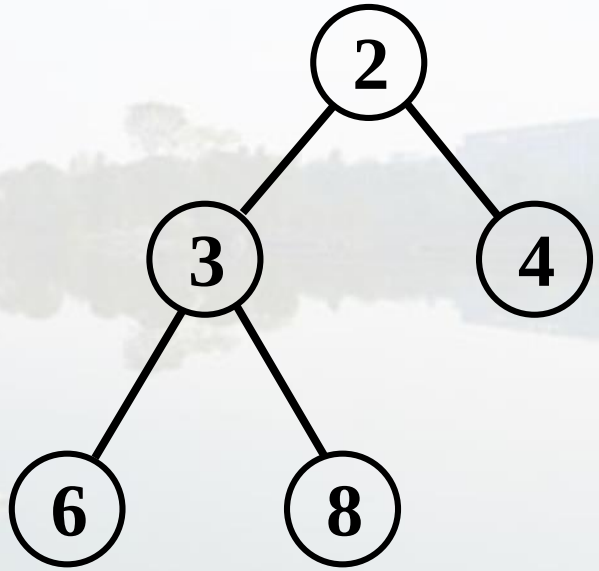
3. 堆

堆是一棵完全二叉树，一般将其分为大顶堆和小顶堆。小顶堆是指子结点的值大于父结点的值；大顶堆就是堆的子结点的值都小于父结点的值。下面我们以小顶堆为例说明操作。

在实现的时候，常常使用基于数组的堆(由于是完全二叉树，所以元素在数组中是连续的)。
如果一个结点的编号为n，那么，其对应的左右子结点的编号分别是2n和2n+1。

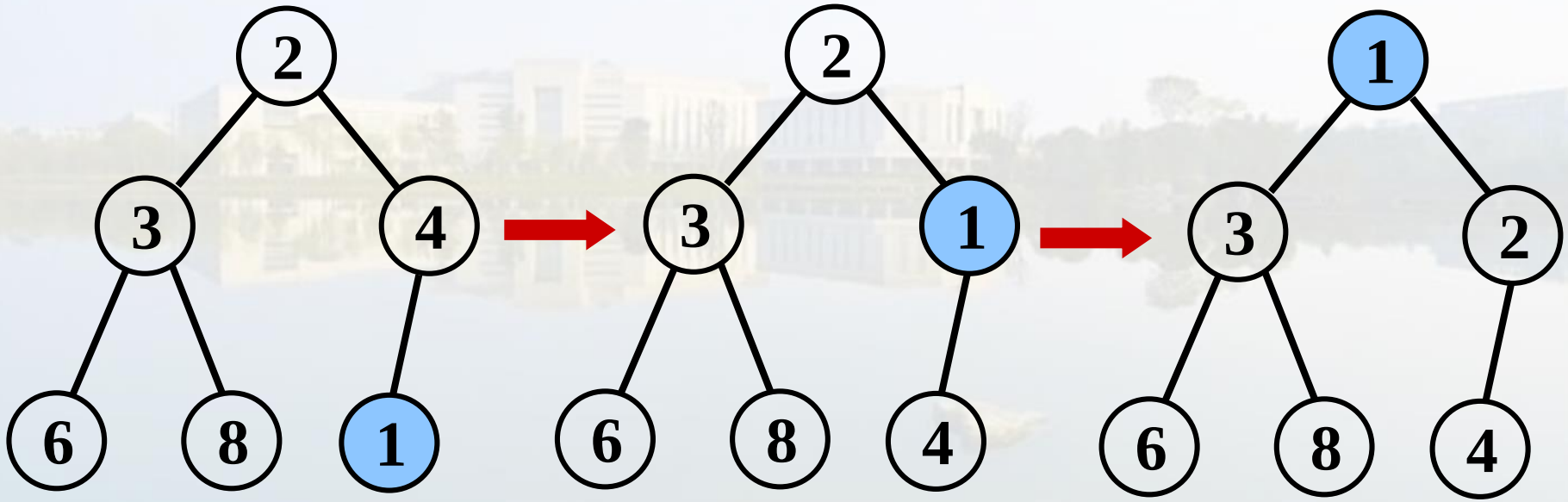


如果一个结点的编号为n，则其父结点的编号为n/2。

(1) 添加操作

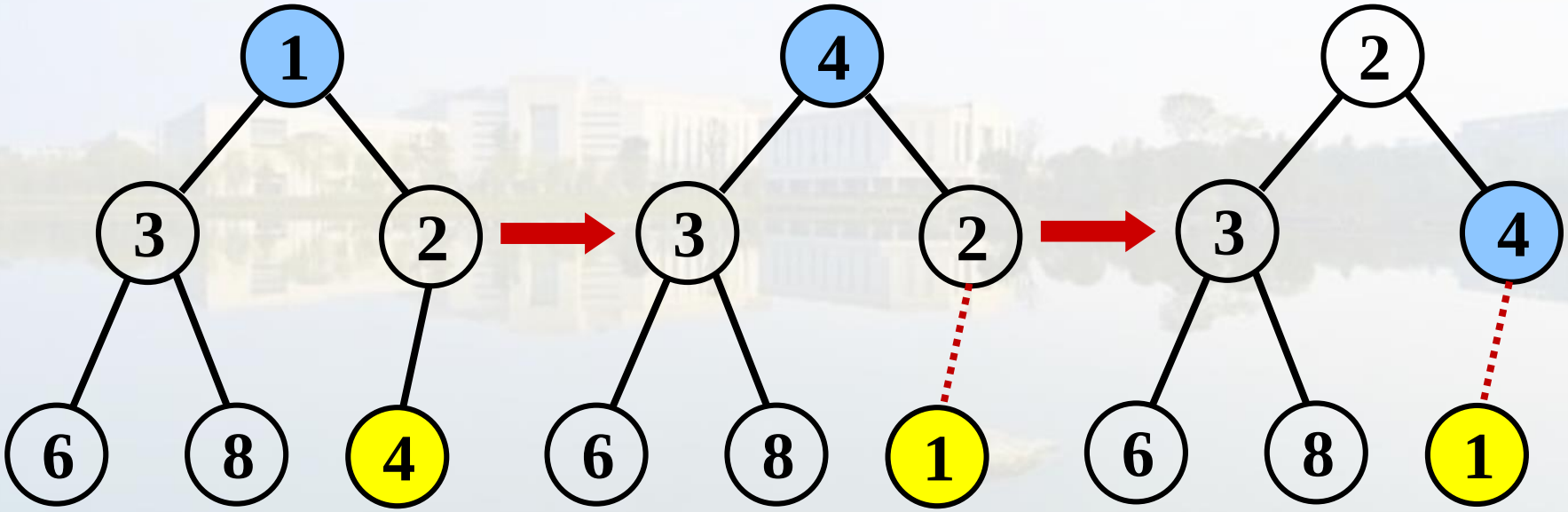
如果要添加一个元素，首先把这个元素放在最后，然后与其父结点比较，如果不满足堆的性质，则交换。如果不满足堆的性质，则继续进行下去即可。

比如要添加元素1:



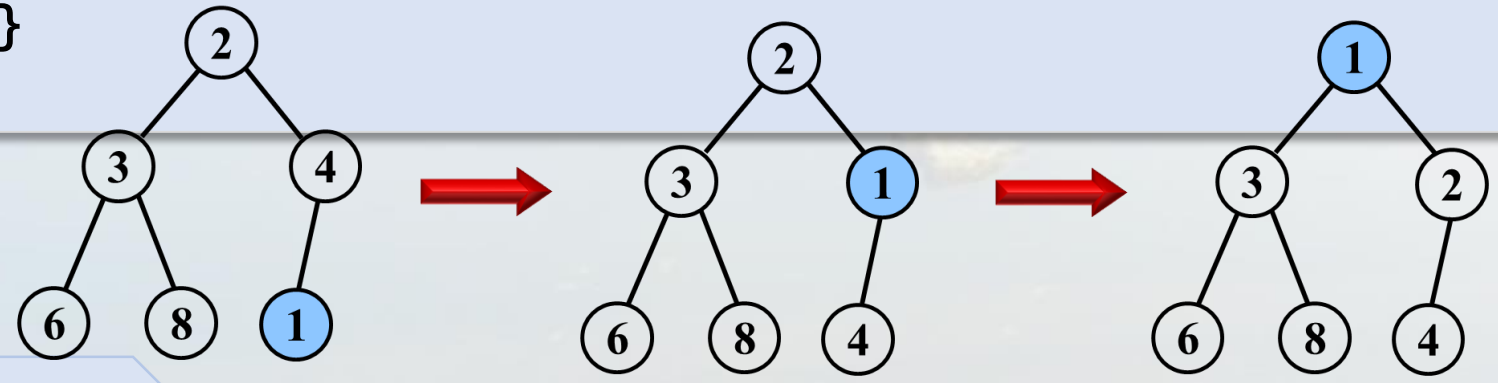
(2) 删顶

首先把最后一个元素与顶交换，如果顶不满足堆的性质，则与左右儿子比较，与较小的儿子交换，然后继续下去，又得到一个正确的堆。



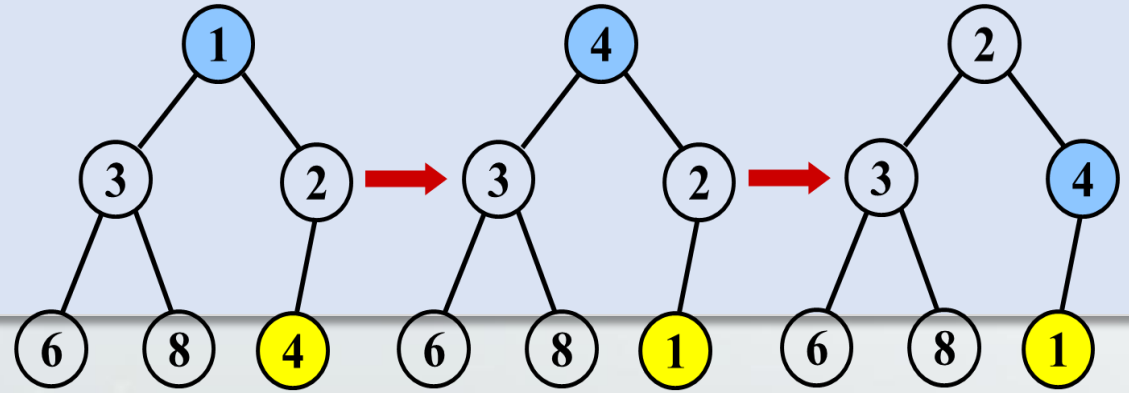
堆排序

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int i, k, n, t, a[200];
    scanf("%d", &n);
    for(i = 1; i <= n; i++)
        scanf("%d", &a[i]);
    //从小到大排列，需要构造初始大顶堆
    for(i = 2; i <= n; i++) {//开始堆有唯一元素a[1]
        k = i; //新添加元素，在最后i
        while(k > 0) {
            if(k/2 <= 0 || a[k] <= a[k/2])//满足堆定义
                break;
            swap(a[k], a[k/2]);//与父结点交换
            k = k/2;
        }
    }
```



```
    //排序,不断的把最大元交换到最后
    for(i = n; i > 0; i--) {
        swap(a[1], a[i]); //顶交换到最后
        k = 1;
        while(k < i) {//调整，使满足堆定义
            t = k;
            if(2*k < i && a[2*k] > a[t])        t = 2*k;
            if(2*k+1 < i && a[2*k+1] > a[t])    t = 2*k+1;
            if(t == k)  break;
            swap(a[k], a[t]);
            k = t;
        }
    }
    for(i = 1; i <= n; i++)
        printf("%d ", a[i]);
    return 0;
}
```

```
E:\Projects\C_C++\temp\bin\Debug\temp.exe
5
78 32 189 23 45
23 32 45 78 189
Process returned 0 (0x0)   execution
```

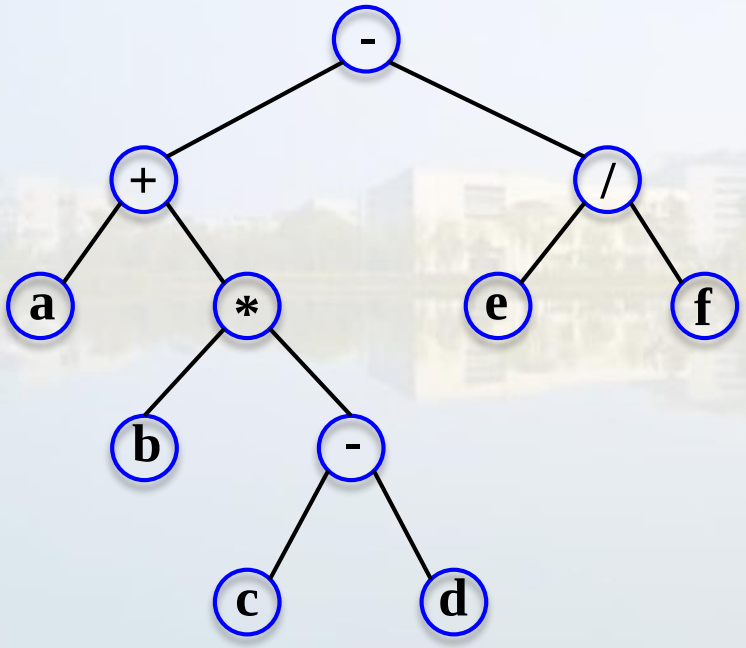


问 题

如果一个文件内存储了10亿个商品的销量数据，请你在其中找出前1000大的数据。

表达式树

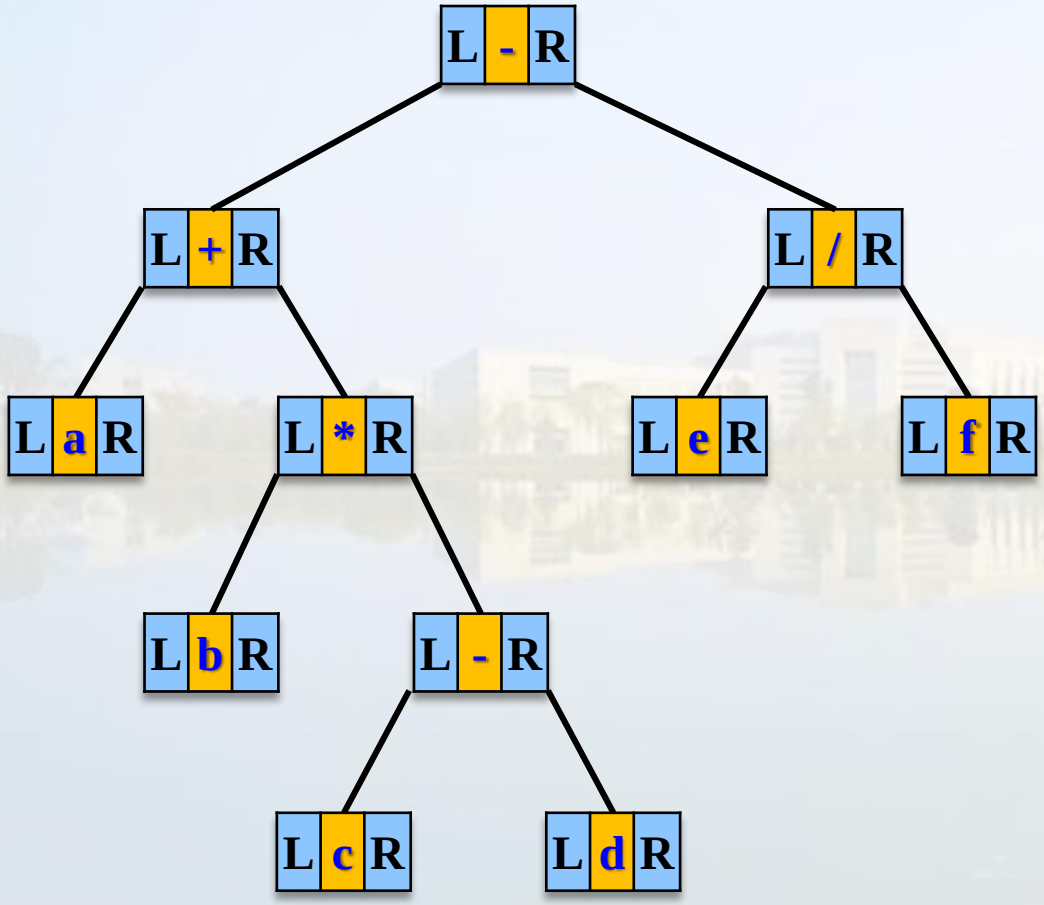
假设一个表达式只含单字符、四则运算符和圆括号(除开圆括号，表达式字符数不超过1000)，试建立一个表达式树，使得中序遍历的结果恰好是该表达式。下面是a+b*(c-d)-e/f对应的表达式树。



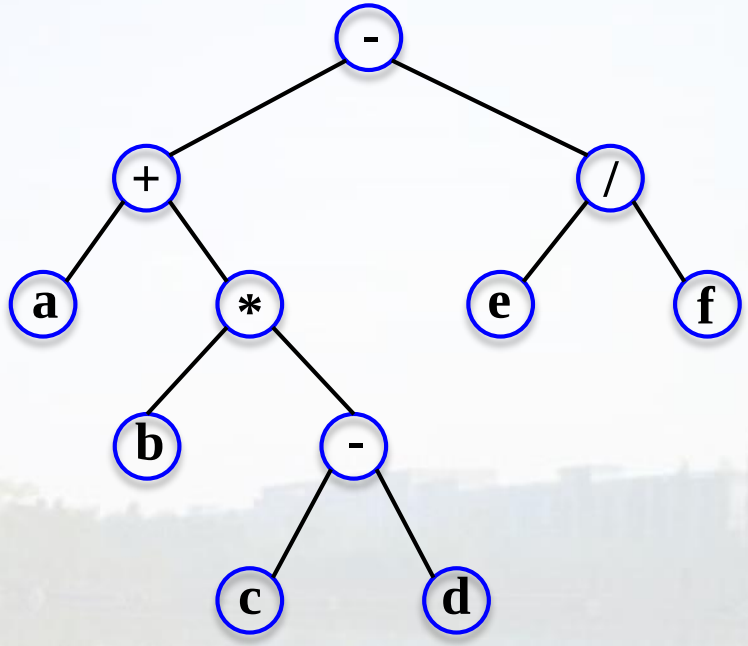
样例输入	样例输出
a+b*(c-d)-e/f	(((a)+((b)*((c)-(d))))-((e)/(f)))

题目分析

a+b*(c-d)-e/f



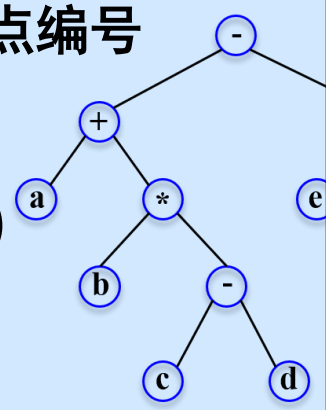
	lch	op	rch
0	1	-	8
1	2	+	3
2	-1	a	-1
3	4	*	5
4	-1	b	-1
5	6	-	7
6	-1	c	-1
7	-1	d	-1
8	9	/	10
9	-1	e	-1
10	-1	f	-1



参考程序

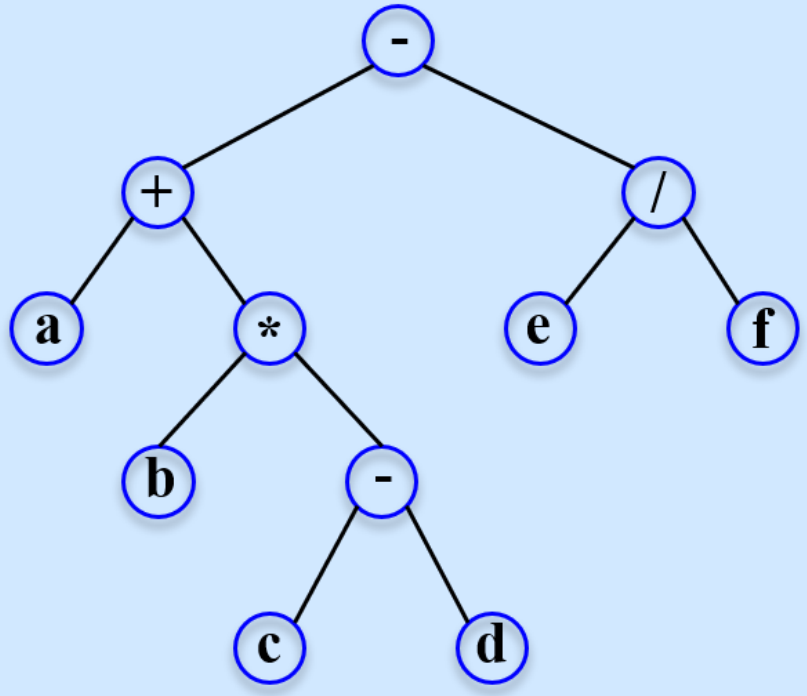
a+b*(c-d)-e/f

```
#include <stdio.h>
#include <string.h>
#define N 1000
int lch[N], rch[N]; //左右儿子结点编号
char op[N]; //表达式字符
int nc; //结点数计数器
int build_tree(char* s, int x, int y)
{
   int i, c1 = -1, c2 = -1, p = 0, u;
   if(y - x == 1)
   {
      u = nc++;
      lch[u] = rch[u] = -1;//表示是叶子
      op[u] = s[x];
      return u;
   }
```




```
for(i = x; i < y; i++) {
      switch(s[i]) {
         case '(': p++; break;
         case ')': p--; break;
         case '+':
         case '-':  if(p == 0) c1 = i; break;
         case '*':
         case '/': if(p == 0) c2 = i; break;
      }
   }
   if(c1 == -1) c1 = c2; //括号外没有加减号
   if(c1 == -1) //整个表达式外有一个括号
      return build_tree(s, x + 1, y - 1);
   u = nc++;
   op[u] = s[c1];
   lch[u] = build_tree(s, x, c1);
   rch[u] = build_tree(s, c1 + 1, y);
   return u;
}
```

```
void middfs(int u)
{
    if(u == -1)  return;
    printf("(");
    middfs(lch[u]);
    printf("%c", op[u]);
    middfs(rch[u]);
    printf(")");
}
int main()
{
    char s[10 * N];
    while(scanf("%s", s) == 1)
    {
        nc = 0;
        build_tree(s, 0, strlen(s));
        middfs(0);
    }
    return 0;
}
```



样例输入	样例输出
a+b*(c-d)-e/f	(((a)+((b)*((c)-(d))))-((e)/(f)))

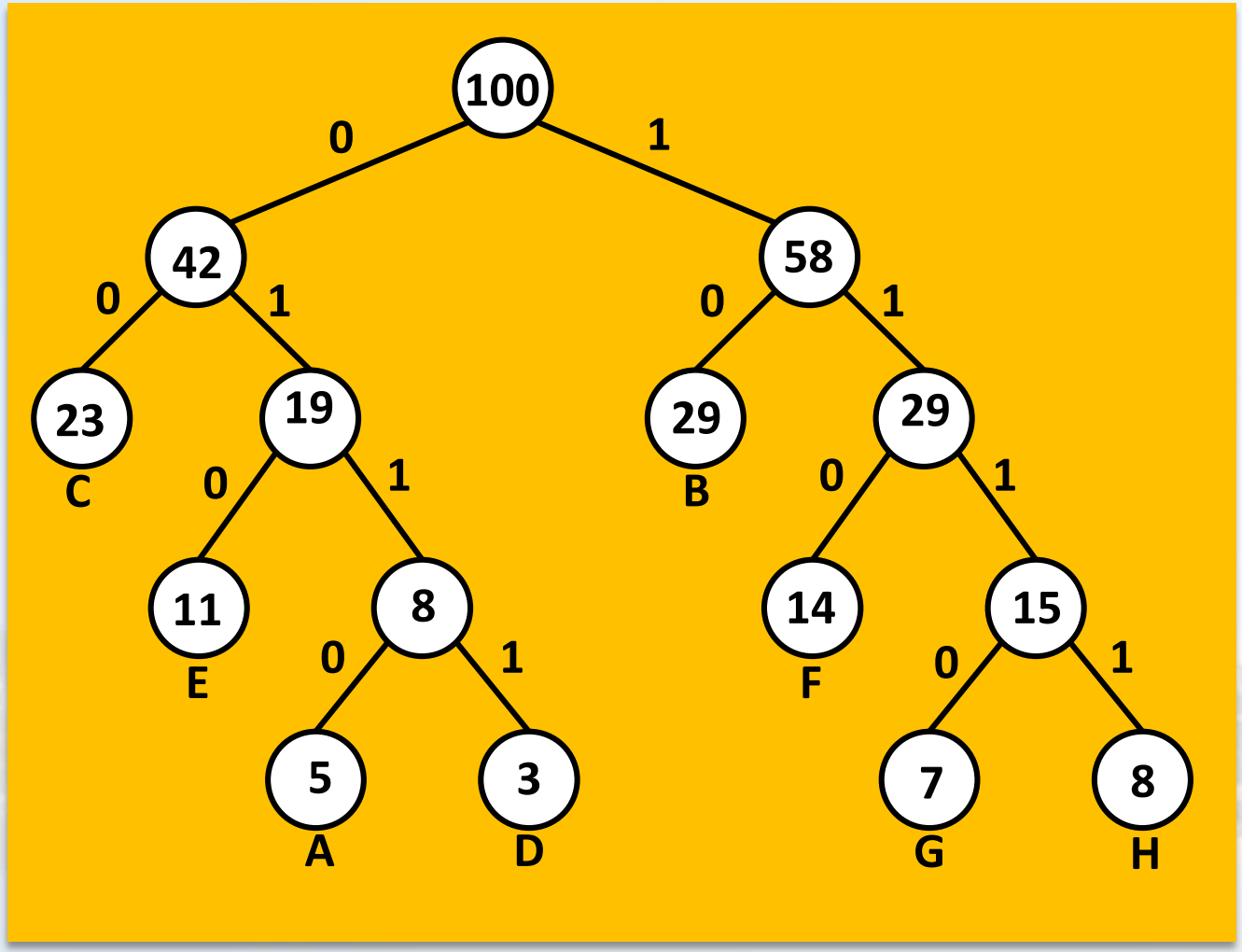
问题：文本编码问题

利用最优二叉树(也称哈夫曼树)可以对文本进行编码. 例如一个文本内只出现了A,B,C,D,E,F,G,H八种符号, 并且各自出现的频数如下表:

字符	A	B	C	D	E	F	G	H
频数	5	29	23	3	11	14	7	8
ASCII	01000001	01000010	01000011	01000100	01000101	01000110	01000111	01001000

容易算出该文本占用字节数: 5+29+23+3+11+14+7+8 = 100.

可以按以下方法构造最优二叉树, 实现这些字符的重新编码.



D	A	G	H	E	F	C	B
3	5	7	8	11	14	23	29

G	H	1	E	F	C	B
7	8	8	11	14	23	29

1	E	F	2	C	B
8	11	14	15	23	29

F	2	3	C	B
14	15	19	23	29

3	C	B	4
19	23	29	29

B	4	5
29	29	42

5	6
42	58

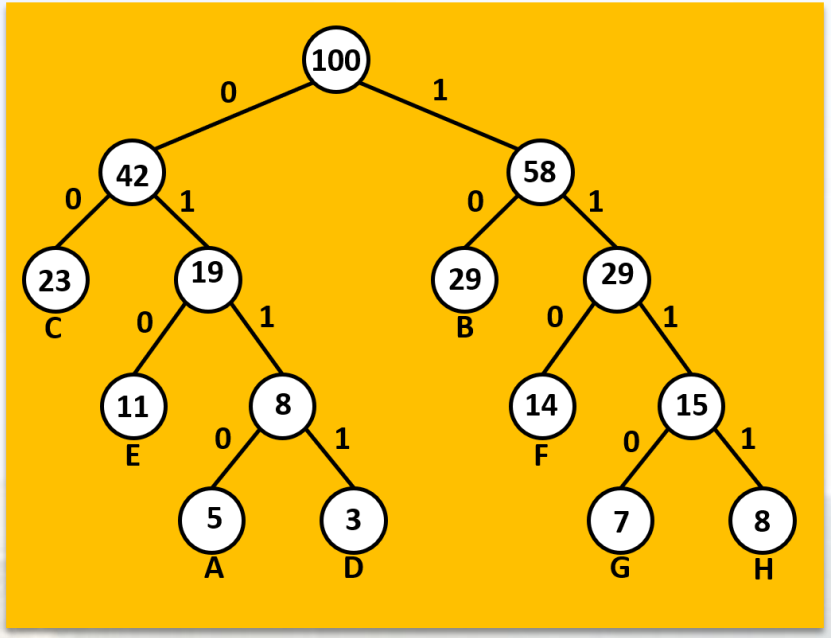
7
100

字符	A	B	C	D	E	F	G	H
频数	5	29	23	3	11	14	7	8
编码	0110	10	00	0111	010	110	1110	1111

字符	A	B	C	D	E	F	G	H
频数	5	29	23	3	11	14	7	8
编码	0110	10	00	0111	010	110	1110	1111

该文本占用的字节数为34字节.

(5*4+29*2+23*2+3*4+11*3+14*3+7*4+8*4)/8 = 33.875



如果文本文件中的字符序列为:

ABCDFF……

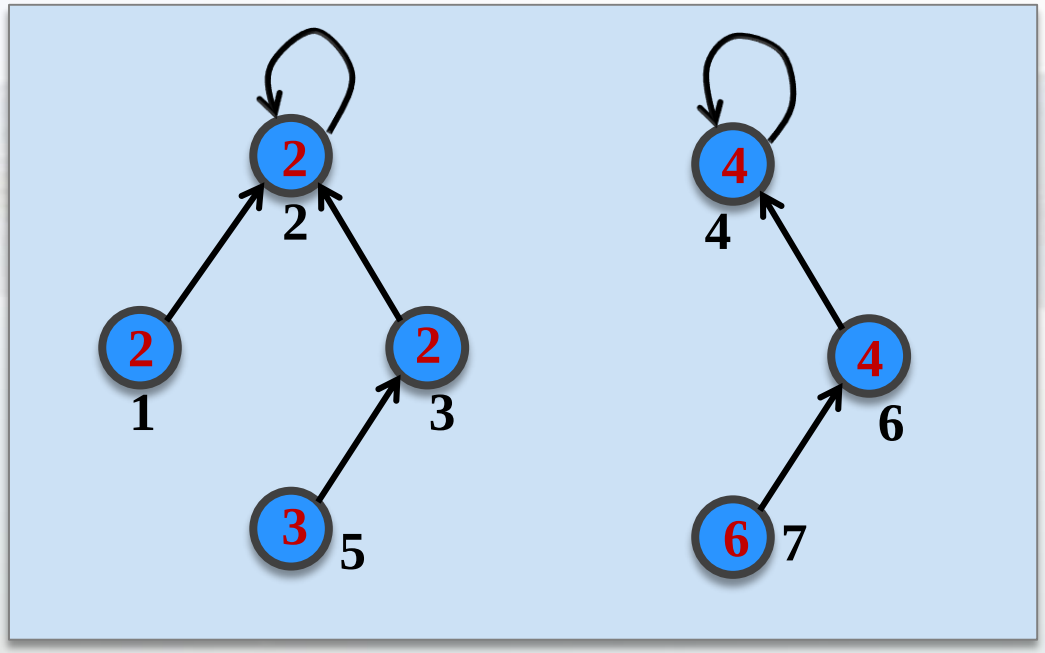
则该文本文件的编码文件的二进制序列为: 0110100001111101110……

反过来, 由此二进制序列, 结合上面的哈夫曼树, 可以解码得到对应的字符序列.

你的任务是选择合适的数据结构, 实现上面的算法过程, 输出字符的新编码.

五、并查集

- 并查集(Union Find Set)是一种用于处理分离集合的抽象数据类型. 当给出两个元素的一个无序对(a,b)时, 需要快速合并a和b分别所在的集合, 这期间需要反复查找某元素所在的集合, “并”、“查”和“集”三字由此而来.
- 例: 一个城市居住着n个市民, 已知一些人互为朋友, 而且朋友的朋友也是朋友. 现请你根据给出的若干朋友关系, 求出最大的一个朋友圈的人数.
- 并查集的主要操作:
  - 初始化
  - 查找
  - 合并
- 并查集的主要实现方法: 树结构.



亲戚

或许你并不知道, 你的某个朋友是你的亲戚. 他可能是你的曾祖父的外公的女婿的外甥女的表姐的孙子. 如果能得到完整的家谱, 判断两个人是否是亲戚应该是可行的, 但如果两个人的最近公共祖先与他们相隔好几代, 使得家谱十分庞大, 那么检验亲戚关系实非人力所能及. 在这种情况下, 最好的帮手就是计算机.

为了将问题简化, 你将得到一些亲戚关系的信息, 如Marry和Tom是亲戚, Tom和Ben是亲戚等, 从这些信息中, 你可以推出Marry和Ben是亲戚. 请写一个程序, 对于我们的关于亲戚关系的提问, 以最快的速度给出答案.

- **Standard Input**

输入由两部分组成.

第一部分的第一行是以空格隔开的n,m. n为问题涉及的人数(1<=n<=20000), 这些人的编号为1,2,3,...,n. 下面有m行(1<=m<=1000000), 每行有两个数a_i和b_i, 表示已知a_i和b_i是亲戚.

第二部分的第一行为q, 表示提问次数(1 <= q <= 1000000). 下面的q行每行有两个数c_i和d_i, 表示询问c_i和d_i是否为亲戚.

- **Standard Output**

对于每个提问, 输出一行, 若c_i和d_i为亲戚, 则输出“Yes”, 否则输出“No”.

- **Samples**

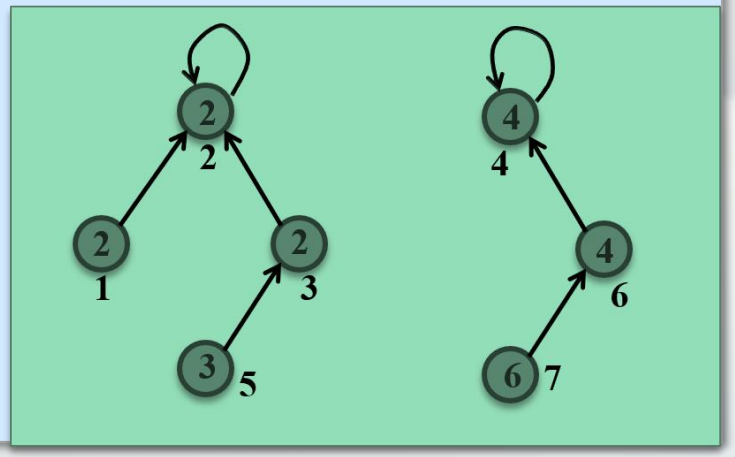
Input	Output
10 7 2 4 5 7 1 3 8 9 1 2 5 6 2 3 3 3 4 7 10 8 9	Yes No Yes

题目分析

- 将每个人抽象在一个点, 若两人有由m条边给出的亲戚关系, 则连一条边, 则得到一个由n个点m条边构成的图模型.
- 同是亲戚的人显然构成一个连通块, 不是亲戚关系的显然不在同一个连通块.
- 判断两个人是否是亲戚关系, 关键是判定两个人是否在一个连通块. 实际上就是判断两人是否位于同一个集合. 用并查集非常合适.
- 本题不仅需要判断, 显然读入边时还需要一个合并过程.

参考程序

```
#include <stdio.h>
#define N 10010
int a[N], n, m, q;
int find(int u) {
    return (a[u] == u)? u : (a[u] = find(a[u]));
}
int main() {
    scanf("%d%d", &n, &m);
    for(int i = 1; i <= n; i++)
        a[i] = i;
    for(int i = 1; i <= m; i++)  {
        int u, v;
        scanf("%d%d", &u, &v);
        int s = find(u);
        int t = find(v);
        if(s != t)   a[t] = s;
    }
    scanf("%d", &q);
    for(int i = 1; i <= q; i++) {
        int u, v;
        scanf("%d%d", &u, &v);
        int s = find(u);
        int t = find(v);
        if(s == t)
            printf("Yes\n");
        else
            printf("No\n");
    }
    return 0;
}
```



六、快速排序

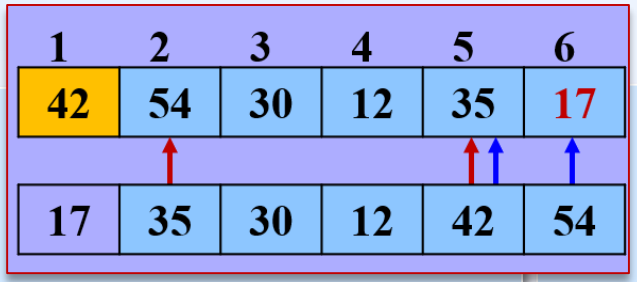
- 快速排序(Quicksort)是对冒泡排序的一种改进, 由C. A. R. Hoare在1962年提出。
- 基本思想是: 通过一趟排序将要排序的数据分割成独立的两部分, 其中一部分的所有数据都比另外一部分的所有数据都要小, 然后再按此方法对这两部分数据分别进行快速排序. 整个排序过程可以递归进行, 以此达到整个数据变成有序序列.
- 算法的关键是先随机选择其中一个元素作为基准元素, 把比它小的放在左边, 比它的大的放在右边, 从而完成一次划分.

Input	Output
6 2 4 5 1 3 7	1 2 3 4 5 7

题目分析


1 2 3 4 5 6 7 8 9 10 11 12 13 14
42 54 98 68 63 83 94 55 35 12 63 30 17 97
55 54 98 68 63 83 94 42 35 12 63 30 17 97
17 54 30 12 35 42 55 94 83 63 63 68 98 97
42 54 30 12 35 17
17 35 30 12 42 54
12 35 30 17
12 35 30 17
17 30 35
17 30 35
35 30
30 35
68 83 63 63 94 98 97
63 63 68 83 94 98 97
63 63
63 63
83 94 98 97
83 94 98 97
97 98 94
94 97 98

参考程序




```
#include <bits/stdc++.h>
using namespace std;
int a[100005], n;
void quick_sort(int left, int right)
{
    if(left >= right) return;
    if(left < right)
    {
        swap(a[left], a[rand()%(right-left+1)+left]);
        int i = left, j = right, x = a[left];
        while(i < j)
        {
            while(i < j && a[j] >= x) j--;
            if(i < j) a[i++] = a[j];
            while(i < j && a[i] <= x) i++;
            if(i < j) a[j--] = a[i];
        }
        a[i] = x;
        quick_sort(left, i-1);
        quick_sort(i+1, right);
    }
}
int main()
{
    cin >> n;
    for(int i = 1; i <= n; i++)
        cin >> a[i];
    srand(int(time(0)));
    quick_sort(1, n);
    for(int i = 1; i <= n; i++)
        cout << a[i] << " ";
    cout << endl;
    return 0;
}
```