1.结点的成员分析：

typedef struct CTNODE {

double cost; //存放个人经费消耗

int person\_num; //值为第个几人，值等于当前结点所在层次数-1

int task\_num; //值为选择第几个任务，值等于母结点的从左往右的第几个子结点数+1

//task\_num=0时，代表不做任务。

int task\_finished; //记录已经完成多少任务

int\* travellist; //记录是遍历过哪些任务，travellist从下标1开始，travellist[i]的值代表第i个人选择的task\_num(即第i个人选择的任务序号)

double allcost; //记录总费用

CTNODE\* child; //子结点

};

2.树的初始化部分：

从根结点开始初始化，使用了递归调用。

void createchildnode(CTNODE& parnode, int m, int n) {

if (parnode.person\_num < n) { //当结点的person\_num=n时停止递归，即叶子结点时跳过

parnode.child = new CTNODE[m + 1]; //创建m+1个叶子结点

for (int i = 0; i <= m; i++) {

parnode.child[i].person\_num = parnode.person\_num + 1; //子结点的层数为母结点的层数+1

parnode.child[i].task\_num = i; //子结点的task\_num赋值为母结点的第几个结点 parnode.child[i].task\_finished = 0; //完成任务数初始化为0

parnode.child[i].travellist = new int[n + 1];

for (int j = 0; j <= n; j++) {

parnode.travellist[j] = -1; //子结点的travellist中的值全部赋值-1

}

createchildnode(parnode.child[i], m, n); //递归

}

}

}

3.遍历思路：

从根结点开始遍历，遍历到第i层时：

遍历第0个子结点代表第i个人不分配任务，该结点的cost设为0，完成任务数(task\_finished)不增加。

遍历到第j个子结点代表第i个人分配第j个任务，该结点的cost设为提供的cost矩阵中的cost[i][j]，完成任务数(task\_finished)增加1。于此同时,遍历到该结点的总费用(allcost)=母结点的总费用+该结点的cost，该结点travellist[i]=j。

遍历同样使用递归实现。但是要用到回溯法。简单来说，就是一个判定：

如果子结点的task\_num在母结点的travellist中出现过，那么意味着第j个任务已经被完成了，停止递归。

遍历的停止有两种情况：

①task\_num=m，即完成任务数等于所有的任务数，此时把该结点存进辅助的leafnode数组。

②已经遍历到第n层，跳过。

4.排序思路：

这个好办，遍历完成以后，辅助的leafnode数组中已经放入所有可行方案的结点，从所有的结点里选出allcost值最小的那个就行，把最小的那个对应的结点记为minleaf。

minleaf的allcost即为总费用，minleaf的travellist[i]的值=第i个人选择的任务编号（编号为0时代表不分配任务）。