目录

[TOC]

1 基础算法

1.1 二分模板

```
//ans>=mi,ans<=ma
l=mi,r=ma,mid;
while(l<=r)
{
    mid=(l+r)>>1;
    if(isOk(mid))
    {
        //求最大值
        l=mid+1;
        ans=max(ans,mid);
        //求最小值
        r=mid-1;
        ans=min(ans,mid);
    }
    else
    {
        //求最大值
```

```
        r=mid-1;
        //求最小值
        l=mid+1;
    }
}
```

1.2 1933 二分偏好模板

```
while(l<r){
    int mid=(l+r)>>1;
    if(a[mid]>=x)
        r=mid;
    else
        l=mid+1;//最后 l 就是答案，区间 [l,r)
}
```

1.3 三分求函数极值点

1.3.1 实数域

这里算的极大值点，极小值类似分析即可，这里算的是函数极值点。

```
const double EPS = 1e-9;
while(r - l < EPS) {
    double lmid = l + (r - l) / 3;
    double rmid = r - (r - l) / 3;
    lans = cal(lmid),rans = cal(rmid);

    // 求凹函数的极小值
    if(lans <= rans) r = rmid;
    else l = lmid;

    // 求凸函数的极大值
    if(lans >= rans) l = lmid;
    else r = rmid;
}
// 输出 l 或 r 都可
cout << l << endl;
```

1.3.2 类推到整数域：

```
double lans,rans
int l = 1,r = 100;
while(l < r) {
    int lmid = l + (r - l) / 3; // l + 1/3 区间大小
    int rmid = r - (r - l) / 3;  // r - 1/3 区间大小
    lans = cal(lmid),rans = cal(rmid);

    // 求凹函数的极小值
    if(lans <= rans) r = rmid - 1;
    else l = lmid + 1;

    // 求凸函数的极大值
    if(lans >= rans) l = lmid + 1;
    else r = rmid - 1;
}

cout << l << endl;
```

1.4 尺取模板

```
//ans: 答案，n: 个数，t 给定标准 ，求这 n 个数中和大于等于 t 的最小连续序列。
// 反过来，对于每个 l，求和 <t 的最长连续序列，然后再加上后面一个数
void ruler(){
    int l=0,r=0,sum=0;//注意区间 [0, n)
    int ans=INF;
    while(1){
        while(r<n&&sum+a[r]<t){//如果下一个满足约束条件，就走一步
            sum+=a[r++];
        }
        //  跳出循环的是一个满足的区间,r 指向第一个不满足的约束条件的位置
        if(r==n)break;// 区间 [0, n)，注意跳出的条件
        ans=min(ans,r-l+1);// 因为 r 指向第一个不满足的位置，所以这里要算上这个位置
        sum-=a[l++];//l 移动
    }
    printf("%d\n",ans);
}
```

1.5 进制转化

1.5.1 十进制转换 k 进制

```
#include<cstdio>
#include<iostream>
#include<ctime>
char a[1000];
using namespace std;
int main()
{
    int y=0,k,n,x;
    char z='A';
    scanf ("%d %d",&n,&x);
    while (n!=0)
    {
          y++;
             a[y]=n%x;
          n=n/x;
          if (a[y]>9) a[y]=z+(a[y]-10);
          else a[y]=a[y]+'0';
    }
    for (int i=y;i>0;i--)
    printf ("%c",a[i]);
    return 0;
}
```

1.5.2 M 进制转十进制

```
#include<cstdio>
#include<iostream>
#include<cstdlib>
#include<cstring>
char a[10000];
using namespace std;
int main()
{
   int n,m;
   int f=0;
   scanf ("%s%d",a,&m);
   for (int i=0;i<strlen(a);i++)
```

```
    {
        f*=m;
        if (a[i]=='A'||a[i]=='B'||a[i]=='C'||a[i]=='D'||a[i]=='E'||a[i]=='F')
        {
            f=f+(a[i]-'A'+10);
        }
        else
        {
            f=f+(a[i]-'0');
        }
    }
    printf ("%d",f);
    return 0;
}
```

再来个高端操作：利用短除法计算高精度乘除法实现进制转换

例题：https://www.acwing.com/problem/content/126/

```
#include<iostream>
#include<algorithm>

#define ll long long

using namespace std;
const int maxn = 1e4 + 10;

int x[maxn],y[maxn];

int main()
{
    int t;
    cin >> t;
    while(t--)
    {
        int n,m;
        cin >> n >> m;
        string s,ans;
        cin >> s;
        int len = s.length();
        //-----------------------------------------------------------------核心
        vector<int> num;
        for(int i = len - 1;i >= 0;i--)
```

```
        {
            if(s[i] >= '0' && s[i] <= '9') num.push_back(s[i] - '0');
            else if(s[i] >= 'A' && s[i] <= 'Z') num.push_back(s[i] - 'A' + 10);
            else if(s[i] >= 'a' && s[i] <= 'z') num.push_back(s[i] - 'a' + 36);
        }

        vector<int> res;
        while(num.size())
        {
                int r = 0;
                for(int i = num.size() - 1;i >= 0;i--)
                {
                    num[i] += r * n;
                    r = num[i] % m;
                    num[i] /= m;
                }
                res.push_back(r);
                while(num.size() && !num.back()) num.pop_back();
        }
        reverse(res.begin(),res.end());
        for(auto x : res)
        {
            if(x <= 9) ans += char(x + '0');
            else if(x >= 10 && x <= 35) ans += char(x + 'A' - 10);
            else if(x >= 36) ans += char(x + 'a' - 36);
        }
        //-----------------------------------------------------------------
        cout << n << ' ' << s << endl;
        cout << m << ' ' << ans << endl;
        cout << endl;
    }
    return 0;
}
```

1.6 高精度板子

```
//注意: 只支持正整数
#include <algorithm> // max
#include <cassert>   // assert
#include <cstdio>    // printf,sprintf
#include <cstring>   // strlen
```

```cpp
#include <iostream>  // cin,cout
#include <string>    // string 类
#include <vector>    // vector 类
struct BigInteger {
    typedef unsigned long long LL;

    static const int BASE = 100000000;
    static const int WIDTH = 8;//压位存储，一个空间存 7 位
    vector<int> s;

    BigInteger& clean(){while(!s.back()&&s.size()>1)s.pop_back(); return *this;}//去掉前导 0
    BigInteger(LL num = 0) {*this = num;}
    BigInteger(string s) {*this = s;}
    BigInteger& operator = (long long num) {
        s.clear();
        do {
            s.push_back(num % BASE);
            num /= BASE;
        } while (num > 0);
        return *this;
    }
    BigInteger& operator = (const string& str) {
        s.clear();
        int x, len = (str.length() - 1) / WIDTH + 1;
        for (int i = 0; i < len; i++) {
            int end = str.length() - i*WIDTH;
            int start = max(0, end - WIDTH);
            sscanf(str.substr(start,end-start).c_str(), "%d", &x);
            s.push_back(x);
        }
        return (*this).clean();
    }

    BigInteger operator + (const BigInteger& b) const {
        BigInteger c; c.s.clear();
        for (int i = 0, g = 0; ; i++) {
            if (g == 0 && i >= s.size() && i >= b.s.size()) break;
            int x = g;
            if (i < s.size()) x += s[i];
            if (i < b.s.size()) x += b.s[i];
            c.s.push_back(x % BASE);
            g = x / BASE;
```

```
        }
        return c;
    }
     // 减数不能大于被减数
    BigInteger operator - (const BigInteger& b) const {
        assert(b <= *this);
        BigInteger c; c.s.clear();
        for (int i = 0, g = 0; ; i++) {
            if (g == 0 && i >= s.size() && i >= b.s.size()) break;
            int x = s[i] + g;
            if (i < b.s.size()) x -= b.s[i];
            if (x < 0) {g = -1; x += BASE;} else g = 0;
            c.s.push_back(x);
        }
        return c.clean();
    }
    BigInteger operator * (const BigInteger& b) const {
        int i, j; LL g;
        vector<LL> v(s.size()+b.s.size(), 0);
        BigInteger c; c.s.clear();
        for(i=0;i<s.size();i++) for(j=0;j<b.s.size();j++) v[i+j]+=LL(s[i])*b.s[j];
        for (i = 0, g = 0; ; i++) {
            if (g ==0 && i >= v.size()) break;
            LL x = v[i] + g;
            c.s.push_back(x % BASE);
            g = x / BASE;
        }
        return c.clean();
    }
    BigInteger operator / (const BigInteger& b) const {
        assert(b > 0);  // 除数必须大于 0
        BigInteger c = *this;       // 商: 主要是让 c.s 和 (*this).s 的 vector 一样大
        BigInteger m;               // 余数: 初始化为 0
        for (int i = s.size()-1; i >= 0; i--) {
            m = m*BASE + s[i];
            c.s[i] = bsearch(b, m);
            m -= b*c.s[i];
        }
        return c.clean();
    }
    BigInteger operator % (const BigInteger& b) const { //方法与除法相同
        BigInteger c = *this;
```

```
        BigInteger m;
        for (int i = s.size()-1; i >= 0; i--) {
            m = m*BASE + s[i];
            c.s[i] = bsearch(b, m);
            m -= b*c.s[i];
        }
        return m;
    }
    // 二分法找出满足 bx<=m 的最大的 x
    int bsearch(const BigInteger& b, const BigInteger& m) const{
        int L = 0, R = BASE-1, x;
        while (1) {
            x = (L+R)>>1;
            if (b*x<=m) {if (b*(x+1)>m) return x; else L = x;}
            else R = x;
        }
    }
    BigInteger& operator += (const BigInteger& b) {*this = *this + b; return *this;}
    BigInteger& operator -= (const BigInteger& b) {*this = *this - b; return *this;}
    BigInteger& operator *= (const BigInteger& b) {*this = *this * b; return *this;}
    BigInteger& operator /= (const BigInteger& b) {*this = *this / b; return *this;}
    BigInteger& operator %= (const BigInteger& b) {*this = *this % b; return *this;}

    bool operator < (const BigInteger& b) const {
        if (s.size() != b.s.size()) return s.size() < b.s.size();
        for (int i = s.size()-1; i >= 0; i--)
            if (s[i] != b.s[i]) return s[i] < b.s[i];
        return false;
    }
    bool operator >(const BigInteger& b) const{return b < *this;}
    bool operator<=(const BigInteger& b) const{return !(b < *this);}
    bool operator>=(const BigInteger& b) const{return !(*this < b);}
    bool operator!=(const BigInteger& b) const{return b < *this || *this < b;}
    bool operator==(const BigInteger& b) const{return !(b < *this) && !(b > *this);}
};

ostream& operator << (ostream& out, const BigInteger& x) {
    out << x.s.back();
    for (int i = x.s.size()-2; i >= 0; i--) {
        char buf[20];
        sprintf(buf, "%08d", x.s[i]);
        for (int j = 0; j < strlen(buf); j++) out << buf[j];
```

```
    }
    return out;
}

istream& operator >> (istream& in, BigInteger& x) {
    string s;
    if (!(in >> s)) return in;
    x = s;
    return in;
}
```

1.7 尺取法

尺取法：顾名思义，像尺子一样取一段，尺取法通常是对数组保存一对下标，即所选取的区间的左右端点，然后根据实际情况不断地推进区间左右端点以得出答案。**尺取法比直接暴力枚举区间效率高很多**，尤其是数据量大的时候，所以说尺取法是一种高效的枚举区间的方法，是一种技巧，**一般用于求取有一定限制的区间个数或最短的区间**等等。当然任何技巧都存在其不足的地方，有些情况下尺取法不可行，无法得出正确答案，所以要先判断是否可以使用尺取法再进行计算

使用尺取法时应清楚以下四点：1、什么情况下能使用尺取法？2、何时推进区间的端点？3、如何推进区间的端点？4、何时结束区间的枚举？

尺取法通常适用于**选取区间有一定规律，或者说所选取的区间有一定的变化趋势的情况**，通俗地说，在对所选取区间进行判断之后，我们可以明确如何进一步有方向地推进区间端点以求解满足条件的区间，如果已经判断了目前所选取的区间，但却无法确定所要求解的区间如何进一步得到根据其端点得到，那么尺取法便是不可行的。首先，明确题目所需要求解的量之后，区间左右端点一般从最整个数组的起点开始，之后判断区间是否符合条件在根据实际情况变化区间的端点求解答案。

例题：https://ac.nowcoder.com/acm/contest/5674/F

参考 AC 代码：

```
#include<iostream>
#include<cstdio>
#include<cstring>
#include<cmath>
#include<algorithm>
using namespace std;
const int maxn = 2e6 + 17;
int vis[maxn] = {0};
struct Node{
    int x,line;
}N[maxn];
bool cmd(Node a,Node b)
{
```

```
    return a.x < b.x;
}
int main()
{
    int n,m,num = 0;
    scanf("%d%d",&n,&m);
    for(int i = 1;i <= n;i++){
        int k;
        scanf("%d",&k);
        for(int j = 1;j <= k;j++){
            scanf("%d",&N[++num].x);
            N[num].line = i;
        }
    }
    sort(N + 1,N + 1 + num,cmd);
    int l = 1,r = m,temp = 0,ans = 0x3f3f3f3f;
    for(int i = l;i <= r;i++){
        if(!vis[N[i].line]) {
            vis[N[i].line]++;
            temp++;
        }
    }
    while(temp < m){
        r++;
        if(!vis[N[r].line]) {
            vis[N[r].line]++;
            temp++;
        }
    }
    for(l,r;l <= r && r <= num;){
        ans = min(ans,N[r].x - N[l].x);
        vis[N[l].line]--;l++;
        if(vis[N[l - 1].line] == 0) temp--;
        while(r <= num && temp < m){
            r++;
            if(!vis[N[r].line]) {
                vis[N[r].line]++;
                temp;
            }
        }
    }
    printf("%d\n",ans);
```

```
    return 0;
}
```

1.8 最长上升子序列 O（nlogn）板子

这里求的是严格单调的情况

```
#include<bits/stdc++.h>
using namespace std;
#define debug(x) cout<<"###"<<x<<"###"<<endl;
const int Mod=1e4+7,INF=0x3f3f3f3f;
const double eps=1e-8;
typedef long long ll;
const int N=1e5+5;
// qi 表示长度为 i 的上升子序列的末尾最小值为 qi
int q[N];int a[N];
int main(){
    int n;
    cin>>n;
    for(int i=1;i<=n;i++){
        scanf("%d",&a[i]);
    }
    int len=0;q[++len]=a[1];
    for(int i=2;i<=n;i++){
        if(q[len]<a[i]){
            q[++len]=a[i];
        }
        else{
             int j=(lower_bound(q+1,q+1+len,a[i])-q);
            q[j]=a[i];
        }
    }
    cout<<len<<endl;
}
```

1.9 反悔贪心

我们有 n 个任务 (n<=1e5)，每个任务都恰好需要一个单位时间完成，并且每个任务都有两个属性——截止日期和价值。在截止日期前完成任务就可以得到这个任务产生的价值。每个单位时间我们都可以选择这 n 个任务当中还没有做过的并且还没有截止的任务中的一个去完成，问我们最后最多能得到多少价值呢?

- 将性价比低的选择替换掉

```cpp
#include<bits/stdc++.h>
using namespace std;
#define debug(x) cout<<"###"<<x<<"###"<<endl;
typedef long long ll;
const double eps=1e-8;
const int INF=0x3f3f3f3f;
struct node{
    int t;
    ll v;//t 时间，v 价值
    bool operator < (const node &a)const {
        if(v>a.v){
            return true;
        }
        else return false;//定义价值小根堆
    }
};
const int N=1e5+5;
node a[N];
priority_queue<node> q;//小根堆，按价值排序。
bool cmp(node x,node y){//按照截止时间排序
    return x.t<y.t;
}
int main(){
    int n;
    cin>>n;
    for(int i=1;i<=n;i++){
        scanf("%d%lld",&a[i].t,&a[i].v);
    }
    sort(a+1,a+1+n,cmp);
    ll ans=0;
    for(int i=1;i<=n;i++){
        if(a[i].t>q.size()){
            ans+=a[i].v;
            q.push(a[i]);
        }
        else if(a[i].v>q.top().v){//说明当前这个更优，反悔，除去性价比最低的物品
            ans-=q.top().v;
            ans+=a[i].v;
            q.pop();
            q.push(a[i]);
        }
    }
```

```
    cout<<ans<<endl;
}
```

1.10 归并排序带求逆序对的写法（个人感觉比树状数组好用）

```
#include <stdio.h>
long long a[100005],b[100005];
long long cnt=0;
void m(int l,int r){
    if(l>=r)return ;
    int mid=(l+r)/2;
    int p=l;
    int q=mid+1;
    int pos=l;
    m(l,mid);
    m(mid+1,r);
    while(p<=mid||q<=r){
        if(q>r||(p<=mid&&a[p]<=a[q])){
            b[pos++]=a[p++];
        }
        else {
            cnt+=mid-p+1;
            b[pos++]=a[q++];
        }
    }
    for(int i=l;i<=r;i++){
        a[i]=b[i];
    }
}
int main(){
    int n;
    scanf("%d",&n);
    for(int i=1;i<=n;i++){
        scanf("%lld",&a[i]);
    }
    m(1,n);
    printf("%lld",cnt);
}
```

1.11 树状数组求逆序对

```
#include<iostream>
#include<algorithm>
#include<math.h>
#include<cstdio>
#include<string>
#include<string.h>
#include<list>
#include<queue>
#include<sstream>
#include<vector>
#include<set>
#include<map>
#include<deque>
#include<stack>
using namespace std;
#define debug(x) cout<<"###"<<x<<"###"<<endl;
const int INF=0x3f3f3f3f,Maxn=5e5+10;
typedef long long ll;
#define lowbit(x) ((x)&-(x))
int tree[Maxn],n;
void add(int x,int d){
    while(x<=n){
        tree[x]+=d;
        x+=lowbit(x);
    }
}
ll sum(int x){
    int sum=0;
    while(x>0){
        sum+=tree[x];
        x-=lowbit(x);
    }
    return sum;
}
struct Node{
    ll v,id;
}node[Maxn];
bool cmp(Node a,Node b){
    return a.v<b.v;
}
```

```
int a[Maxn];
int main(){
    while(scanf("%d",&n),n){
        ll cnt=0;
        memset(tree,0,sizeof(tree));
        for(int i=1;i<=n;i++){
            scanf("%lld",&node[i].v);
            node[i].id=i;
        }
        sort(node+1,node+1+n,cmp);
        for(int i=1;i<=n;i++){
            a[node[i].id]=i;
        }
        for(int i=n;i>=1;--i){
            cnt+=sum(a[i]);
            add(a[i],1);
        }
        cout<<cnt<<endl;
    }
}
```