二 叉 堆

【知识要点】

1、二叉堆定义

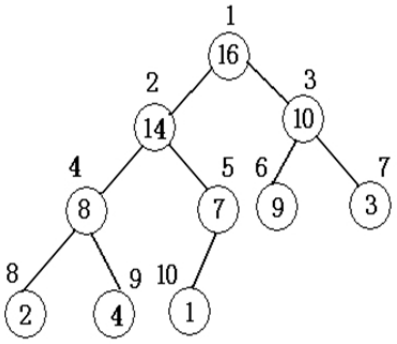
2、二叉堆的基本操作

3、二叉堆的基础应用

一、　二叉堆的基本知识

1、二叉堆定义

定义:堆是一棵完全二叉树，对于每一个非叶子结点，它的权值都不大于（或不小于）左右孩子的权值，我们称这样的堆为小根堆（或大根堆）。



(a)

1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1

(b)

2、二叉堆插入元素

首先将元素 x 放到堆中的最后一个位置(即最底层最右边的位置),然后不断地把 x 往上调整，直到 x 调不动为止(即大于它现在的父亲,或者 x 处于根结点)。

参考代码：

```
void push(int x){  //向上堆化
    int now;
    h[++h_siz]=x;
    now = h_siz;
    while(now>1&&h[now]< a[now>>1]){
         swap(h[now],h[now>>1];
         now = now>>1;
    }
}
```

3、删除二叉堆顶部元素

首先把顶部元素和最后一个位置的元素交换,然后删去最后一个位置,这样 w 上的元素就被删除了。接着把位置 w 上的新元素不断**下调**，直到满足堆的性质。

参考代码：

```
void pop(){
    h[1]=h[siz--];
    int now=1;
    while(now*2<=siz){
        int son=now*2;
        if(son+1<=siz && h[son+1]<h[son])son++;
        if(h[now]<h[son])break;
```

```
        swap(h[son],h[now]);
        now=son;
    }
}
```

4、STL 优先队列

priority_queue 是 STL 提供的优先队列，可以直接实现堆的操作。常见有 4 个函数：

q.push(a)：使 a 入队。

q.top()：返回优先级最高的元素，但不会移除元素。

q.pop()：移除优先级最高的元素。该函数没有返回值。

q.empty()：判断队列是否为空。

定义方法：priority_queue<元素类型，容器类型，比较器> 队列名

最小值优先出队：priority_queue< int , vector< int>,greater< int> > q;

最大值优先出队：priority_queue< int , vector< int>,less< int> > q;

其中第一个参数为必须，后续可以使用默认值，容器默认为 vector，比较器为<（默认为大根堆）。

二、　二叉堆的基础应用

1、 合并果子(fruit) G1049

【题意】给出 N 个数字，每一次取两个数字出来求和，将结果放回序列直到序列只剩一个数字。每次的代价为取出数字的总和。问合并为一个数字总计代价最小。

【分析】

将这个问题换一个角度描述：给定 n 个叶结点，每个结点有一个权值 W[i]，将它们中两个、两个合并为树，假设每个结点从根到它的距离是 D[i]，使得最终∑(wi * di)最小。

于是，这个问题就变为了经典的 Huffman 树问题。Huffman 树的构造方法如下：

（1）从森林里取两个权和最小的结点；

（2）将它们的权和相加，得到新的结点，并且把原结点删除，将新结点插入到森林中；

（3）重复(1)~(2)，直到整个森林里只有一棵树。

通过上述算法可知，每次需要取出最小的节点，二叉堆最适合这个操作，具体做法为：每次取两次堆顶元素，然后将他们求和为 temp，再将 temp 插入到堆中，如此反复直到堆只剩下一个数字。

【参考代码】

```
#include<bits/stdc++.h>
using namespace std;
int h[100010],siz=0;
int n,x,ans=0;
void push(int x){
    int now;
    h[++siz]=x;
    now=siz;
    while(now>1 && h[now] < h[now>>1]){
        swap(h[now],h[now>>1]);
        now=now>>1;
    }
}
void pop(){
```

```
    h[1]=h[siz--];
    int now=1;
    while(1){
        if(now*2 > siz )break;
        int son=now*2;
        if(son+1<=siz && h[son+1]<h[son])son++;//
        if(h[now]<h[son])break;
        swap(h[son],h[now]);
        now=son;
    }
}
int main(){
    cin>>n;
    for(int i=1;i<=n;i++){
        cin>>x;push(x);
    }
    for(int i=1;i<n;i++){
        x=h[1];pop();
        int y=h[1];pop();
        ans+=x+y;
        push(x+y);
    }
    cout<<ans;
    return 0;
}
```

2、 P2584 序列合并

【题意】有两个长度都为 N 的序列 A,B，在 A 和 B 中各取一个数相加可以得到 N*N 个和，求着 N*N 个和中最小的前 N 个。N<=1e5.

【分析】由于 2 个序列都是升序的，则选择的组合数有单调性。可以利用这个特点来构造算法：

因为 a[1]+b[i]<=a[1]+b[i+1]同时小于 a[2]+b[i]。所以可以用堆来维护，每次取出一个数 a[i]+b[j]，

就加入一个后面的数(a[i+1]+b[j])进去。

【参考代码】

```
#include<bits/stdc++.h>
using namespace std;
int read(){
    int x=0;char ch=getchar();
    while(ch<'0'||ch>'9')ch=getchar();
    while(ch>='0'&&ch<='9'){
        x=x*10+ch-'0';ch=getchar();
    }return x;
}
struct node{
```

```
    int id,s;
    node(int _id=0,int _s=0):id(_id),s(_s){}
    bool operator<(const node&t)const{
        return s<t.s;
    }
};
struct Heap{
    int siz;
    node h[100010];
    node top(){
        return h[1];
    }
    void push(node x){
        int now;
        h[++siz]=x;
        now=siz;
        while(now>1 && h[now] < h[now>>1]){
            swap(h[now],h[now>>1]);
            now=now>>1;
        }
    }
    void pop(){
        h[1]=h[siz--];
        int now=1;
        while(1){
            if(now*2 > siz )break;
            int son=now*2;
            if(son+1<=siz && h[son+1]<h[son])son++;//
            if(h[now]<h[son])break;
            swap(h[son],h[now]);
            now=son;
        }
    }
};
Heap h;
node x;
int a[100010],b[100010],n;
int main(){
    n=read();
    for(int i=1;i<=n;i++)a[i]=read();
    for(int i=1;i<=n;i++)b[i]=read();
    for(int i=1;i<=n;i++)
        h.push((node){1,a[1]+b[i]});
    for(int i=1;i<=n;i++){
```

```
        x=h.top();h.pop();
        cout<<x.s<<" ";
        x.s=x.s-a[x.id]+a[x.id+1];
        x.id++;
        h.push(x);
    }

}
```

3、 鱼塘钓鱼 P10009

【题意】n 个鱼塘线性分布在一条线上，从起点出发，在 T 时间内最多钓多少与。已知每个鱼塘每分钟能钓鱼的数量，及到鱼塘的距离。

【分析】

我们可以这样想：如果知道了取到最大值的情况下，人最后在第 i 个鱼塘里钓鱼，那么用在路上的时间是固定的，因为我们不会先跑到第 i 个鱼塘里钓一分钟后再返回前面的鱼塘钓鱼，这样把时间浪费在路上显然不划算，再说在你没到某个鱼塘里去钓鱼之前，这个塘里的鱼也不会跑掉（即数量不会减少）。所以这时我们是按照从左往右的顺序钓鱼的，也可以看成路上是不需要时间的，即人可以自由在 1~i 个鱼塘之间来回走，只要尽可能选取钓到的鱼多的地方就可以了，这就是我们的贪心思想。其实，这个贪心思想并不是模拟钓鱼的过程，只是统计出在各个鱼塘钓鱼的次数。程序实现时，只要分别枚举钓鱼的终点鱼塘（从鱼塘 1 到鱼塘 n），每次按照上述贪心思想确定在哪些鱼塘里钓鱼，经过 n 次后确定后最终得到的一定是最优方案。

建立以 fish 为关键字的大根堆，包括能钓到鱼的数量和池塘的编号。然后借助枚举创造条件，实现复杂度为 O（m*nlogn）的算法。

【参考代码】

```
#include<bits/stdc++.h>
using namespace std;
struct Node{
    int fish,lake;
    Node(int fish=0,int lake=0):fish(fish),lake(lake){   }
    bool operator<(const Node& t)const{
        return fish<t.fish;
    }
};

int n,f[101],d[101],t[101];
int ans,m,t1=0,  Time,Max=0;
int main(){
    cin>>n;
    for(int i=1;i<=n;i++)cin>>f[i];
    for(int i=1;i<=n;i++)cin>>d[i];
    for(int i=1;i<n;i++)cin>>t[i];
    cin>>m;
    for(int k=1;k<=n;k++){
        Time=m-t1;
```

```
        ans=0;
        priority_queue<Node> heap;
        for(int i=1;i<=k;i++)
           heap.push(Node(f[i],i));
       while(Time>0&&heap.top().fish>0){
           Node a=heap.top();heap.pop();
           ans+=a.fish;
           a.fish-=d[a.lake];
           heap.push(a);
           Time--;
       }
       if(ans>Max) Max=ans;
       t1+=t[k];
    }
    cout<<Max;
    return 0;
}
```