贪心算法

信智学院　陈琰宏

教学内容

01 选择不相交区间问题

02 区间选点问题

03 区间覆盖问题

04 流水作业调度问题

第1章 贪心算法

一、贪心算法

贪心算法是从问题的初始状态出发，通过若干次的贪心选择而得出最优值（或较优值）的一种方法。贪心策略并不是从整体上考虑问题，它所做出的选择只是在某种意义上的局部最优解，而许多问题自身的特性决定了该题运用贪心策略可以得到最优解或较优解。

二、贪心算法的特点

1.贪心选择性质：所谓贪心选择性质是指应用同一规划，将原问题变为一个相似的但规模更小的子问题，而后的每一步都是当前看似最佳的选择，这种选择依赖于已做出的选择，但不依赖于未做的选择。

2.最优子结构性质：算法中每一次都取得了最优解(即局部最优解)，要保证最后的结果最优，则必须满足全局最优解包含局部最优解。

三、几个简单的贪心例子

1.最优装载问题:

给n个物体，第i个物体重量为wi，选择尽量多的物体，使得总重量不超过C。

【问题分析】

贪心策略：先装最轻的。

2.部分背包问题：

有n个物体，第i个物体的重量为wi，价值为vi，在总重量不超过C的情况下让总价值尽量高。每一个物体可以只取走一部分，价值和重量按比例计算。

【问题分析】

贪心策略：先拿性价比高的。

3.乘船问题：

有n个人，第i个人重量为wi。每艘船的载重量均为C，最多乘两个人。用最少的船装载所有人。

【问题分析】

贪心策略：最轻的人和最重的人配对。

四、贪心的经典应用

1、选择不相交区间问题：

给定n个开区间(ai,bi)，选择尽量多个区间，使得这些区间两两没有公共点。

【算法】

首先按照结束时间b1<=b2<=…<=bn的顺序排序，依次考虑各个活动，如果没有和已经选择的活动冲突，就选；否则就不选。

【正确性】

假设bj<bi且(aj,bj)(ai,bi)分别与之前的活动不冲突，如果当前选(aj,bj)，若(aj,bj)(ai,bi)不冲突则还可以选择(ai,bi)，答案加一；若(aj,bj)(ai,bi)冲突，因为bj<bi，所以(aj,bj)对以后的影响更小。

[3561] 活动安排

设有n个活动的集合E={1,2……n},其中每个活动都要求使用同一资源。而在同一时间内只有一个活动能使用这一资源。每个活动i都有一个要求使用该资源的起始时间si和一个结束时间fi。且si<fi。如果选择了活动i，则它在时间区间[si,fi)内占用资源。若区间[si,fi)与区间[sj,fj)不相交，则两活动是相容的。选择出相互兼容的活动组成的最大集合。n<=100000.

[3561] 活动安排

输入格式 :第一行一个整数n ； 接下来的 n行，每行两个整数 si 和fi 。
输出格式: 输出互相兼容的最大活动个数。
输入样例
4
1 3
4 6
2 5
1 7
输出样例
2

```cpp
#include <bits/stdc++.h>
using namespace std;
struct work{
    int s;//开始时间
    int f;//结束时间
} a[1005];

bool cmp(work a, work b) {
    return a.f < b.f;//按照结束时间排序
}

int n, ans, now;
```

```
14  int main() {
15      scanf("%d", &n);
16      for (int i=1; i<=n; i++)
17          scanf("%d%d", &a[i].s, &a[i].f);
18      sort(a+1, a+1+n, cmp);
19      for (int i=1; i<=n; i++) {
20          if (a[i].s>=a[now].f) {
21              ans++;//当前活动的开始时间>=上一个活动的结束时间
22              now=i;
23          }
24      }
25      printf("%d", ans);
26      return 0;
27  }
```

2、区间选点问题

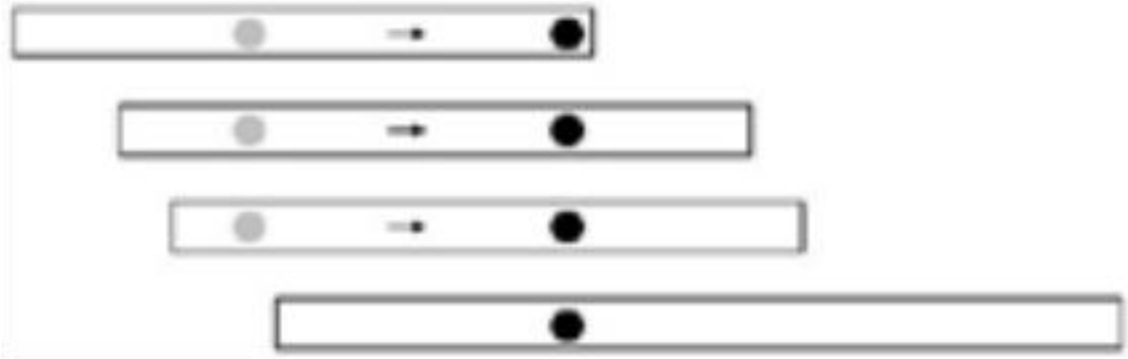
给定n个闭区间[ai, bi]，在数轴上选尽量少的点，使得每个区间内都至少有一个点（不同区间内含的点可以是同一个）。

【算法】

首先按照结束位置从小到大排序。然后从区间1到区间n进行选择：对于当前区间，若集合中的数不能覆盖它，则将区间末尾的数加入集合。

贪心策略是：取最后一个。

如下图，如果选灰色点，则移动到黑色点会更优。



[3562] 种树

一条街道的一边有几座房子。因为环保原因居民想要在路边种些树，路边的地区被分割成n块，并被编号为1…n，每块大小为一个单位尺寸并最多可种一棵树。每个居民想在门前种些树并指定了三个数b, e, t这三个数分别表示该居民想在b和e之间种最少t棵树，当然，b<=e, t<=e-b+1, 允许居民想种树的子区域可以交叉。出于资金紧缺的原因，环保部门请你能够满足所有居民种树要求时所需要种的树的最少数量。

输入第一行为n ，表示路段数。第二行为h ，表示建议数。下面 h 行描述一条建议：b，e, t用一个空格分隔。n<=30000, h<=5000

输出只有一个数，为满足所有居民的建议，所需要种树的最少数量。

[3562] 种树

输入样例

```
9
4
1 4 2
4 6 2
8 9 2
3 5 2
```

输出样例

```
5
```

分析

典型的区间选点问题。每一个房子代表一个要求，这个要求对应着一个左端点和右端点，以及一个要求的树数。它就相当于是给定一个区间，在这个区间内插入要求的点数。求的种数的最小数量即为总点数。

将这个问题转化为一个区间选点问题并得出基本思路。

基本思路：种树要种得少，就要尽量让一棵树处在多个区间内。这样，我们只需要尽量在重叠区间种树即可。而重叠部分一定位于区间尾部。因此，我们可以先按照所有区间的结束位置进行排序，之后依次处理每一个区间，先在一个区间的尾部种满足够的树，对下一个区间，只需要判断还需要多少棵树就在区间的尾部中多少就可以。

想要种树种得少，就要一棵树在多个区间同时出现。
所以，在重叠部分种尽可能多的树即可。
然而重叠部分一定在区间的尾部。
所以先对结束位置进行排序，然后依次在区间的尾部从前往后种树直到满足要求，对于下一个区间，看看差多少树，就在结尾补多少。

贪心：

- 按结束位置排序
- 对每个区间一次处理
- 从前往后扫描区间，统计已有的树的个数
- 若已选点超过要求个数，则continue
- 否则从后往前，添加缺少的覆盖点
- 输出ans

```cpp
#include<bits/stdc++.h>
using namespace std;
struct node{
    int s;//起始位置
    int e;
    int num;//[s,e]间要种树的数量
}a[5005];

int used[30005];// 记录每个位置的状态，0表示没有种树

bool cmp(node b,node c){
    return b.e<c.e;
}

int main(){
    int n,m,ans=0;
    cin>>n>>m;
    for(int i=1;i<=m;i++)
        cin>>a[i].s>>a[i].e>>a[i].num;
    sort(a+1,a+1+m,cmp);//完成初始化
```

```
22      for(int i=1;i<=m;i++){//对每个区间按顺序进行处理
23          //1先统计该区间的已有的树的数量
24          int k=0;
25          for(int j=a[i].s;j<=a[i].e;j++)
26              if(used[j])k++;
27          if(k>=a[i].num)continue;//该区间数量已够，不用再种
28          //如果不够开始从后往前种树
29          for(int j=a[i].e;j>=a[i].s;j--){
30              if(!used[j]){
31                  used[j]=1;//种树
32                  k++;//统计该区间的树的数量
33                  ans++;//统计总的种树数量
34                  if(k==a[i].num)
35                      break;//树已够，退出
36              }
37          }
38      }
39      cout<<ans;
40      return 0;
41  }
```

3、区间覆盖问题

给n个闭区间[ai,bi]，选择尽量少的区间覆盖一条指定线段[s,t]。

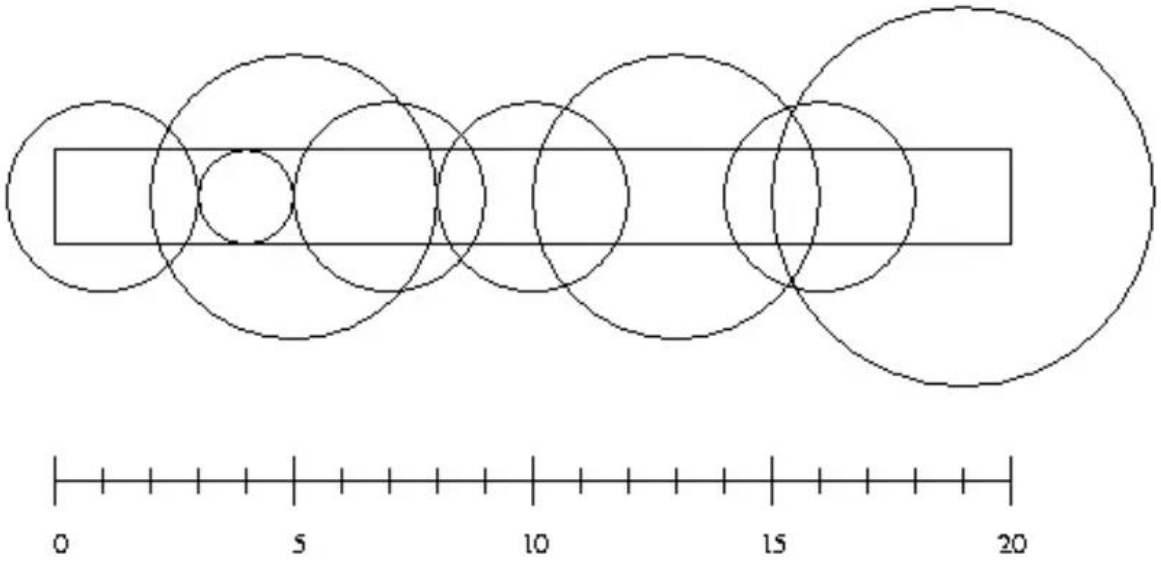
【算法】

将所有的区间按左端点从小到大排序，依次处理每个区间。每次选择覆盖点s的区间中右端点坐标最大的一个，并将s更新为该区间的右端点坐标，直到选择的区间已包含了t为止。

贪心思想：在某个时刻s，找一个满足a[i]<=s的b[i]最大值即可。

[3599] 喷水装置

长L米，宽W米的草坪里装有n个浇灌喷头，每个喷头都装在草坪的中心线上(离两边各W/2米)。我们知道每个喷头的位置(离草坪中心线左端的距离)，以及它能覆盖到的半径。问如果要同时浇灌整块草坪，最少需要打开多少个喷头。若无法覆盖整个草坪则输出-1.



[3599] 喷水装置

输入包含若干组测试数据。第一行一个整数 T表示数据组数；每组数据的第一行是整数 n、L 和 W；n<=10000 接下来的 n 行，每行包含两个整数，给出一个喷头的位置和浇灌半径（上面的示意图是样例输入第一组数据所描述的情况）。

对每组测试数据输出一个数字，表示要浇灌整块草坪所需喷头数目的最小值。如果所有喷头都打开也不能浇灌整块草坪，则输出 -1 。

输入样例

```
3
8 20 2
5 3
4 1
1 2
7 2
10 2
13 3
16 2
19 4
3 10 1
3 5
9 3
6 1
3 10 1
5 3
1 1
9 1
```

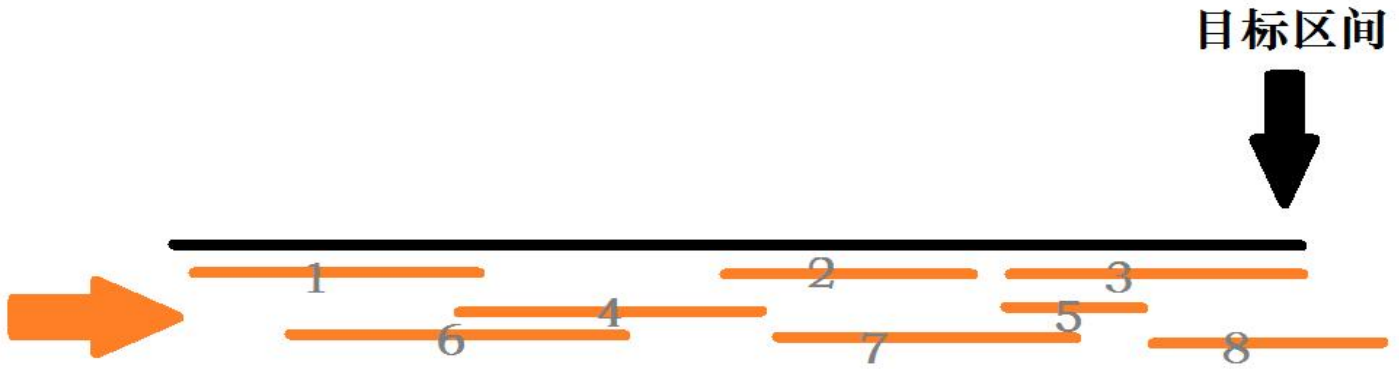
输出样例

```
6
2
-1
```

分析

题目简化：给定一条线段（区间）l（其端点为A, ）和k个区间(p,q），请选用最少的区间覆盖这一条线段（区间）
贪心策略

将所有的区间按照左端点从小到大排序，选择覆盖点A的区间中右端点最大的一个，并将A更新为这个区间右端点的坐标，再次选择覆盖点A的区间中右端点最大的一个，将A更新为这个区间右端点的坐标，如此往复，直到覆盖了点B
一些小细节。



分析

对于本题，还有一些小细节需要注意，如下：

读入的是喷水装置（即一个圆）的半径r， 而不是覆盖的范围，覆盖范围应该是：

a[k].x=o-sqrt(r*r-(w/2)*(w/2))

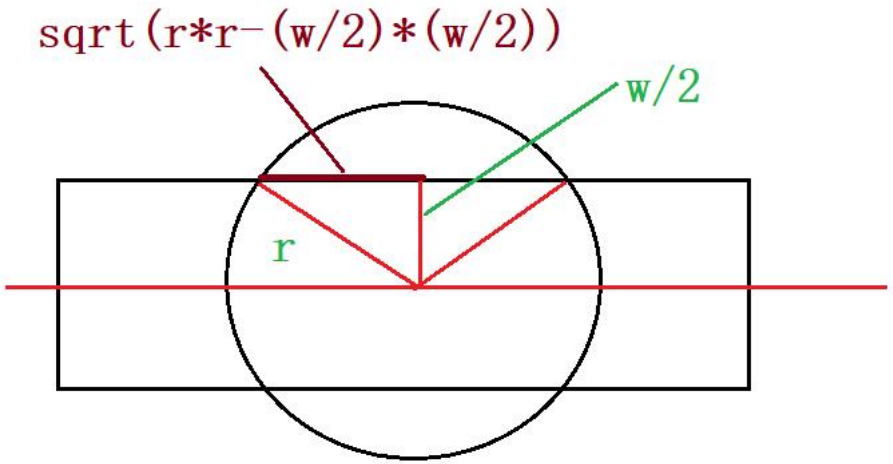
a[k].y=o+sqrt(r*r+(w/2)*(w/2))

这里x和y是区间的左右端点，o为喷水装置的坐标，

即喷水装置喷水范围的圆心，r为其半径，w为草坪的宽度

如果喷水装置的半径小于草坪的宽度，那么这个喷水装置肯定不能用，

因为它一块区间都覆盖不了



```
3   int T;
4   struct node{
5       double x,y;
6   }a[20005];//注意数组大小
7   int main()
8   {
9       scanf("%d",&T);
10      while(T--)
11      {
12          int n,l,w;
13          scanf("%d%d%d",&n,&l,&w);
14          int cnt = 0;
15          for(int i = 1;i <= n;i++)
16          {
17              int p,q;
18              scanf("%d%d",&p,&q);
19              if(2 * q <= w)continue;
20              cnt++;
21              a[cnt].x = p - sqrt(q * q - w * w/4.0);
22              a[cnt].y = p + sqrt(q * q - w * w/4.0);
23          }
```

```
24          double t = 0;
25          int ans = 0,flag = 0;
26          while(t < l)
27          {//在所有左端点小于上一个区间右端点的线段中选择右端点最大的一个
28              ans++;
29              double s = t;
30              for(int i = 1;i <= cnt;i++)
31              {
32                  if(a[i].x <= s&&t < a[i].y)t = a[i].y;
33              }
34              if(t == s && s < l){
35              //如果找不到两个区域之间断开了或者所有区域长度之和小于规定长度就输出-1
36                  printf("-1\n");
37                  flag = 1;
38                  break;
39              }
40          }
41          if(flag == 0)printf("%d\n",ans);
42      }
43      return 0;
```

```
4  #include<algorithm>
5  using namespace std;
6
7  struct node
8  {
9      double x,y;//x-y这一区间的长度
10 }a[20010];
11
12 int n,last,ans;
13 double st,ed,l,w;
14 int cmp(node n1,node n2)
15 {
16     return n1.x<n2.x;//头越小，整个区间就越长
17 }
18 double gougu(double r)//勾股定理
19 {
20     return sqrt(r*r-(w/2)*(w/2));
21     //利用三角形的勾股定理来求出覆盖的长度
22 }
```

```
23  int main()
24  {
25      int t;
26      scanf("%d",&t);
27      while(t--)
28      {
29          scanf("%d%lf%lf",&n,&l,&w);
30          double o,r;
31          for(int i=1;i<=n;i++)
32          {
33              scanf("%lf%lf",&o,&r);
34              if(r<(w/2)) a[i].x=a[i].y=0;
35              /*
36              如果浇灌的半径大于宽的一半，其实就是长度大于宽的话，
37              那么这个区间就不需要记录，因为这个可以直接实现
38              */
39              else a[i].x=o-gougu(r),a[i].y=o+gougu(r);
40              //否则利用勾股求出最小的头和最大的尾
41          }
42
```

```
43          sort(a+1,a+n+1,cmp);//这是贪心的一步
44          st=0,ed=0,last=0,ans=0;
45          //st是用来保存当前所到的区间的头 //ed是用来保存当前所到区间的尾
46          //last是用来表示当前到的喷头数    //ans是用来保存已经使用的喷头
47          while(ed<l && last<=n)//当还有喷头并且没有到尽头的时候进入
48          {
49              for(int i=last+1;i<=n;i++)
50              {
51                  if(a[i].x<=st)//如果当前的区间的头小于之前的，就记录
52                  {
53                      if(a[i].y>ed)
54                      {
55                          ed=a[i].y;
56                          last=i;
57                      }//说明找到了最新数据的区间也就是最长的（贪心）
58                  }
59                  else break;
60              }
61              if(ed==st)
62              //如果这个尾等于头，说明就算打开全部喷头也做不到，就输出-1
```

```
61              if(ed==st)
62              //如果这个尾等于头，说明就算打开全部喷头也做不到，就输出-1
63              {
64                  ans=-1;
65                  break;//不能return 0是因为有多组数据
66              }
67              st=ed; ans++;
68              /*否则就可以把当前的头记录成尾巴，使用一个喷头，
69              说明这一个区间是成立中的最大的 */
70          }
71
72          if(last>n) printf("-1\n");//如果最后用到的喷头数大于拥有的，输出-1
73          else printf("%d\n",ans);//否则输出用到的最少的喷头数
74      }
75      return 0;
76
77  }
```

今天的课程结束啦……

下课了…
同学们再见！