贪心法

1. 删数问题　　//洛谷 P1106

【问题描述】

键盘输入一个高精度的正整数 n（<=240 位），去掉其中任意 s 个数字后剩下的数字按原左右次序，将组成一个新的正整数。编程对给定的 n 和 s，寻找一种方案，使得剩下的数字组成的新数最小。

输入：

n

s

输出：

最后剩下的最小数。

【样例输入】

178543

4

【样例输出】

13

【参考程序】

```
#include <cstdio>
#include <iostream>
#include <string>
using namespace std;

int main()
{
   string s;
   int n, i;

   cin >> s;    //"178543", "123456"
   cin >> n;    //4

   while (n>0)
   {
      i = 0;   //从串首开始找
      while (i<=s.size()-2 && s[i]<=s[i+1]) i++;
      s = s.substr(0,i) + s.substr(i+1);   //删除字符串 n 的第 i 个字符
      n--;
   }

   while (s.length()>1 && s[0]=='0')    //删去串首可能产生的无用零
      s = s.substr(1);   //"000"
```

```
    cout << s;
    return 0;
}
```

2. 纪念品分组(group.cpp) 1094 //www.luogu.org luogu.lohu.info

【问题描述】

元旦快到了，校学生会让乐乐负责新年晚会的纪念品发放工作。为使得参加晚会的同学所获得的纪念品价值相对均衡，他要把购来的纪念品根据价格进行分组，但每组最多只能包括两件纪念品，并且每组纪念品的价格之和不能超过一个给定的整数。为了保证在尽量短的时间内发完所有纪念品，乐乐希望分组的数目最少。

你的任务是写一个程序，找出所有分组方案中分组数最少的一种，输出最少的分组数目。

【输入】

输入文件 group.in 包含 n+2 行：

第 1 行包括一个整数 w，为每组纪念品价格之和的上限。

第 2 行为一个整数 n，表示购来的纪念品的总件数。

第 3～n+2 行每行包含一个正整数 p_i(5<=p_i<=w)，表示所对应纪念品的价格。

【输出】

输出文件 group.out 仅一行，包含一个整数，即最少的分组数目。

【输入输出样例】

group.in	group.out
100 9 90 20 20 30 50 60 70 80 90	6

【限制】

50％的数据满足：1<=n<=15

100％的数据满足：1<=n<=30000，80<=w<=200

【参考程序】

```
#include <cstdio>
#include <iostream>
#include <algorithm>
using namespace std;
```

```
int main()
{
   int w, n;
   cin >> w;
   cin >> n;
   int a[n];

   for (int i=0; i<n; i++)  cin >> a[i];

   sort(a, a+n);

   int L=0, r = n-1, s=0;

   while (L<=r)
   {
      if (a[L]+a[r]<=w){
         L++; r--;
      }
      else
         r--;
      s++;
   }

   cout << s;
   return 0;
}
```

3. 独木舟（kaj.cpp）

【问题描述】

我们计划组织一个独木舟旅行。租用的独木舟都是一样的，最多乘两人，而且载重有一个限度。现在要节约费用，所以要尽可能地租用最少的舟。你的任务是读入独木舟的载重量，参加旅行的人数以及每个人的体重，计算出所需要的租船数目。

【输入格式】

输入文件名为 kaj . in，其第一行是 w（80≤w≤200），表示每条独木舟最大的载重量。第二行是整数 n（1≤n≤30000，参加旅行的人数。接下来的 n 行，每行是一个整数 Ti（5≤Ti≤w），表示每个人的重量。

【输出格式】

输出文件仅一行，表示最少的租船数目。

【样例输入】

```
100
  9
```

90
20
20
30
50
60
70
80
90

【 样例输出】

6

4. 排队接水(water.cpp) 1223

【问题描述】

有 n 个人在一个水龙头前排队接水。假如每个人接水的时间为 Ti，请编程找出这 n 个人排队的一种顺序，使得 n 个人的平均等待时间最小。

【输入】

输入文件共两行，第一行为 n；第二行分别表示第 1 个人到第 n 个人每人的接水时间 T1，T2，……，Tn，每个数据之间有一个空格。

【输出】

输出文件有两行，第一行为一种排队顺序，即 1 到 n 的一种排列；第二行为这种排列方案下的平均等待时间（输出结果精确到小数点后两位）。

【样例输入】(water.in)

```
10
56 12 1 99 1000 234 33 55 99 812
```

【样例输出】(water.out)

```
3 2 7 8 1 4 9 6 10 5
291.90
```

【参考程序】

```
#include <cstdio>
#include <iostream>
#include <iomanip>
#include <algorithm>
using namespace std;

struct node{
    int t;
    int order;
}a[1005];

bool cmp(node x, node y)
{
```

```
    return x.t < y.t;
}

int main()
{
    //freopen("water.in","r", stdin);
    //freopen("water.out", "w", stdout);
    ios::sync_with_stdio(false);

    int i, j, k, p, s;
    cin >> k;

    for(i=1; i<=k; i++) {
       cin >> a[i].t;
       a[i].order = i;
    }

    sort(a+1,a+k+1,cmp);

    for (i=1; i<=k; i++)
        cout<<a[i].order<<' ';
    cout<<endl;

    p = 0;
    s = 0;

    for (i=1; i<=k; i++) {
       p += s;
       s += a[i].t;
    }

    cout << fixed<<setprecision(2)<<p*1.0/k;
    return 0;
}
```

【参考程序】

```
#incldue <cstdio>
#include <iostream>
#include <iomanip>
#include <algorithm>
using namespace std;

int main()
{
   freopen("water.in","r", stdin);
```

```
  freopen("water.out", "w", stdout);
  ios::sync_with_stdin(false);

  int i, j, k, p, s, a[201][3];
  cin >> k;

  for(i=1; i<=k; i++) {
    cin >> a[i][1];
    a[i][2] = i;
  }

  for (i=1; i<=k-1; i++)
    for (j=i+1; j<=k; j++)
      if (a[i][1] > a[j][1])
      {
         swap(a[i][1], a[j][1]);
         swap(a[i][2], a[j][2]);
      }

  for (i=1; i<=k; i++)
    cout<<a[i][2]<<' ';
  cout<<endl;

  p = 0;
  s = a[1][1];

  for (i=2; i<=k; i++) {
    p += s;
    s += a[i][1];
  }

  cout << fixed<<setprecision(2)<<p*1.0/k;
  return 0;
}
```

5. 智力大冲浪 (riddle.cpp)1230

【问题描述】

小伟报名参加中央电视台的智力大冲浪节目。本次挑战赛吸引了众多参赛者，主持人为了表彰大家的勇气，先奖励每个参赛者 m 元。先不要太高兴！因为这些钱还不一定都是你的！接下来，主持人宣布了比赛规则：

首先，比赛时间分为 n 个时段（n<=5000），它又给出了很多小游戏，每个小游戏都必须在规定期限 ti 前完成（1<=ti<=n）。如果一个游戏没能在规定期限前完成，则要从奖励费

m 元中扣去一部分钱 wi，wi 为自然数。不同的游戏扣去的钱数是不一样的。当然，每个游戏本身都很简单，保证每个参赛者都能在一个时段内完成，而且都必须从整时段开始。主持人只是想考考每个参赛者如何安排组织自己做游戏的顺序。作为参赛者，小伟很想赢得冠军，当然更想赢取最多的钱！注意：比赛绝对不会让参赛者赔钱！

【输入】

输入文件 riddle.in，共 4 行。

第 1 行为 m，表示一开始奖励给每位参赛者的钱；

第 2 行为 n，表示有 n 个小游戏；

第 3 行有 n 个数，分别表示游戏 1 到 n 的规定完成期限；

第 4 行有 n 个数，分别表示游戏 1 到 n 不能在规定期限前完成的扣款数。

【输出】

输出文件 riddle.out，仅 1 行，表示小伟能赢取最多的钱。

【样例】

riddle.in：

```
10000
7
4  2  4  3  1  4  6
70  60  50  40  30  20  10
```

riddle.out：

```
9950
```

【参考程序】

```
#include <iostream>
#include <algorithm>
using namespace std;

struct node{
    int time;
    int fine;
}a[105];

bool cmp(node x, node y)
{
    return x.fine > y.fine;
}

bool used[5005];

int main()
{
   //freopen("riddle.in","r", stdin);
   //freopen("riddle.out", "w", stdout);
   ios::sync_with_stdio(false);

   int money, i, j, sum=0;
```

```
   cin >> money;
   cin >> n;

   for(i=0; i<n; i++)
      cin >> a[i].time;

   for (i=0; i<n; i++)
      cin >> a[i].fine;

   sort(a, a+n, cmp);

   for (i=0; i<n; i++)
   {
      int j=a[i].time;  //j=4
      while (used[j]) j--;
      if(j!=0)
         used[j]=true;
      else
         sum += a[i].fine;
   }

   cout << money - sum;
   return 0;
}

#include <cstdio>
#include <iostream>
#include <cstring>
using namespace std;

int i, j, k, n, s, m, a[101], b[101];
bool boo, hash[101];

void sort1()
{
   int i, j;

   for (i= 1; i<=n-1; i++)
      for (j=i+1; j<=n; j++)
         if (b[i] < b[j]) {
            swap(b[i], b[j]);
            swap(a[i], a[j]);
         }
}
```

```
int main()
{
   memset(hash, true, sizeof(hash));
   freopen("riddle.in", "r", stdin);
   freopen("riddle.out", "w", stdout);
   ios::sync_with_stdin(false);

   cin >> m >> n;

   for (i=1; i<=n; i++)  cin >> a[i];

   for (i=1; i<=n; i++)  cin >> b[i];

   sort1();

   for (i=1; i<=n; i++)
   {
      boo = true;
      for (j=a[i]; j>=1; j--)
         if (hash[j]) {
            boo = false;
            hash[j] = false;
            break;
         }
     if (boo) s += b[i];   // 不可安置该活动
   }

   cout << m - s;
   return 0;
}
```

6. **活动选择** (act.cpp)

【问题描述】

假设有一个需要使用某一资源的 n 个活动所组成的集合 S，S={1,…,n}。该资源一次只能被一个活动所占用，每一个活动有一个开始时间 bi 和结束时间 ei(bi<=ei)。若 bi>=ej 或 bj>=ei，则称活动 i 和活动 j 兼容（即两个活动在时间上不重叠）。

你的任务是：选择由互相兼容的活动所组成的最大集合（亦即，找出最多的、相容的活动）。

【输入】

输入文件名为 act.in，共 n+1 行，其中第 1 行为 n，第 2 行到第 n+1 行表示 n 个活动的开始时间和结束时间(中间用空格隔开)，格式为：

n

b1　e1

……

bn　en

【输出】

输出文件名为 act.out，共两行，第 1 行为满足要求的活动占用的时间 t，第 2 行为最大集合中的活动序号，每个数据之间用一个空格隔开。

【样例输入】

```
11
3 5
1 4
12 14
8 12
0 6
8 11
6 10
5 7
3 8
5 9
2 13
```

【样例输出】

```
14
2 3 6 8
```

【问题分析】

{这个问题有什么特点？拿到这个问题后，你最先想到的解法可能是什么？是穷举吗？这种最直觉性的解法有什么不足？可否用动态规划？}

我们使用的贪心策略如下：每一步总是选择这样一个活动来占用资源：它能够使得余下的未调度的时间最大化，使得兼容的活动尽可能多。为了达到这一目的，我们将 n 个待选活动按结束时间递增的顺序排列：e1'<=e2'<=…<=en'。

首先，让这一序列中的活动 1 进入集合 S，再依次分析递增序列中的活动 2、活动 3、……、活动 n，每次将与 S 中的活动兼容的活动加入到集合 S 中。

我们结合问题的样例输入，先将 11 个活动的活动号、开始时间、结束时间及递增编号列表如下：

活动序号 i	开始时间 b[i]	结束时间 e[i]	按活动结束时间递增的序号 j
1	3	5	2
2	1	4	1
3	12	14	11
4	8	12	9
5	0	6	3
6	8	11	8
7	6	10	7
8	5	7	4
9	3	8	5
10	5	9	6

11	2	13	10

所以，问题的解为：t=14，S={2，8，6，3}。根据以上算法编写的程序框架如下：

```
S:={1};    j:=1;  {递增序列中的活动 1 进入  集合 S，设定最近进入 S 的活动序号为 1}
for i:=2 to n do
    if bi'>=ej'   then begin   {若递增序列中的活动 i 与 S 集合中的活动兼容}
        S:=S+[i];    {活动 i 进入集合 S}
        t:=ei';        {计算占用的时间}
        j:=i;          {设定最近进入 S 的活动序号为 i}
    end;
输出占用的时间 t;
for i:=1 to n do
     if i in S then  输出活动 i 的序号;
```

【问题深入】

那么，如何来证明上述的贪心选择得出的解一定就是最优解呢？一般的方法是这样的：

（1）构造一个最优解，然后，对该解进行修正，使其第一步为一个贪心选择，证明总是存在一个以贪心选择开始的求解方案。对于本题，设最优解为 A，第一个选中的活动为 k（k<>1），如果另一个解 B 开始于贪心选择活动 1：

$$B = A - \{k\} \cup \{1\}$$

因 e1'<=ek'，所以 B 中的活动是兼容的。又因为 B 与 A 的活动数相同，所以 B 也是最优的。由此证明总存在一个以贪心选择开始的最优调度。

（2）证明经过若干次贪心选择后，可以得出一个最优解。当我们做出了对活动 1 的贪心选择后，原问题就变成了在活动 2，……，活动 n 中找与活动 1 兼容的那些活动的子问题。亦即，如果问题 A 为原问题的一个最优解，则 A'=A-{1}就是活动选择问题 S'={i∈S|bi'>=e1'}的一个最优解。为什么会这样呢？如果我们能找到一个 S'的含有比 A'更多活动的解 B'，则将活动 1 加入 B'后就得到 S 的一个包含比 A 更多活动的解 B，这就与 A 的最优性相矛盾了。因此，在每一次贪心选择后，留下的是一个与原问题具有相同形式的最优化问题。通过对所做选择次数的归纳可以证明，对每一步进行贪心选择，就可以得到一个最优解。

由此可见，贪心选择的设计及其产生最优解的证明是对实际问题分析归纳的结果。所以，归纳分析对贪心求解问题很关键，有的问题看起来很复杂，但用贪心法求解的话，则会惊人地简单。{问题：哪些问题具有这样的特点？}

【参考程序】

```
#include <cstdio>
#include <iostream>
using namespace std;

struct node{
   int b;
   int e;
}act[100];

bool cmp(node a, node b)
{
```

```
    if (a.e !=b.e)
        return a.e < b.e;
    else
        return a.b > b.b;
}

int main()
{
    int n, i, j, num;

    ios::sync_with_stdio(false);

    cin >> n;

    for (i=0; i<n; i++)
        cin >>act[i].b >> act[i].e;

    sort(act, act+n, cmp);

    num=0;  i=0;
    choosed[num]=i;

    for (i=1; i<n; i++)
    {
        for (j=0; j<=num; j++)
            if (act[i].b < act[choosed[j]].e)  break;
        if (j>num) {
            num++;
            choosed[num]=i;
        }
    }

    cout << num+1;
    return 0;
}
```

7. **种树** (trees.cpp)

【问题描述】

一条街的一边有几座房子，因为环保原因，居民们想要在路边种些树。路边的地区被分割成块，并被编号成 1..N。每个部分为一个单位尺寸大小，并且最多可以种植一棵树。每个居民都想在门前种些树，并要指定三个号码 B、E、T，这三个数表示该居民想在 B 和 E 之间最少种植 T 棵树。当然，B<=E。居民们必须记住在指定区域不能种植多于区域地块数的树，

所以 T<=E-B+1。居民们想种植树的区域可以交叉，你的任务是求出能满足所有要求的最少的树的数量和位置。

【输入】（trees.in）

第一行包含数据 N，表示区域的个数（0<N<=30000）；

第二行包含 H，房子的数目（0<H<=5000）；

下面的 H 行描述了居民们的需要：B E T（0<B<=E<=30000，T<=E-B+1）。

【输出】（trees.out）

输出文件的第一行为树的数目，第二行为所有树的位置，相邻两数之间用一个空格隔开。

【样例】

trees.in:

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

trees.out:

```
5
1  4  5  8  9
```

【分析】

不难想到下面贪心算法：按照每个区间的右端点排序，从左向右扫描，把每个区间内的树都尽量种在该区间的右端，由于后一个区间的右端不在当前这个区间的右端的左边（这是排序的结果），可以保证这些树会尽可能多地被下面的区间利用到。

扫描需要的时间为 O(h)，更新需要的时间为 O(n)，所以，总的时间复杂度为 O(n*h)。

8． 排座椅 (seat.cpp/c/pas)

【问题描述】

上课的时候，总有一些同学和前后左右的人交头接耳，这是令小学班主任十分头疼的一件事情。不过，班主任小雪发现了一些有趣的现象：当同学们的座次确定下来之后，只有有限的 D 对同学上课时会交头接耳。同学们在教室中坐成了 M 行 N 列，坐在第 i 行第 j 列的同学的位置是（i,j），为了方便同学们进出，在教室中设置了 K 条横向的通道、L 条纵向的通道。于是，聪明的小雪想到了一个办法，或许可以减少上课时学生交头接耳的问题：她打算重新摆放桌椅，改变同学们桌椅间通道的位置，因为如果一条通道隔开了两个会交头接耳的同学，那么他们就不会交头接耳了。

请你帮忙给小雪编写一个程序，给出最好的通道划分方案。在该方案下，上课时交头接耳的学生的对数最少。

【输入】

输入文件 seat.in 的第一行，有 5 个空格隔开的整数，分别是 M、N、K、L、D（2<=N, M<= 1000，0<=K < M，0<=L <N，D<= 2000）。

接下来的 D 行，每行有 4 个用空格隔开的整数。第 i 行的 4 个整数 X_i、Y_i、P_i、Q_i，表示坐在位置（X_i，Y_i）与（P_i，Q_i）的两个同学会交头接耳（输入保证他们前后相邻或者

左右相邻)。

输入数据保证最优方案的唯一性。

【输出】

输出文件 seat.out 共两行。

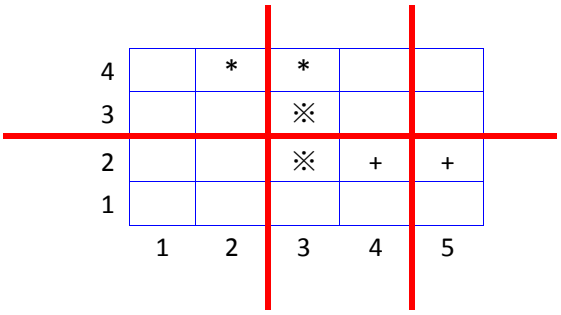
第一行包含 K 个整数 a_1 a_2 …… a_k，表示第 a_1 行和 a_1+1 行之间、第 a_2 行和 a_2+1 行之间、…、第 a_k 行和第 a_k+1 行之间要开辟通道，其中 $a_i < a_i+1$，每两个整数之间用空格隔开（行尾没有空格）。

第二行包含 L 个整数，b_1 b_2 …… b_L，表示第 b_1 列和 b_1+1 列之间、第 b_2 列和 b_2+1 列之间、…、第 b_L 列和第 b_L+1 列之间要开辟通道，其中 $b_i<b_i+1$，每两个整数之间用空格隔开（行尾没有空格）。

【输入输出样例】

seat.in	**Seat.out**
4 5 1 2 3 4 2 4 3 2 3 3 3 2 5 2 4	2 2 4

【输入输出样例解释】



上图中用符号*、※、+标出了 3 对会交头接耳的学生的位置，图中 3 条粗线的位置表示通道，图示的通道划分方案是唯一的最佳方案。

【参考程序】{C++}

```
#include <cstdio>
#include <iostream>
using namespace std;

int main()
{
   int m, n, k, L, d;

   cin>>m>>n>>k>>L>>d;
   int b[d+1[4+1], hang[m][3], lie[n][3];

   for (i=1; i<=d; i++)
   {
      cin>>b[1][1]>>b[i][2]>>b[i][3]>>b[i][4];
```

```
    if (b[i][1]==b[i][3])
       lie[min(b[i][2],b[i][4])][1]++;
    else
       hang[min[(b[i][1],b[i][3])[1]++;
  }

  for (i=1; i<=m-1; i++)   hang[i][2] = i;

  for (i=1; i<=n-1; i++)  lie[i][2]=i;

  for (i=1; i<=m-2; i++)
    for (j=i+1; j<=m-1; j++)
      if (hang[i][1]<hang[j][1]) {
         swap(hang[i][1],hang[j][1]);
         swap(hang[i][2],hang[j][2]);
      }

  for (i=1; i<=n-2; i++)
    for (j=i+1; j<=n-1; j++)
      if (lie[i][1]<lie[j][1]){
         swap(lie[i][1],lie[j][1]);
         swap(lie[i][2],lie[j][2]);
      }

  for (i=1; i<=k; i++)  cout<<hang[i][2];
  cout << endl;

  for (i=1; i<=L; i++)  cout << lie[i][2];
  return 0;
}
```