线段覆盖专题

T1： 线段无权，数量最少

题意：给定x轴上的N（0<N<100）条线段，每个线段由它的二个端点 l 和 r 确定。有些线段之间会相互交叠或覆盖。请你编写一个程序，从给出的线段中去掉尽量少的线段，使得剩下的线段两两之间没有内部公共点。

题解:

基础线段覆盖模型 -> 贪心

直接按照右端点从小到大排序。优先选排在前面的。

证明：排序后显然当你选了一个线段后，再要从后面选一个线段，其左断点必须>=你的右端点。右端点当然越小越好

```
#include<bits/stdc++.h>
using namespace std;
struct line
{
    int l, r;

}a[101];
bool operator<(const line& a,const line& b) {
    return a.r < b.r;

}
int main() {
    int n,tot=1;
    cin >> n;
    for (int i = 1; i <= n; i++) {
        cin >> a[i].l >> a[i].r;
        if (a[i].l>a[i].r) swap(a[i].l, a[i].r);
    }
    sort(a + 1, a + n + 1);
    int end = a[1].r;
    for (int i = 2; i <= n; i++) {
        if (a[i].l >= end) {
            tot++;
            end = a[i].r;
    }
    }
    cout << tot << endl;
    return 0;
}
```

T2:线段价值之和最大

题意： 数轴上有n条线段，线段的两端都是整数坐标，坐标范围在0~1000000，每条线段有一个价值，请从n条线段中挑出若干条线段，使得这些线段两两不覆盖（端点可以重合）且线段价值之和最大。

题解： 先排序，再dp。

f[i]表示以线段i为最后选择的线段时的最大权值和，则 $f[i] = max(f[j]) + val[i](1 <= j < i, \text{且线段}i, j\text{不相交})$。

直接dp显然复杂度为$O(n^2)$

$O(nlog_2 n)$做法：

1、我们可以发现，按照右端点排序后的两条线段i,j（设i排在j后面），如果它们不相交，需要满足i的左端点>=j的右端点。线段i的左端点在这轮决策中是确定的，而右端点已近排好了序，满足单调性，所以，我们可以通过二分来决策一下。不过要注意，因为我们还要保持已决策答案的单调性，所以我们需要的是一个类似单调栈的东西来存放 f 数组，在当前决策出的结果大于栈顶的结果时，才把它入栈。最后，栈顶就是答案。

```
var f,a,b,c,w,y:array[0..1000010]of int64;
  n,i,j,t,l,r,m:longint;
  ans:int64;
procedure qs(l,r:longint);
var i,j:longint;
  m,t:int64;
begin
  i:=l; j:=r; m:=b[(l*3+r)>>2];
  repeat
    while b[i]<m do inc(i);
    while b[j]>m do dec(j);
    if i<=j then begin
      t:=a[i]; a[i]:=a[j]; a[j]:=t;
      t:=b[i]; b[i]:=b[j]; b[j]:=t;
      t:=c[i]; c[i]:=c[j]; c[j]:=t;
      inc(i); dec(j);
    end;
  until i>j;
  if l<j then qs(l,j);
  if i<r then qs(i,r);
end;
begin
  read(n);
  for i:=1 to n do
    read(a[i],b[i],c[i]); //左端点，右端点，权值
  qs(1,n);
  t:=1; y[1]:=b[1]; w[1]:=c[1]; f[1]:=c[1];
  for i:=2 to n do begin
    l:=0; r:=t+1;
    while l+1<r do begin
      m:=(l+r)>>1;
      if y[m]>a[i] then r:=m
      else l:=m;
    end;
    f[i]:=w[l]+c[i];
    if f[i]>w[t] then begin
      inc(t); y[t]:=b[i]; w[t]:=f[i];
    end;
  end;
  writeln(w[t]);
end.
```

2、离散化+二分查找，按右端点排序，f[i]代表1~i的满足不覆盖的最大值，对于第i条线段，要么选，要么不选，所以方程为$f[i] = max(f[i-1], f[find(i)] + e[i].v)$;

$find(i)$是从右往左数第一个与i不相交的线段的下标，ans为$f[n]$;

```
#include<bits/stdc++.h>
using namespace std;
struct line
{
    long long l,r,v;
    bool operator <(const line&a)const
    {
        return r<a.r;
    }
}e[1000010];
long long n;
long long b[2000010];
long long f[1000010];
long long ans;
void in(long long &x)
{
    char c=getchar();x=0;
    while(c<'0'||c>'9')c=getchar();
    while(c<='9'&&c>='0')x=x*10+c-'0',c=getchar();
}

long long find(long long x)
{
    long long l=0,r=x-1,mid;
    while(l+1<r)
    {
        mid=(l+r)>>1;
        if(e[mid].r>e[x].l)
        r=mid;
        else
        l=mid;
    }
    if (e[r].r <= e[x].l)
    return r;
    else
    return l;
}

int main()
{
  in(n);
  for(long long i=1;i<=n;i++)
   {
       in(e[i].l),in(e[i].r),in(e[i].v);
       b[i]=e[i].l;
       b[i+n]=e[i].r;
    }
  sort(b+1,b+2*n+1);
  long long m=unique(b+1,b+2*n+1)-b-1;
  for(long long i=1;i<=n;i++)
    {
        e[i].l=lower_bound(b+1,b+m+1,e[i].l)-b;
        e[i].r=lower_bound(b+1,b+m+1,e[i].r)-b;
    }
    sort(e+1,e+n+1);
```

```
    for(long long i=1;i<=n;i++)
     {
         f[i]=max(f[i-1],f[find(i)]+e[i].v);
      }
      cout<<f[n];
      return 0;
}
```

T3：HDU6240 Server (ccpc哈尔滨)

题目大意：

用n条线段覆盖区间$[1,t]$上的整点。每条线段有4个属性(S_i,T_i,A_i,B_i)，表示用第i条线段可以覆盖区间$[Si,Ti]$。若选取线段的集合为S，最后总代价为$\frac{\sum_{i\in S}A_i}{\sum_{i\in S}B_i}$。问区间$[1,t]$全部覆盖时，最小代价为多少?

思路：

分数规划经典模型。

二分答案k，表示代价为k。若k为可行的，则我们需要找到一个合法的S，满足$\frac{\sum_{i\in S}A_i}{\sum_{i\in S}B_i}<=k$。移项得$\sum_{i\in S}(A_ik-B_i)\geq 0$

可以将所有线段按照左端点排序，然后考虑每个线段对答案的贡献。

若$A_ik-B_i\geq 0$，则选了这个线段肯定能够让答案尽可能大于0，因此把它选上。

如果把所有$A_ik-B_i\geq 0$的线段选上后能够覆盖完整个区间，那么k为可行的代价。

如果不能覆盖完，那么我们要考虑选上一些$A_ik-B_i<0$的线段，使得代价增加最少的情况下，能把区间覆盖完。

用sum表示必选线段A_ik-B_i的和，f_i表示覆盖完i点后，sum要减少的值。

那么对于必选线段i，$f_{r_i}=min\{f_j\mid S_i\leq j\leq t\}$（选了以后并不会影响$sum$）

对于其它线段，$f_{r_i}=min\{f_j\mid S_i\leq j\leq t\}+B_i-A_ik$。由于$f_{r_i}$可能会被计算多次，因此取最小值即可。转移的时候相当于后缀最小值，显然可以使用树状数组优化。最后只需要判断$sum-f_t\geq 0$即可。

```
#include<cstdio>
#include<cctype>
#include<algorithm>
inline int getint() {
    register char ch;
    while(!isdigit(ch=getchar()));
    register int x=ch^'0';
    while(isdigit(ch=getchar())) x=(((x<<2)+x)<<1)+(ch^'0');
    return x;
}
const int N=100001;
const double eps=1e-5,inf=1e10;
struct Node {
    int l,r,w,a;
    bool operator < (const Node &another) const {
        return l<another.l||(l==another.l&&r<another.r);
    }
};
Node node[N];
int n,m;
struct SuffixFenwickTree {
    double val[N];
    int lowbit(const int &x) const {
        return x&-x;
```

```
    }
    void reset() {
        for(register int i=1;i<=m;i++) {
            val[i]=-inf;
        }
    }
    void modify(int p,const double &x) {
        while(p) {
            val[p]=std::max(val[p],x);
            p-=lowbit(p);
        }
    }
    double query(int p) const {
        if(!p) return 0;
        double ret=-inf;
        while(p<=m) {
            ret=std::max(ret,val[p]);
            p+=lowbit(p);
        }
        return ret;
    }
};
SuffixFenwickTree t;
inline bool check(const double &k) {
    t.reset();
    int last=0;
    for(register int i=1;i<=n;i++) {
        if(node[i].a*k-node[i].w<0) continue;
        if(node[i].l-1<=last) last=std::max(last,node[i].r);
    }
    if(last>=m) return true;
    double sum=0;
    t.modify(last,0);
    for(register int i=1;i<=n;i++) {
        if(node[i].a*k-node[i].w<0) {
            t.modify(node[i].r,t.query(node[i].l-1)+node[i].a*k-node[i].w);
        } else {
            sum+=node[i].a*k-node[i].w;
            t.modify(node[i].r,t.query(node[i].l-1));
        }
    }
    return sum+t.query(m)>=0;
}
int main() {
    for(register int T=getint();T;T--) {
        n=getint(),m=getint();
        int sumw=0,suma=0;
        for(register int i=1;i<=n;i++) {
            int l=getint(),r=getint(),w=getint(),a=getint();
            sumw+=w,suma+=a;
            node[i]=(Node){l,r,w,a};
        }
        std::sort(&node[1],&node[n+1]);
        double l=0,r=sumw*1./suma;
        while(r-l>=eps) {
            const double mid=(l+r)/2;
            (check(mid)?r:l)=mid;
        }
```

```
        printf("%.3f\n",r);
    }
    return 0;
}
```