北京大学
PEKING UNIVERSITY

信息科学技术学院

例题: Roads



美国黄石公园

ROADS (百练1724)

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

$2 \leq N \leq 100$

$0 \leq K \leq 10000$

$1 \leq R \leq 10000$

每条路的长度 L , $1 \leq L \leq 100$

每条路的过路费 T , $0 \leq T \leq 100$

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，总长度已经大于等于 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，总长度已经大于等于 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;
```

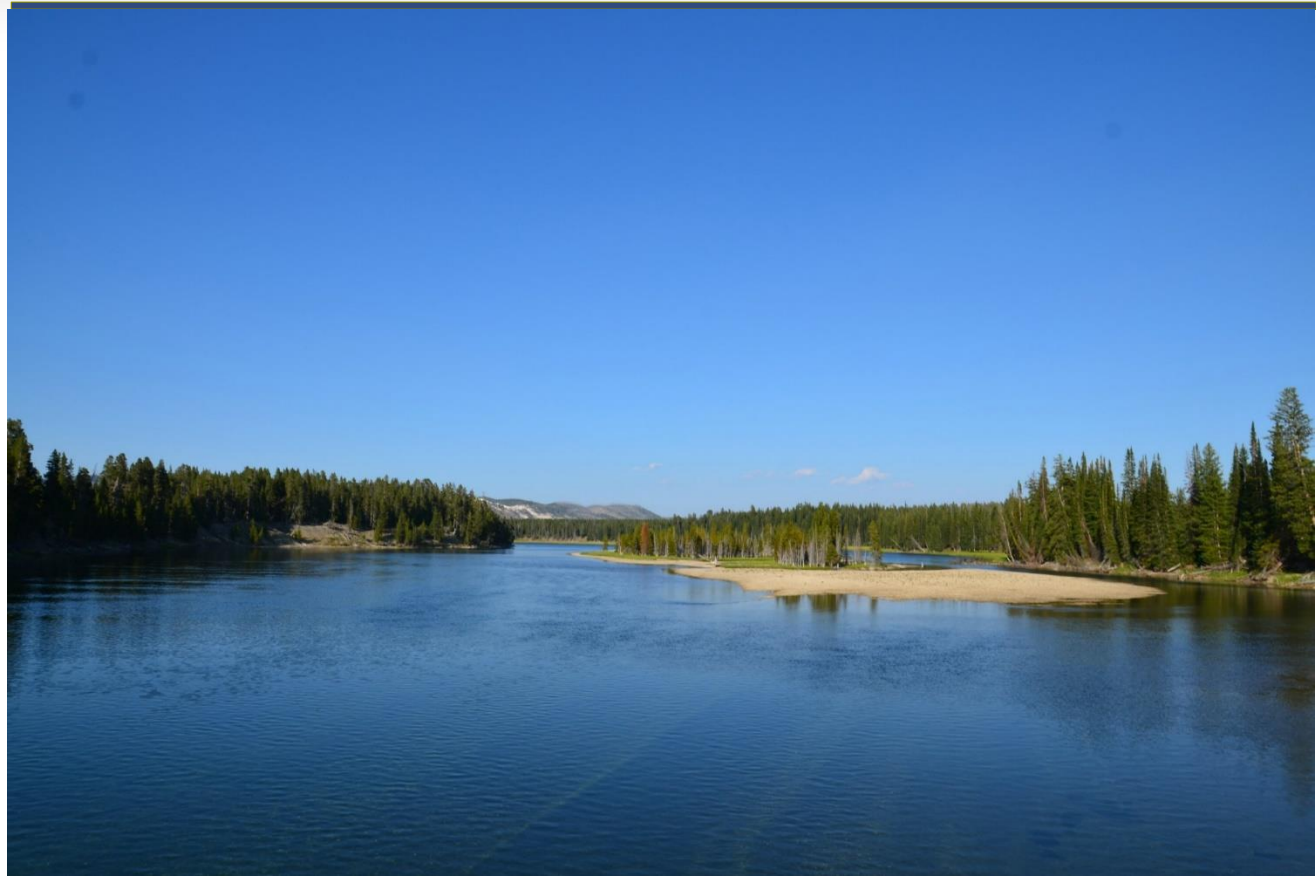
```
minLen = 1 << 30;  
Dfs(1);  
if( minLen < (1 << 30))  
    cout << minLen << endl;  
else  
    cout << "-1" << endl;  
}
```



北京大学
PEKING UNIVERSITY

信息科学技术学院

例题:生日蛋糕



美国黄石公园

生日蛋糕 (百练1190)

要制作一个体积为 $N\pi$ 的 M 层生日蛋糕，每层都是一个圆柱体。

设从下往上数第 i ($1 \leq i \leq M$)层蛋糕是半径为 R_i , 高度为 H_i 的圆柱。当 $i < M$ 时，要求 $R_i > R_{i+1}$ 且 $H_i > H_{i+1}$ 。

由于要在蛋糕上抹奶油，为尽可能节约经费，我们希望蛋糕外表面（最下一层的下底面除外）的面积 Q 最小。

$$\text{令 } Q = S\pi$$

请编程对给出的 N 和 M ，找出蛋糕的制作方案（适当的 R_i 和 H_i 的值），使 S 最小。

。（除 Q 外，以上所有数据皆为正整数）

解题思路

- 深度优先搜索，枚举什么？

解题思路

- 深度优先搜索，枚举什么？
枚举每一层可能的高度和半径。
- 如何确定搜索范围？

解题思路

- 深度优先搜索，枚举什么？
枚举每一层可能的高度和半径。
- 如何确定搜索范围？
底层蛋糕的最大可能半径和最大可能高度
- 搜索顺序，哪些地方体现搜索顺序？

解题思路

- 深度优先搜索，枚举什么？
枚举每一层可能的高度和半径。
- 如何确定搜索范围？
底层蛋糕的最大可能半径和最大可能高度
- 搜索顺序，哪些地方体现搜索顺序？
从底层往上搭蛋糕，而不是从顶层往下搭
- 如何剪枝？

```
#include <iostream>
#include <vector>
#include <cstring>
#include <cmath>
using namespace std;

int N,M;

int minArea = 1 << 30; //最优表面积
int area = 0; //正在搭建中的蛋糕的表面积
int main()
{
    .....
    Dfs ( N,M,maxR,maxH );
    if( minArea == 1 << 30)
        cout << 0 << endl;
    else
        cout << minArea << endl;
}

}
```

```

void Dfs(int v, int n,int r,int h)
//要用n层去凑体积v,最底层半径不能超过r,高度不能超过h
//求出最小表面积放入 minArea
{
    if( n == 0 ) {
        if( v ) return;
        else { minArea = min(minArea,area); return; }
    }
    if( v <= 0)
        return ;
    for( int rr = r; rr >=n; -- rr ) {
        if( n == M ) //底面积
            area = rr * rr;
        for( int hh = h; hh >= n ; --hh ) {
            area += 2 * rr * hh;
            Dfs(v-rr*rr*hh,n-1,rr-1,hh-1);
            area -= 2 * rr * hh;
        }
    }
}
}

```

剪枝

- 剪枝1：搭建过程中发现已建好的面积已经**不小于**目前求得的最优表面积，或者预见搭完后面积一定会**不小于**目前最优表面积,则停止搭建（**最优性剪枝**）

剪枝

- 剪枝1：搭建过程中发现已建好的面积已经超过目前求得的最优表面积，或者预见到搭完后面积一定会超过目前最优表面积,则停止搭建
(最优性剪枝)
- 剪枝2：搭建过程中预见到再往上搭，高度已经无法安排，或者半径已经无法安排，则停止搭建(可行性剪枝)

剪枝

- 剪枝1：搭建过程中发现已建好的面积已经超过目前求得的最优表面积，或者预见到搭完后面积一定会超过目前最优表面积,则停止搭建
(最优性剪枝)
- 剪枝2：搭建过程中预见到再往上搭，高度已经无法安排，或者半径已经无法安排，则停止搭建(可行性剪枝)
- 剪枝3：搭建过程中发现还没搭的那些层的体积，一定会超过还缺的体积，则停止搭建(可行性剪枝)

剪枝

- 剪枝1：搭建过程中发现已建好的面积已经超过目前求得的最优表面积，或者预见到搭完后面积一定会超过目前最优表面积,则停止搭建
(最优性剪枝)
- 剪枝2：搭建过程中预见到再往上搭，高度已经无法安排，或者半径已经无法安排，则停止搭建(可行性剪枝)
- 剪枝3：搭建过程中发现还没搭的那些层的体积，一定会超过还缺的体积，则停止搭建(可行性剪枝)
- 剪枝4：搭建过程中发现还没搭的那些层的体积，最大也到不了还缺的体积，则停止搭建(可行性剪枝)