

- 1 人们都说昆明四季如春，南京却是“春如四季”。为了验证这一说法，南京市气象局在开春以后的 n 天，每天测量一个固定时刻的温度，并记录在数组 $a[1..n]$ 中，希望从中找出温差最大的两天。也就是说，希望找到 $a[i]$ 和 $a[j]$ ， $i < j$ 使得 $a[j]-a[i]$ 的绝对值 M 最大，即 $M = \max\{|a[j]-a[i]|, 1 \leq i < j \leq n\}$ 。试设计一个复杂度为 $O(n \log n)$ 分治算法求解 M 。请说明理由并给出伪代码及时间复杂度分析。（20 分）

一、思路

不断尽可能均分区域，最终由各区域最大最小值的差判定合并后的分区结果，最终得到最优解。

二、算法伪代码

```
1. int* divide_and_conquer(int* a){
2.     int M[4] = {0,0}; //M[0]存 min, M[1]存 max, M[2]存 min 的位置, M[3]存 max 的位置
3.     if(|a|<=2) M = solve(a);
4.     //将传入的 a 的数组划分为数量相等或差 1 的左右两个子集 al 与 ar
5.     int* SL = divide_and_conquer(al);
6.     int* SR = divide_and_conquer(ar);
7.     return merge(SL,SR); //得到的 M 数组中, M[0]=min, M[1]=max, M[2]与 M[3]为温差最大的两天
8. }
9. int* solve(int* a){
10.    int M[2]={0,0} //M[0]存 min, M[1]存 max
11.    if (length(a)==1)M[0]=M[1]=a[0];
12.    else{
13.        M[0]=min(a);
14.        M[1]=max(a); //记录最值
15.        M[2]=loc(min(a));
16.        M[3]=loc(max(a)); //记录最值位置
17.    }
18.    return M;
19. }
20. int* merge(int*SL,int* SR){
21.    //比较 SL 与 SR 中的信息, 使得:
22.    M[0]=min(SL[0],SR[0]);
23.    M[1]=max(SL[1],SR[1]); //选出最值
24.    M[2]=loc(min(SL[2],SR[2]));
25.    M[3]=loc(max(SL[3],SR[3])); //更改最值位置
26.    return M;
27. }
```

三、时间复杂度

$$T(n) = \begin{cases} O(1), n \leq 2 \\ T(nl) + T(nr) + T(merge), n > 2 \end{cases}$$

$$T(nl) = \begin{cases} O(1), n \leq 2 \\ T(nll) + T(nlr) + T(merge), n > 2 \end{cases}$$

$$T(nr) = \begin{cases} O(1), n \leq 2 \\ T(nrl) + T(nrr) + T(merge), n > 2 \end{cases}$$

$$\text{因此 } T(n) = \frac{n}{2}T_{solve} + \frac{n}{4}T_{merge} + \frac{n}{8}T_{merge} + \cdots + \frac{n}{2^i}T_{merge}$$

$$= \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \cdots + \frac{n}{2^i}$$

$$= O(n)$$

$$\leq O(n \log n)$$

- 2 假设有 n 个活动在申请一项公共的资源。活动 a_i ($1 \leq i \leq n$) 的开始时间和结束时间分别为 $S[i]$ 和 $F[i]$, $0 < S[i] < F[i]$ 。若两个活动 a_i 和 a_j ($1 \leq i, j \leq n$) 的 $[S[i], F[i])$ 和 $[S[j], F[j])$ 不重叠, 我们就说这两个活动是相互兼容的。此外, 任一活动 a_i ($1 \leq i \leq n$) 有一个相应的收益 $V[i]$ (> 0), 即为若活动 a_i 被选择使用该资源可以带来的收益。假设活动已经按照开始时间排好序, 即 $S[1] \leq S[2] \leq \dots \leq S[n]$ 。

(a) 请设计一个 $O(n^2)$ 的动态规划算法用于选择一个兼容的活动子集, 使得这些活动带来的收益最大化。请定义子问题, 建立递推式, 给出伪代码, 时间复杂度计算公式 (12 分)。

(b) 请将该算法具体应用于下面的活动集, 选择可以带来最大收益的兼容活动子集 (8 分)。

i	1	2	3	4	5	6	7	8	9	10	11
$S[i]$	2	3	5	6	7	9	10	12	13	14	16
$F[i]$	6	5	7	10	8	13	16	14	14	18	20
$V[i]$	7	3	9	1	6	10	2	5	8	4	5

(a)

一、思路

第 i 次合并 $i+1$ 个活动, 直到所有活动被考虑到;

二、子问题

范围为 $[l, r]$ 时的最大收益是多少, 当 $l=r$ 时返回此活动 value;

三、算法伪代码

```

1. //second
2. int ActivitiesValue(int l,int r){
3.     if(l==r) return V[l];
4.     int u = 0; //存储最优收益
5.     for(int i=l;i<=r;i++){
6.         if(F[i]<=S[i+1]){
7.             if(ActivitiesValue(l,i)+ActivitiesValue(i+1,r) > u)
8.                 u = ActivitiesValue(l,i)+ActivitiesValue(i+1,r);
9.             //同时可在主函数中设定一个数组, 存储取 u 时的位置信息
10.        }else{

```

```

11.         u = max(ActivitiesValue(l,i),ActivitiesValue(i+1,r));
12.     }
13. }
14.     return u;
15. }

```

四、时间复杂度

$$T(n) = \begin{cases} 1, n = 1 \\ 2(n-1)T(n-1), n > 1 \end{cases}$$

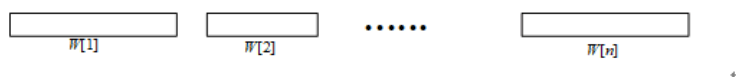
(b)

	A	B	C	D	E	F	G	H	I	J	K	L
1	r=1	V[1,1]=7	V[2,2]=3	V[3,3]=9	V[4,4]=1	V[5,5]=6	V[6,6]=10	V[7,7]=2	V[8,8]=5	V[9,9]=8	V[10,10]=4	V[11,11]=5
2		1	2	3	4	5	6	7	8	9	10	11
3	r=2	V[1,2]=7	V[2,3]=12	V[3,4]=9	V[4,5]=6	V[5,6]=16	V[6,7]=10	V[7,8]=5	V[8,9]=8	V[9,10]=12	V[10,11]=5	
4		1	2,3	3	5	5,6	6	8	9	9,10	11	
5	r=3	V[1,3]=12	V[2,4]=12	V[3,5]=15	V[4,6]=16	V[5,7]=16	V[6,8]=10	V[7,9]=8	V[8,10]=12	V[9,11]=13		
6		2,3	2,3	3,5	5,6	5,6	6	9	9,10	9,11		
7	r=4	V[1,4]=12	V[2,5]=18	V[3,6]=25	V[4,7]=16	V[5,8]=16	V[6,9]=18	V[7,10]=12	V[8,11]=13			
8		2,3	2,3,5	3,5,6	5,6	5,6	6,9	9,10	9,11			
9	r=5	V[1,5]=18	V[2,6]=28	V[3,7]=25	V[4,8]=16	V[5,9]=24	V[6,10]=22	V[7,11]=13				
10		2,3,5	2,3,5,6	3,5,6	5,6	5,6,9	6,9,10	9,11				
11	r=6	V[1,6]=28	V[2,7]=28	V[3,8]=25	V[4,9]=24	V[5,10]=28	V[6,11]=23					
12		2,3,5,6	2,3,5,6	3,5,6	5,6,9	5,6,9,10	6,9,11					
13	r=7	V[1,7]=28	V[2,8]=28	V[3,9]=33	V[4,10]=28	V[5,11]=29						
14		2,3,5,6	2,3,5,6	3,5,6,9	5,6,9,10	5,6,9,11						
15	r=8	V[1,8]=28	V[2,9]=36	V[3,10]=37	V[4,11]=29							
16		2,3,5,6	2,3,5,6,9	3,5,6,9,10	5,6,9,11							
17	r=9	V[1,9]=36	V[2,10]=40	V[3,11]=38								
18		2,3,5,6,9	2,3,5,6,9,10	3,5,6,9,11								
19	r=10	V[1,10]=40	V[2,11]=41									
20		2,3,5,6,9,10	2,3,5,6,9,11									
21	r=11	V[1,11]=41										
22		2,3,5,6,9,11										

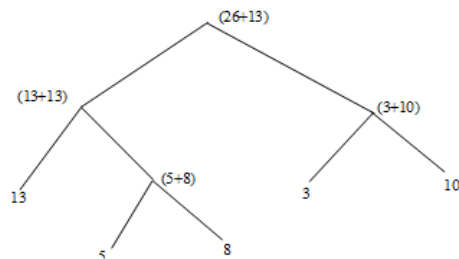
所以最大收益为 41，最大收益的兼容活动子集为{2,3,5,6,9,11}。

3 假设我们须要把 n 个钢管， a_1, a_2, \dots, a_n 焊成一根钢管。这些钢管的重量分别是 $w[i]$ 。

$1 \leq i \leq n$ ，如下图所示。



每次焊接你可以从被焊钢管中任选二根来焊，但每次焊接的代价等于被焊两根钢管重量之和。比如我们有 5 根钢管，重量为 3，8，5，10，13。显然，任何一个焊接计划可以用一个有 n 个叶子的满二叉树（full binary tree）表示。如果我们按下面二叉树所示的顺序焊接，那么总的代价为 $(5+8) + (13+13) + (3+10) + (26+13) = 91$ 。



(a) 假设有一个 n 个叶子的满二叉树 T 表示一个焊接计划，证明这个焊接计划的总代价为 $\text{Cost}(T) = \sum_{k=1}^n w[k] \text{depth}(k)$ ，其中 $\text{depth}(k)$ 是代表钢管 a_k 的叶子在树中的深度（5分）。

(b) 设计一个有效贪心算法为这 n 个钢管产生一个最佳焊接计划，给出贪心策略（5分）、伪代码（5分）、算法的正确性证明（5分）。

(a)

证明: \because 每个叶子节点返回一个实值

$$\begin{aligned}\therefore \text{Cost}(T) &= \text{depth}(1)*W[1] + \text{depth}(2)*W[2] + \dots + \text{depth}(n)*W[n] \\ &= \sum_{k=1}^n W[k]*\text{depth}(k)\end{aligned}$$

(b)

贪心策略:

每次选择 Cost 最小的两个节点合并, 并记录新的 Cost 作为该结点的 Cost, 直到所有节点合并为一个结点, 得到的 Cost 则为最小的 Cost, 之前的合并路径则为最佳焊接计划。

伪代码:

```
1. int* temp = W; //存放新的排列, 下标大于 n 时存放新结点 Cost
2. int plan[2][n];
3. int j = n+1;
4. while(!temp 只有最后一项不为零){
5.     for(int i=1; i<=j; i++){ //每次选择最小的两个值
6.         if(temp[i]==0) continue;
7.         min1=最小值;
8.         min2=次小值;
9.     }
10.    //将下标记录在 plan 对应次数-1 中
11.    temp[j++]=min1+min2;
12.    //置 temp 中原始 min1, min2 对应值为 0
13. }
14. //最终 temp[j] 为最小值 Cost
15. //最佳方案为 plan, 其中 temp[x] (x>n) 的部分可由 plan 的前半部分推出
```

证明:

① 最优子结构

假设 $S_i = \{g_i\}$ 不是最优解, 即存在更小的 Cost: $\min < \min1 \leq \min2$

而 $S_{i+1} = \{g_{i+1}\}$ 为局部最优解, 但 $\min + \min1 + \min3 < \min1 + \min2 + \min3$, 即存在更优解
二者矛盾, 综上: $S_i = \{g_i\}$ 是最优解。

② 包含第一次的结果

假设 S^* 为最优解

若 $S^* = \{g_1^*, g_2^*, \dots, g_n^*\}$ 不包含第一次的值, 则 $\exists \|g_1^*\| \geq \|g_1\|$, 则 $\|S^*\| \geq \|S\|$

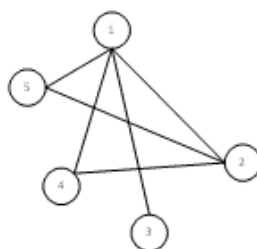
又 $\because S^*$ 最优, 则 $\|S^*\| \leq \|S\|$

$\therefore S^* = S$, 即一定包含第一次的结果

4 原始部落 byteland 中的居民们为了争夺有限的资源，经常发生冲突。几乎每个居民都有他的仇敌，部落首长为了组织一支保卫部落的队伍，希望从部落的居民中选出最多的居民入伍，并保证队伍中任何 2 个人都不是仇敌。给定 byteland 部落中居民间的仇敌关系（如下图所示，图中的点表示居民，边表示居民 {1, 2, 3, 4, 5} 的仇敌关系），计算组成部落卫队的一种最佳方案。

a) 令解空间树的第 i 层表示居民 i 是否被选中入伍，定义该问题可行性约束函数和限界函数（4 分），用回溯法求解该问题，画出剪枝后的解空间树（4 分），说明每次剪枝的理由（2 分）；请给出回溯法具体实现的代码（10 分）。

b) 我们可通过设计更好的限界函数和优先级函数、对居民重新排序（如令解空间树的第 i 层表示居民 $p(i)$ 是否被选中入伍）等方法使剪枝效果更好。请根据上面提示设计基于优先级队列的分支限界算法，写出优先级函数和居民的排序（4 分），画出剪枝后的解空间树并标明节点扩展的顺序（4 分），说明每次剪枝的理由（2 分）。请给出基于优先级队列的分支限界法的具体实现的代码（10 分）。

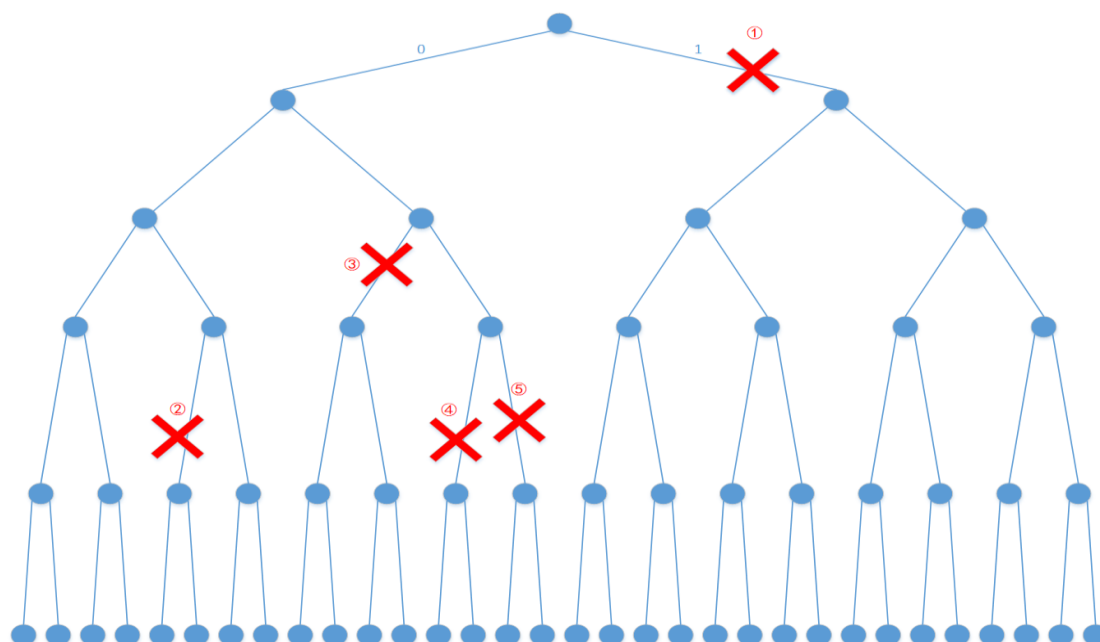


a)

Constraint Function: $\text{Enemy}(i,j) \neq \text{Edge}$

Bounding Function: Whether there are enough citizens can be employed.

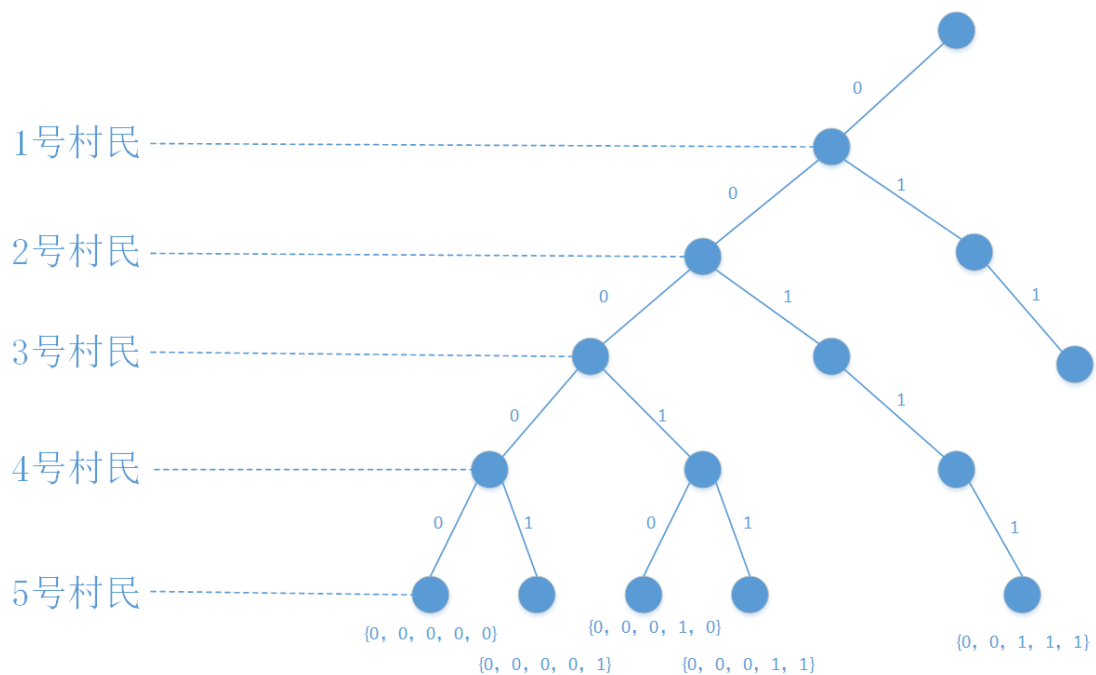
Current amount(cuNum) + the amount of remaining citizens(reNum) > the largest amount so far(bestNum)



剪枝原因：

- ①Constraint Function 不满足，1 与其他所有人都为仇敌；
- ②Bounding Function 不满足，最多只有 2 个人，最多与之前最多结果相同；
- ③Bounding Function 不满足，最多只有 3 个人，最多与之前最多结果相同；
- ④Bounding Function 不满足，最多只有 3 个人，最多与之前最多结果相同；
- ⑤Constraint Function 不满足，4 与 2 为仇敌关系。

剪枝结果：



代码实现（详见程序附件 1.cpp）：

```
1. #include <iostream>
2. using namespace std;
3. #define citizensNum 5
4.
5. int bestNum = 0;
6. class byteland
7. {
8. public:
9.     int citizens[5] = { 1,2,3,4,5 };
10.    int enemy[5][5] = { {0,1,1,1,1},{1,0,0,1,1},{1,0,0,0,0},{1,1,0,0,0},{1,1,0,0,0} };//仇敌关系
11. };
12.
13. typedef struct node{//节点与树结构体
14.     int value = 0;
15.     int depth = 0;
16.     int num = 0;
17.     int trace[5] = { 0 };
```

```

18.     struct node *leftChild;
19.     struct node *rightChild;
20.     node():leftChild(NULL),rightChild(NULL){}
21. }Node, *Tree;
22.
23. void printTrace(node cuNode) {
24.     for (int i = 0; i < citizensNum; i++) {
25.         if (cuNode.trace[i] == 1)    cout << i + 2 << '\t';
26.     }
27.     cout << "may be a group." << endl;
28. }
29.
30. void createTree(int depth, int scale, Tree &T, int value, int* trace){//构造
    一棵 scale 深度的完全二叉树
31.     if (depth+1 == scale) {
32.         T = new Node();
33.         T->value = value;
34.         T->depth = depth+1;
35.         for (int i = 1; i <= scale; i++) {
36.             if (i == depth+1) {
37.                 T->trace[i - 1] = T->value;
38.                 continue;
39.             }
40.             T->trace[i - 1] = trace[i - 1];
41.         }
42.         // printTrace(*T);
43.     }
44.     else {
45.         T = new Node;
46.         T->value = value;
47.         T->depth = depth+1;
48.         for (int i = 1; i <= scale; i++) {
49.             if (i == T->depth) {
50.                 T->trace[i-1] = T -> value;
51.                 continue;
52.             }
53.             T->trace[i-1] = trace[i-1];
54.         }
55.         createTree(T->depth, scale, T->leftChild, 0, T->trace);
56.         // printTrace(*T);
57.         createTree(T->depth, scale, T->rightChild,1, T->trace);
58.         // printTrace(*T);
59.     }
60. }

```

```

61.
62. bool cstrFunc(byteland BL, node cuNode) {
63.     if (cuNode.value == 1 && BL.enemy[cuNode.depth-
        1][cuNode.depth] == 1) return false;//当二者为仇敌时退出返回 false
64.     else return true;
65. }
66.
67. int sumOfTrace(node cuNode) {
68.     int cuNum = 0;
69.     for (int i = 0; i < cuNode.depth; i++) { //计算当前 trace 上的和
70.         cuNum += cuNode.trace[i];
71.     }
72.     return cuNum;
73. }
74.
75. bool bndFunc(byteland BL, node cuNode) {
76.     if (sumOfTrace(cuNode) + citizensNum - cuNode.depth <= bestNum) return f
        alse;//当不会出现更优解时退出返回 false
77.     else return true;
78. }
79.
80. bool getLeaf(byteland BL, node cuNode) { //若有解返回 true, 被剪枝返回 false
81.     // if (cstrFunc(BL, cuNode)) cout << "c";
82.     // if (bndFunc(BL, cuNode)) cout << "b";
83.     if (cuNode.depth != citizensNum && cstrFunc(BL, cuNode) && bndFunc(BL, c
        uNode)) { //没到底+满足约束+满足边界
84.         if (getLeaf(BL, *cuNode.leftChild) || getLeaf(BL, *cuNode.rightChild
            )) {
85.             if (getLeaf(BL, *cuNode.leftChild)) {
86.                 //bestNum += cuNode.leftChild->value;
87.                 //cout << cuNode.leftChild->value;
88.             }
89.             if (getLeaf(BL, *cuNode.rightChild)) {
90.                 //bestNum += cuNode.rightChild->value;
91.                 //cout << cuNode.rightChild->value;
92.             }
93.             return true;
94.         }
95.         else {
96.             return false;
97.         }
98.     }
99.     else if (cuNode.depth == citizensNum && cstrFunc(BL, cuNode) && bndFunc(
        BL, cuNode)) {

```

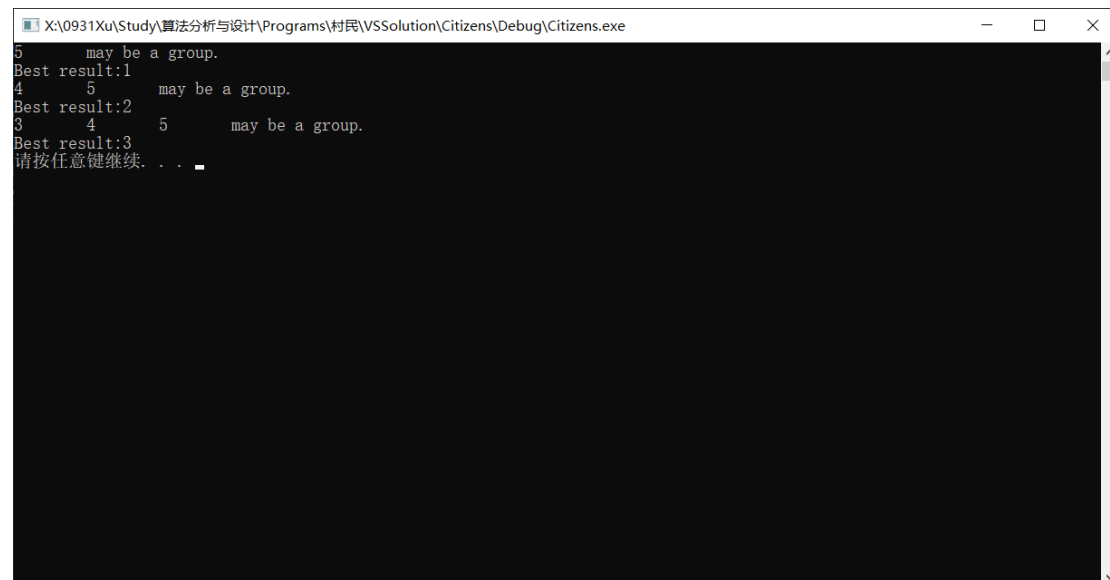


```

100.         bestNum = sumOfTrace(cuNode);
101.         printTrace(cuNode);
102.         cout << "Best result:" << bestNum << endl;
103.         return true;
104.     }
105.     else {
106.         return false;
107.     }
108. }
109.
110. int main()
111. {
112.     Tree root;
113.     byteland myBL;
114.     int scale = citizensNum;
115.     int trace[5] = { 0 };
116.
117.     createTree(0, scale, root, 0, trace);
118.     getLeaf(myBL, *root);
119.
120.     system("pause");
121.     return 0;
122. }

```

结果截图：



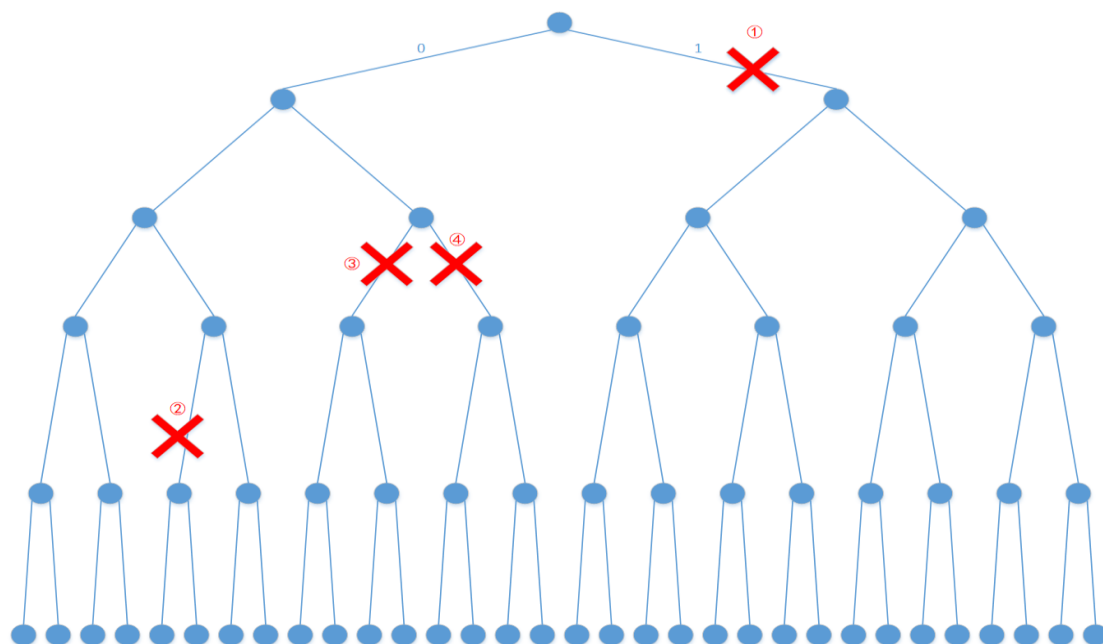
```

X:\0931Xu\Study\算法分析与设计\Programs\村民\VSSolution\Citizens\Debug\Citizens.exe
5      may be a group.
Best result:1
4      5      may be a group.
Best result:2
3      4      5      may be a group.
Best result:3
请按任意键继续. . .

```

b)

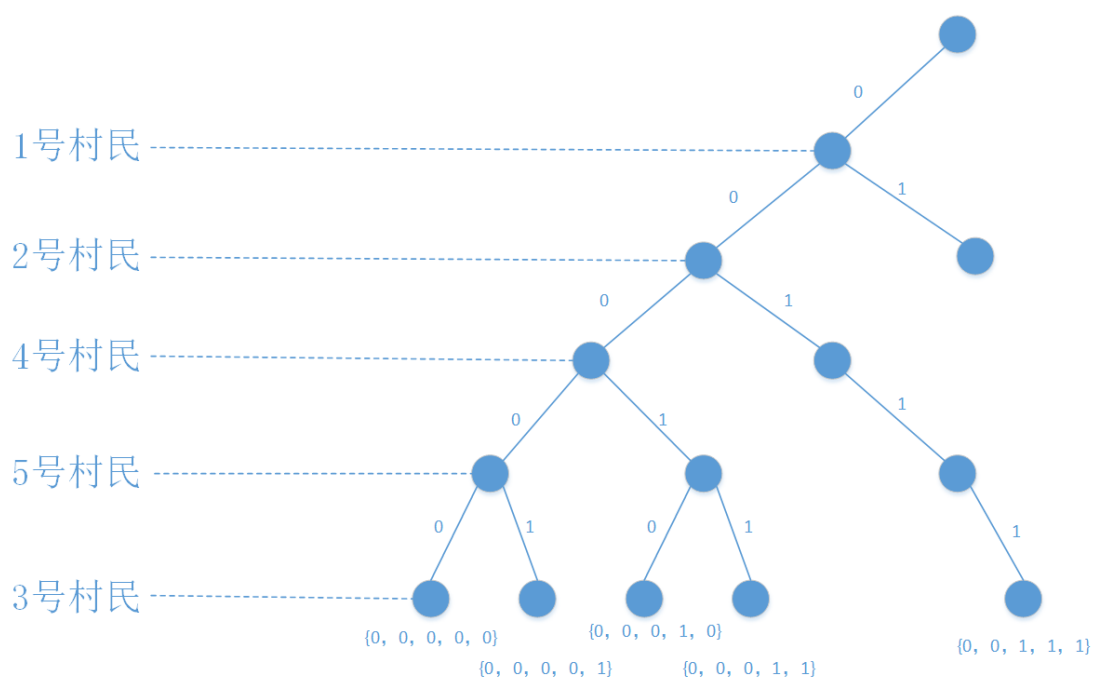
优先级函数：f(i)为以仇敌数从多到少重新排列的函数，重排后为{1,2,4,5,3}



剪枝原因：

- ①Constraint Function 不满足，1 与其他所有人都为仇敌；
- ②Bounding Function 不满足，最多只有 2 个人，最多与之前最多结果相同；
- ③Bounding Function 不满足，最多只有 3 个人，最多与之前最多结果相同；
- ④Constraint Function 不满足，2 与 4 为仇敌关系。

剪枝后结果：



代码实现：

添加一段对村民以仇敌数降序排序的函数即可实现（主函数等参见程序 2.cpp 附件）；

```
1. int* getEnemy() {
2.     int sumArray[citizensNum] = { 0 };
}
```

```

3.     for (int i = 0; i < citizensNum; i++) { //循环得到各村民仇敌数
4.         for (int j = 0; j < citizensNum; j++) {
5.             sumArray[i] += enemy[i][j];
6.         }
7.     }
8.     return sumArray;
9. };
10.
11. void enemyRank() {
12.     for (int m = 0; m < citizensNum; m++) { //将村民顺序以仇敌数递减排序
13.         while (getEnemy()[m] < getEnemy()[m + 1]) {
14.             swap(enemy[m], enemy[m + 1]);
15.             swap(citizens[m], citizens[m + 1]);
16.             m = 0;
17.         }
18.     }
19. };

```

结果截图：

```

X:\0931Xu\Study\算法分析与设计\Programs\村民\VSsolution\Citizens\Debug\Citizens.exe
3      may be a group.
Best result:1
5      3      may be a group.
Best result:2
4      5      3      may be a group.
Best result:3
请按任意键继续. . .

```