

区间贪心

[POJ 2376: Cleaning Shifts](#) 选择尽量少的区间覆盖一段线段。将所有区间按照左端点升序排序，左端点相同按右端点升序排序。最后从左往右扫描，每次取合法区间中右端点尽量大的。代码如下

```
/*

*/

#define method_1
#ifdef method_1
/*
将所有区间按照左端点升序排序，左端点相同按右端点升序排序。
最后从左往右扫描，每次取合法区间中右端点尽量大的。
*/
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>
#include<set>
#include<map>
#include<queue>
#include<stack>
#include<vector>
#include<cstring>
#include<cstdlib>
#include<iomanip>
#include<ctime>
#include<string>
#define D(x) cout<<#x<<" = "<<x<<"  "
#define E cout<<endl
using namespace std;
typedef long long ll;
typedef pair<int,int>pii;
const int maxn=25000+5;
const int INF=0x3f3f3f3f;
int ans=0,n,t;
struct node{
    int l,r;
    bool operator<(const node &h)const{return (l==h.l)?(r<h.r):l<h.
l;}
}seg[maxn];
int main() {
// ios::sync_with_stdio(false);
```

```

        //freopen("Cleaning Shifts.in","r",stdin);
//  cin>>n>>t; //cin 即使关闭流同步后 依旧是 300ms+
//  for(int i=1;i<=n;i++) cin>>seg[i].l>>seg[i].r;
scanf("%d%d",&n,&t); //用 scanf:60ms
for(int i=1;i<=n;i++) scanf("%d%d",&seg[i].l,&seg[i].r);
sort(seg+1,seg+n+1);
int pos=0,used,rpos=0;
for(int i=1;i<=n){
    if(pos>=t) break;
    if(seg[i].l>pos+1){
        cout<<-1;return 0;
    } //这里注意需要判断是否能接上
    while(seg[i].l<=pos+1&& i<=n){ //注意这里是 pos+1
        if(seg[i].r>rpos){
            rpos=seg[i].r,used=i;
        }
        i++;
    }
    pos=seg[used].r,ans++;
}
pos<t?cout<<-1:cout<<ans;
return 0;
}
#endif
#ifdef method_2
/*

*/

#endif
#ifdef method_3
/*

*/

#endif

```

POJ 1328: Radar Installation

将每个输入的 x,y 坐标作为圆心，半径为 d 作圆。在 x 轴上截取出一段线段(如果圆和 x 轴无交点则无解) 于是问题转化为用尽量少的点覆盖所有线段。将线段按照右端点升序排序，右端点相同按左端点升序排序(事实上降序排序也可以，这样不影响结果)。然后每次选择能够覆盖到的、且右端点最大的线段的右端点放置雷达站，并更新答案即可。代码如下

```

/*
Radar Installation 这个名字会被管家列入黑名单 所以只能取这个名字
*/
#define method_1
#ifdef method_1
/*
将每个输入的 x,y 坐标作为圆心，半径为 d 作圆。在 x 轴上截取出一段线段(如果圆
和 x 轴无交点则无解)
于是问题转化为用尽量少的点覆盖所有线段。
将线段按照右端点升序排序，右端点相同按左端点升序排序(事实上降序排序也可以，
这样不影响结果)。
然后每次选择能够覆盖到的、且右端点最大的线段的右端点放置雷达站，并更新答案
即可。
*/
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>
#include<set>
#include<map>
#include<queue>
#include<stack>
#include<vector>
#include<cstring>
#include<cstdlib>
#include<iomanip>
#include<ctime>
#include<string>
#define D(x) cout<<#x<<" = "<<x<<" "
#define E cout<<endl
using namespace std;
typedef long long ll;
typedef pair<int,int>pii;
const int maxn=1000+5;
const int INF=0x3f3f3f3f;
double x,y,d;
struct node {
    double l,r;
    bool operator<(const node &h)const {
        return (r==h.r)?l<h.l:r<h.r; //这里用 eps=1e-6 判断相等竟然会错
    }
} seg[maxn];
int n,ans,kase=0;

```

```

void cal(){
    double rpos=seg[1].r;
    for(int i=2;i<=n;i++){
        if(seg[i].l>rpos){
            rpos=seg[i].r;
            ans++;
        }
    }
}

int main() {
    ios::sync_with_stdio(false);
    //freopen("Radar Installation.in","r",stdin);
    while(cin>>n>>d) {
        if(n==0) break;
        cout<<"Case "<<++kase<<": ";
        bool flag=true;
        ans=1;
        for(int i=1; i<=n; i++) {
            cin>>x>>y;
            seg[i].l=x-sqrt(d*d-y*y);
            seg[i].r=x+sqrt(d*d-y*y);
            if(abs(y)>d) {
                flag=false;
            }
        }
        if(flag==false) {
            cout<<-1<<endl;
            continue;
        }
        sort(seg+1,seg+n+1);
        /*
        for(int i=1;i<=n;i++){
            D(seg[i].l);D(seg[i].r);E;
        }
        */
        cal();
        cout<<ans<<endl;
    }
    return 0;
}
#endif
#ifdef method_2
/*

```

```

*/

#endif
#ifdef method_3
/*

*/

#endif

```

POJ 3190: Stall Reservations 明显要先处理产奶时间靠前并且结束时间靠前的奶牛，这样才能使后面的奶牛接着使用同一畜栏。首先对结构体按照开始时间升序排列。这样从 1~N 找时只要判断该奶牛的起始时间是否比前面每个畜栏的最小的终止时间大，如果大的话，就可以安排在其后面产奶，并且更新该畜栏的终止时间。如果小的话，则必须新开一间畜栏。因为一直要找最小值，考虑使用优先队列（**priority_queue**），结束时间小的优先级高。这样比较方便找最小值。代码如下

```

/*

*/
#define method_1
#ifdef method_1
/*
明显要先处理产奶时间靠前并且结束时间靠前的奶牛，这样才能使后面的奶牛接着使用同一畜栏。
首先对结构体按照开始时间升序排列。
这样从 1~N 找时只要判断该奶牛的起始时间是否比前面每个畜栏的最小的终止时间大，
如果大的话，就可以安排在其后面产奶，并且更新该畜栏的终止时间。
如果小的话，则必须新开一间畜栏。
因为一直要找最小值，考虑使用优先队列（priority_queue），结束时间小的优先级高。这样比较方便找最小值。
*/
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>
#include<set>
#include<map>
#include<queue>
#include<stack>
#include<vector>
#include<cstring>
#include<cstdlib>
#include<iomanip>

```

```

#include<ctime>
#include<string>
#define D(x) cout<<#x<<" = "<<x<<" "
#define E cout<<endl
using namespace std;
typedef long long ll;
typedef pair<int,int>pii;
const int maxn=50000+5;
const int INF=0x3f3f3f3f;
int n,ans;
struct node {
    int l,r,stall,id;
    bool operator<(const node &h)const {
        return r>h.r; //小根堆 用大于号
    }
} seg[maxn];
priority_queue<node>q;
bool cmp1(node a,node b) {
    return a.l<b.l;
}
bool cmp2(node a,node b) {
    return a.id<b.id;
}
void init(){
    while(q.size()) q.pop();
    ans=1; //第一头牛在第一个 stall 里头
}
int main() {
    // ios::sync_with_stdio(false);
    //freopen("Stall Reservations.in","r",stdin);
    while(scanf("%d",&n)!=EOF){//有多组数据
        init();
        for(int i=1; i<=n; i++) {
            scanf("%d%d",&seg[i].l,&seg[i].r);seg[i].id=i;
        }
        sort(seg+1,seg+n+1,cmp1);
        seg[1].stall=1;q.push(seg[1]);
        for(int i=2;i<=n;i++){
            node now=q.top();
            if(now.r>=seg[i].l){ //需要新建 stall
                ans++;seg[i].stall=ans;q.push(seg[i]);
            }
            else{
                seg[i].stall=now.stall;q.pop();q.push(seg[i]);
            }
        }
    }
}

```

```

        }
    }
    sort(seg+1,seg+n+1,cmp2);
    printf("%d\n",ans);
    for(int i=1;i<=n;i++) printf("%d\n",seg[i].stall);
}
return 0;
}
#endif
#ifdef method_2
/*

*/

#endif
#ifdef method_3
/*

*/

#endif

```

其他贪心

POJ 2393: Yogurt factory $ans = \sum (c_j + s * (i - j)) * y_i = \sum (c_j - s * j) * y_i + s * y_i * i$ 其中 $j \leq i$

用一个小根堆维护 $c_j - s * j$ 即可 代码如下

```

/*

*/
#define method_1
#ifdef method_1
/*
ans=Σ(c_j+s*(i-j))*y_i= Σ(c_j-s*j)*y_i+s*y_i*i
j<=i
用一个小根堆维护 c_j-s*j 即可
*/
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>
#include<set>

```

```

#include<map>
#include<queue>
#include<stack>
#include<vector>
#include<cstring>
#include<cstdlib>
#include<iomanip>
#include<ctime>
#include<string>
#define D(x) cout<<#x<<" = "<<x<<"  "
#define E cout<<endl
using namespace std;
typedef long long ll;
typedef pair<int,int>pii;
const int maxn=10000+5;
const int INF=0x3f3f3f3f;
priority_queue<ll,vector<ll>,greater<ll> >q;
ll n,s,c,y,ans=0;
int main() {
    ios::sync_with_stdio(false);
    //freopen("Yogurt factory.in","r",stdin);
    scanf("%lld%lld",&n,&s);
    for(int i=1;i<=n;i++){
        scanf("%lld%lld",&c,&y);
        q.push(c-s*i);
        ll temp=q.top();
        ans+=temp*y+s*i*y;
    }
    printf("%lld",ans);
    return 0;
}
#endif
#ifdef method_2
/*

*/

#endif
#ifdef method_3
/*

*/

#endif

```

[POJ 1017: Packets](#) 大力分类讨论即可。贪心原则：优先放体积大的，空隙中插入体积小的。代码如下

```
/*

*/
#define method_1
#ifdef method_1
/*
大力分类讨论即可。
贪心原则：优先放体积大的，空隙中插入体积小的。具体详见代码注释。
*/
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>
#include<set>
#include<map>
#include<queue>
#include<stack>
#include<vector>
#include<cstring>
#include<cstdlib>
#include<iomanip>
#include<ctime>
#include<string>
#define D(x) cout<<#x<<" = "<<x<<" "
#define E cout<<endl
using namespace std;
typedef long long ll;
typedef pair<int,int>pii;
const int maxn=+5;
const int INF=0x3f3f3f3f;
int a,b,c,d,e,f,ans;
int main() {
    ios::sync_with_stdio(false);
    //freopen("Packets.in","r",stdin);
    while(cin>>a>>b>>c>>d>>e>>f){
        if(!a&&!b&&!c&&!d&&!e&&!f) break;
        ans=f; //6*6
        ans+=e; //5*5 剩余空间只能填 1*1
        if(a>=11*e) a-=11*e;
        else a=0;
        ans+=d; //4*4 剩余空间优先放 2*2 还有剩余就放 1*1
        if(b>=5*d) b-=5*d;
```

```

else{
    if(a>=36*d-16*d-4*b) a-=36*d-16*d-4*b;
    else a=0;
    b=0;
}
if(c%4==0) ans+=c/4; //3*3 如果刚好有 4 个的倍数 就全部填满
if(c%4==1){ //3*3 多一个出来 能放 5 个 2*2 的 剩下的放 1*1
    ans+=c/4+1;
    if(b>=5){
        b-=5;
        if(a>=7) a-=7;
        else a=0;
    }
    else{ //2*2 的不足 5 个
        if(a>=27-4*b) a-=27-4*b;
        else a=0;
        b=0;
    }
}
if(c%4==2){ //3*3 多两个出来 能放 3 个 2*2 的 剩下的放 1*1
    ans+=c/4+1;
    if(b>=3){
        b-=3;
        if(a>=6) a-=6;
        else a=0;
    }
    else{
        if(a>=18-4*b) a-=18-4*b;
        else a=0;
        b=0;
    }
}
if(c%4==3){ //3*3 多两个出来 能放 1 个 2*2 的 剩下的放 1*1
    ans+=c/4+1;
    if(b>=1){
        b-=1;
        if(a>=5) a-=5;
        else a=0;
    }
    else{ //b=0
        if(a>=9) a-=9;
        else a=0;
    }
}
}

```

```

        if(b%9==0) ans+=b/9;
        else{
            ans+=b/9+1;
            b%=9;
            if(a>=36-4*b) a-=36-4*b;
            else a=0;
        }
        if(a%36==0) ans+=a/36;
        else ans+=a/36+1;
        cout<<ans<<endl;
    }
    return 0;
}
#endif
#ifdef method_2
/*

*/

#endif
#ifdef method_3
/*

*/

#endif

```

POJ 3040; Allowance 英文题虽然可以看样例猜题意，但是有些东西不会注意到。比如说这里的 **evenly divides** 即整除。数据有了这个性质，就可以贪心了。将硬币从小到大排序，大于等于 c 元的硬币直接支付。对于剩下的硬币，挑选的原则是先找大面额的，尽量使得面额总值接近 C 。接下来用小面额（尽量小，也就是从小到大刷选）的去补充。贪心证明：大面额是小面额的倍数(**evenly divides**)，存在可以用足够量的小面额来代替大面额的情况。那么这样一来就浪费了小面额，所以一开始尽量使用大面额，之后再用小面额补充。代码如下

```

/*

*/
#define method_1
#ifdef method_1
/*

```

英文题虽然可以看样例猜题意，但是有些东西不会注意到。
 比如说这里的 **evenly divides** 即整除。
 数据有了这个性质，就可以贪心了。
 将硬币从小到大排序，大于等于 c 元的硬币直接支付。

对于剩下的硬币，挑选的原则是先找大面额的，尽量使得面额总值接近 c 。

接下来用小面额（尽量小，也就是从小到大刷选）的去补充。

贪心证明：大面额是小面额的倍数(**evenly divides**)，存在可以用足够量的小面额来代替大面额的情况。

那么这样一来就浪费了小面额，所以一开始尽量使用大面额，之后再用小面额补充。

```
*/
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>
#include<set>
#include<map>
#include<queue>
#include<stack>
#include<vector>
#include<cstring>
#include<cstdlib>
#include<iomanip>
#include<ctime>
#include<string>
#define D(x) cout<<#x<<" = "<<x<<" "
#define E cout<<endl
using namespace std;
typedef long long ll;
typedef pair<int,int>pii;
const int maxn=20+5;
const int INF=0x3f3f3f3f;
int n,c,used[maxn],ans=0; //used[i]表示凑一周（凑到c）的情况下，第i大面值的硬币的使用数
struct node {
    int num,v;
    bool operator<(const node &h)const {
        return v<h.v;
    }
} money[maxn];
int main() {
    ios::sync_with_stdio(false);
    //freopen("Allowance.in","r",stdin);
    cin>>n>>c;
    for(int i=1; i<=n; i++) cin>>money[i].v>>money[i].num;
    sort(money+1,money+n+1);
    for(int i=n; i>=1; i--) { //将硬币从小到大排序，大于等于 c 元的硬币
        直接支付。
        if(money[i].v>=c) {
```

```

        ans+=money[i].num;
        money[i].num=0;
    }
    /*
    //注意不能这样写 因为如果所有硬币面值都大于 c 那么 n 就不会被更新
    if(money[i].v<c){
        n=i;break;
    }
    ans+=money[i].num;
    */
}
// D(ans);D(n);
while(true) {
    bool flag=false;
    int sum=c;
    memset(used,0,sizeof(used));
    //对于剩下的硬币，挑选的原则是先找大面额的，尽量使得面额总值接近 c。
    for(int i=n; i>=1; i--) {
        if(money[i].num>0) {
            used[i]=min(sum/money[i].v,money[i].num);
            sum-=used[i]*money[i].v;
            if(sum==0) {
                flag=true;
                break;
            }
        }
    }
}
//接下来用小面额（尽量小，也就是从小到大刷选）的去补充。
if(sum>0) for(int i=1; i<=n; i++) {
    if(money[i].num-used[i]>0) { //未用光
        while(used[i]<money[i].num) {
            sum-=money[i].v;
            used[i]++;
            if(sum<=0) {
                flag=true;
                break;
            }
        }
    }
    if(flag) break;
}
if(!flag) break; //凑不出来了
int minn=INF;
for(int i=1; i<=n; i++) {

```

```
        if(used[i]) minn=min(money[i].num/used[i],minn); //minn
表示这种硬币组合用了几周
```

```
    }
    for(int i=1; i<=n; i++) {
        if(used[i]) money[i].num-=minn*used[i];
    }
    ans+=minn;
}
cout<<ans;
return 0;
}
#endif
#ifdef method_2
/*

*/

#endif
#ifdef method_3
/*

*/

#endif
```

[POJ 1862: Stripies](#)

```
/*
```

```
*/
```

```
#define method_1
```

```
#ifdef method_1
```

```
/*
```

贪心原则：每次选择最重的两个虫子合并，合并 $n-1$ 次就是答案。维护最重的虫子，可以用一个大根堆。

假设三个变形虫的重量分别为 a, b, c

若 $a < b < c$ ，则按照贪心原则，最终得到的重量是 $2 * \sqrt{a * 2 * \sqrt{b * c}}$

采用邻项交换的方法，交换 a 和 b 的位置，则最终得到的重量是 $2 * \sqrt{b * 2 * \sqrt{a * c}}$

这样显然 $2\sqrt{a \cdot 2\sqrt{b \cdot c}} < 2\sqrt{b \cdot 2\sqrt{a \cdot c}}$ 。

因此得证。

```
*/  
  
#include<iostream>  
#include<cstdio>  
#include<algorithm>  
#include<cmath>  
#include<set>  
#include<map>  
#include<queue>  
#include<stack>  
#include<vector>  
#include<cstring>  
#include<cstdlib>  
#include<iomanip>  
#include<ctime>  
#include<string>  
  
#define D(x) cout<<#x<<" = "<<x<<" "  
#define E cout<<endl  
  
using namespace std;  
typedef long long ll;  
typedef pair<int,int>pii;  
const int maxn=100+5;  
const int INF=0x3f3f3f3f;  
  
int n;  
  
double temp1,temp2;  
priority_queue<double>q;  
  
int main() {
```

```

// ios::sync_with_stdio(false);
//freopen("Stripies.in","r",stdin);
cin>>n;
for(int i=1;i<=n;i++) cin>>temp1,q.push(temp1);
for(int i=1;i<=n-1;i++){
    temp1=q.top();q.pop();
    temp2=q.top();q.pop();
    q.push(2*sqrt(temp1*temp2));
}
printf("%.3lf",q.top());
return 0;
}
#endif

#ifdef method_2
/*

*/

#endif

#ifdef method_3
/*

*/

#endif

POJ 3262: Protecting the Flowers

/*

```

```

*/

#define method_1

#ifdef method_1

/*
对于任意两个在运送次序中相邻的奶牛  $i$  和  $j$ 。
显然在运送他们之前和之后的奶牛的代价与  $i$  和  $j$  的相对顺序无关。
因此只考虑  $i$  比  $j$  先运送 和  $i$  比  $j$  后运送两种情况。
 $i$  比  $j$  先运送 产生代价为  $2*t_i*d_j$ 
 $i$  比  $j$  后运送 产生代价为  $2*t_j*d_i$ 
于是只要按照  $t_i*d_j$  升序排序即可。
最后为了方便计算，可以对所有  $d_i$  求和记为  $sum$ ，每次计算时令  $sum-=d_i$ ；
再令  $ans+=2*t_i*sum$ ；即可。
PS：注意乘法会爆  $int$ ，所以要用  $long\ long$ 
*/

#include<iostream>

#include<cstdio>

#include<algorithm>

#include<cmath>

#include<set>

#include<map>

#include<queue>

#include<stack>

#include<vector>

#include<cstring>

#include<cstdlib>

#include<iomanip>

#include<ctime>

#include<string>

```

```

#define D(x) cout<<#x<<" = "<<x<<"  "
#define E cout<<endl
using namespace std;
typedef long long ll;
typedef pair<int,int>pii;
const int maxn=100000+5;
const int INF=0x3f3f3f3f;
int n;
ll sum=0,ans=0;
struct node{
    ll t,d;
    bool operator<(const node &h)const{return t*h.d<d*h.t;}
}cow[maxn];
int main() {
    // ios::sync_with_stdio(false);
    //freopen("Protecting the Flowers.in","r",stdin);
    scanf("%d",&n);
    for(int i=1;i<=n;i++) scanf("%lld%lld",&cow[i].t,&cow[i].d),sum+=cow[i].d;
    sort(cow+1,cow+n+1);
    for(int i=1;i<=n;i++){
        sum-=cow[i].d;
        ans+=2*cow[i].t*sum;
    }
    printf("%lld",ans);
    return 0;
}
#endif
#ifdef method_2

```

```
/*
```

```
*/
```

```
#endif
```

```
#ifdef method_3
```

```
/*
```

```
*/
```

```
#endif
```