最小重量机器设计问题

一．问题分析：

此问题有一个约束条件，即要求所选的部件的总价值不超过 d 值，优化所选的零件的供应商，使得其总重量最低，对于此问题可以采用优先队列的分支限界法，其解空间为排列树。

优先等级的权重：设定为到目前路径上的总重量，

剪枝条件：当不满足约束条件（即总的价值超过了 d），或者目前的路径上的总重量大于当前最优的值。

二．算法设计与分析：

N 为零件的总数目，m 为公司的个数，d 为最大花费。

HeapQueen 是一个二叉树的最小堆,类似于堆排序，可以向堆中添加组元（no(编号),weight(权重)），也可以返回权重最低组元的编号并删除相应的组元（popnode）。

Node 存储节点，包含有(weight(当前节点路径上的总重量)，cost(当前节点路径上的总花费),parentno(父节点编号)，companyno(选择的公司),lays(当前的部件的编号))属性。

Tree 是一个用 c++的 map 构成的树，其中的键为节点的编号，值为当前的节点。

Minweight 记录最优重量值。

1.首先从 heapQueen 队列中取出最小的节点编号，根据编号得到处理节点（Node），

2.如果扩展的子节点不是叶子节点时，即（处理节点的部件编号 lays 不等于 N-1）时，对节点进行拓展（子节点的重量=父节点的重量+当前编号所选择部件的重量，子节点的花费=父节点的花费+当前编号所选择部件的花费），有 m 个可以扩展的子节点，当子节点的花费小于 d 且子节点的总的重量小于 Minweight 时，则将此子节点的编号添加到 heapQueen 中，并将此节点储存至 Tree 树中。

3 如果扩展节点是叶子节点时，则将此叶子节点的总重量与最优重量值 进行比较，如果总重量小于最优值，则将最优重量值设置为当前子节点的总重量。用 MinNo 记录最优叶节点的编号

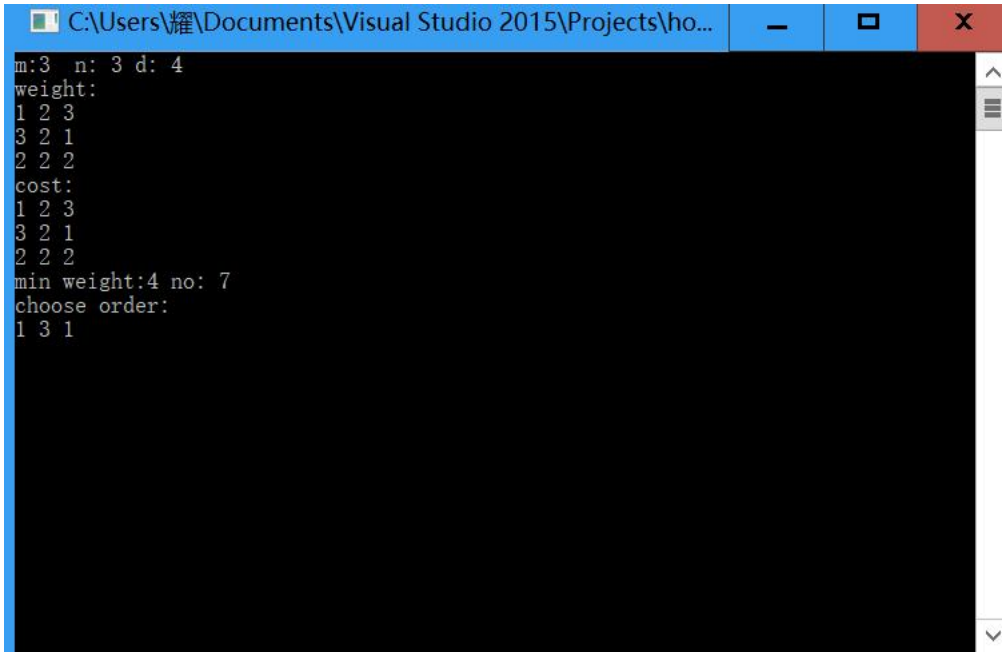
4.重复上述过程直到队列为空为止。

5.通过 MinNo 遍历其父节点，得到选择的路径。

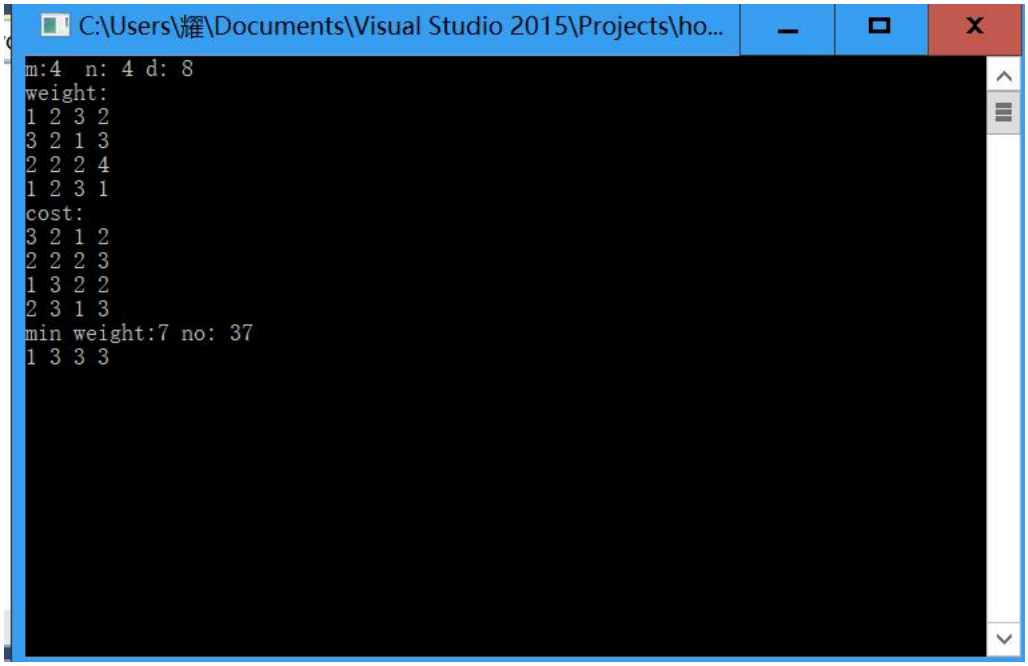
上述的最小堆，其排序算法复杂度为 n lgn,对其解进行搜索时，其空间复杂度和时间复杂度均和所构建的搜索树有关系，搜索树节点数目大小为：$O(m^n)$，故其算法的复杂度为：$O(m^n)$

三．**程序实现：**

Input.txt 记录输入信息，程序执行时会在屏幕上显示相关的信息，在控制台上输出，输出最小运行结果的对应叶节点的编号(no,)以及最小的重量值（min weight）。最后一排显示相应的选择方案，其运行结果如图下所示：



输入另外一组数据，其结果如下：



四. 代码：

```
#include "stdafx.h"
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <map>
#include<queue>
```

```
using namespace std;
struct Node  //节点
{
    int parentno;
    int companyno;
    int cost;
    int weight;
    int lays;
};
class HeapQueen //最小堆实现
{
private :
    vector< pair<int,int>> data; //存储数据：
    int length=0;
public:
    void addNode(int no,int cost) //添加节点：
    {
        pair<int, int> temp(no,cost);
        data.push_back(temp);
        length= data.size();
        int j =length;
        while(j>1)
        {
            int parent = j / 2;
            pair<int, int> tp0 = data[parent-1];
            pair<int, int> tp1 = data[j-1];
            if (tp0.second >= tp1.second)
            {
                swap(parent-1, j-1);
                j = j / 2;
            }
            else
                break;
        }
    }
    bool isempty() //是否为空
    {
        if (length > 0) return false;
        else return true;
    }
    int  popnode() //返回最小权重的节点编号，删除相应的节点。
    {
        int result = -1;
        if (length > 0)
```

```
    {
        pair<int, int > temp = data[0];
        result = temp.first;
        swap(0, length - 1);
        data.pop_back();
        int j = 1;
        length = data.size();
        while (j * 2  <(length-1))
        {
            int left = j * 2;
            int right = j * 2 + 1;
            int min = j;
            pair < int, int> tpl, tpr, tpn;
            tpn = data[j-1];
            tpl = data[left-1];
            tpr = data[right-1];
            if (tpl.second <= tpr.second)
            {
                min = left;
            }
            else
                min = right;
            if (data[min-1].second < tpn.second)
            {
                swap(j-1, min-1);
                j =min;
            }
            else
            {
                j = length;
                break;
            }
        }

    }
    return result;
}

bool swap(int i, int j)//交换数据中的两个元素
{
    if ((i < length) && (j < length))
    {
        pair<int, int > tp1;
        tp1 = data[i];
```

```
            data[i] = data[j];
            data[j] = tpl;
            return true;
        }
        else return false;
    }
    void  travel()
    {

        for (int i = 0; i < length; i++)
        {
            cout << data[i].first << " :" << data[i].second<<"    " ;
        }
    }

};
void testf()
{
    HeapQueen hq = HeapQueen();
    for (int i = 0; i < 20; i++)
    {
        int cost = rand()%100;
        hq.addNode(i, cost);
    }
    hq.travel();
    cout << endl;
    for (int i = 0; i < 20; i++)
    {
        int temp = hq.popnode();
        cout <<  i<<" :" << temp << "   ";
    }

    hq.travel();

}
int main()
{
    string fileurl = "input.txt";//输入文件地址
    ifstream fin(fileurl);
    int m, n, d;
    vector<vector<int>> weight;//存储重量矩阵
    vector<vector<int>> cost;//存储花费矩阵
    map<int, Node> tree;//搜索树
    HeapQueen dataqueue = HeapQueen();//最小堆
```

```
vector<int > result;//结果
bool iscontinue = 1;//循环显示结果
int index = 1;//当前的编号
int minweight =100; //最优总重量
int threshholde = 0;
int minNo = -1;//最优叶子节点编号：
//读取文件信息：
if (fin.is_open())
{
    //读取参数信息
    fin >> n;
    fin >> m;
    fin >> d;
    //读取重量矩阵
    for (int i = 0; i < n; i++)
    {
        vector<int> tv;
        for (int j = 0; j < m; j++)
        {
            int temp;
            fin >> temp;
            tv.push_back(temp);
        }
        weight.push_back(tv);
    }
    //读取花费：
    for (int i = 0; i < n; i++)
    {
        vector<int> tv;
        for (int j = 0; j < m; j++)
        {
            int temp;
            fin >> temp;
            tv.push_back(temp);
        }
        cost.push_back(tv);
    }
}
cout << "m:" << m << "  " << "n: " << n << " d: " << d<<endl;
//显示输入信息：
cout << "weight:" << endl;
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < m; j++)
```

```
    {
        cout << weight[i][j] << " ";
    }
    cout << endl;
}
cout << "cost:" << endl;
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < m; j++)
    {
        cout << cost[i][j]<<" ";
    }
    cout << endl;
}
fin.clear();
fin.close();
//初始化队列：
Node tn = Node();//根节点：
tn.companyno = -1;
tn.parentno = -1;
tn.cost = 0;
tn.weight = 0;
tn.lays = -1;
pair<int, Node> tp;
tp.first = -1;
tp.second = tn;
tree.insert(tp);
dataqueue.addNode(-1, 0);
//进入循环
while (iscontinue)
{
    int no=dataqueue.popnode();
    Node mynode = tree.at(no);
    int tc = mynode.cost;
    int tw = mynode.weight;
    int layer = mynode.lays+1;
    //cout << "no:" << no<< "  layer:" << layer << " weight" << tc << endl;

    for (int i = 0; i < m; i++)//拓展子 节点
    {

        int tcost = cost[layer][i] + tc;
        int tweight = tw + weight[layer][i];
            if ((tcost <= d)&&(tweight>=threshholde))//剪枝函数
```

```
                {
                    Node tnode = Node();
                    tnode.companyno = i;
                    tnode.weight = tweight;
                    tnode.cost = tcost;
                    tnode.lays = layer;
                    tnode.parentno = no;
                //  cout << "brand:" << index << "  layer:" << layer << " totalweight "
<< tweight<< endl;
                    if(layer==(n-1))// 如果是叶子节点
                    {
                        if (minweight > tweight)
                        {
                            minweight = tweight;
                            //cout << "tempminweight:" << minweight << " layer:" <<
layer << endl;
                            threshholde = minweight;
                            minNo = index;
                        }
                    }
                    else // 如果不是叶子节点
                    {
                        dataqueue.addNode(index, tweight);
                    }
                    pair<int, Node> myp;
                    myp.first = index;
                    myp.second = tnode;
                    tree.insert(myp);//加入搜索树中
                    index = index + 1;

                }
            }
        iscontinue = !dataqueue.isempty();
        }
    cout << "min weight:" << minweight << " no: " << minNo << endl;
    //寻找解向量。
    bool end = false;
    {
        int tpno = minNo;
        while (!end)
        {
            if (tpno != -1)
            {
                Node tnode2 = tree.at(tpno);
```

```
                tpno = tnode2.parentno;
                result.push_back(tnode2.companyno + 1);
                //cout << "layer: " << tnode2.lays+1 << " choose:" << tnode2.companyno+1
<< endl;

            }
            else
            {
                end = true;
                break;
            }
        }
    }
    //显示选择结果：
    cout << "choose order:" << endl;
    for (int j = result.size() - 1; j >= 0; j--)
    {
        cout << result[j] << " ";
    }
    int wait;
    cin >> wait;
    return 0;
}
```