子序列问题

信智学院　陈琰宏

教学内容

01 上升子序列

02 连续字段和

03 公共子序列

01 案例讲解

[2811] 求最长不下降序列

有由n个不相同的整数组成的数列，记为:a[1]、a[2]、…a[n]且 a[i]！=a[j] (i!=j)，若存在i1<i2<i3< … < ie且有a[1]<a[2]< … <a[n]则称为长度为e的不下降序列。

程序要求，当原数列给出之后，求出最长的不下降序列。

例如13，7，9，16，38，24，37，18，44，19，21，22，63，15。例中13，16，18，19，21，22，63就是一个长度为7的不下降序列，同时也有7 ，9，16，18，19，21，22，63长度为8的不下降序列。

算法分析

根据动态规划的原理，由由往前进行搜索(当然从后往前也一样)。

1 对a[1]来说，由于它是第一个数，所以当从a[1]开始查找时，只存在长度为1的不下降序列；

2 若从a[2]开始查找，则存在下面的两种可能性：

①若a[1]<a[2]则存在长度为2的不下降序列a[1],a[2]。

②若a[2]>a[1]则存在长度为1的不下降序列a[1]或a[2]。

3 一般若从a[i]开始，此时最长不下降序列应该按下列方法求出:

在a[1],a[2],…,a[i]中，找出一个比a[i]小的且最长的不下降序列，作为它的后继。

求解过程

一般处理过程是：

①在1,2,…,i-1项中，找出比a[i]小的最长长度L;

②若L>0，则dp[i]=L+1;

最后本题经过计算，其数据存储表如下：

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
a[i]	13	7	9	16	38	24	37	18	44	19	21	22	63	15
dp[i]														

1	2	3	4	5	6	7	8	9	10	11	12	13	14
13	7	9	16	38	24	37	18	44	19	21	22	63	15
1	1	2	3	4	4	5	4	5	5	6	7	8	3

状态转移

1	2	3	4	5	6	7	8	9	10	11	12	13	14
13	7	9	16	38	24	37	18	44	19	21	22	63	15
1	1	2	3	4	4	5	4	5	5	6	7	8	3

dp[i]=max(dp[k]+1)
(a[k]<a[i]&&dp[k]=max(dp[j]) j<i)

从前往后

```
4  int main(){
5      int n;
6      cin>>n;
7      for(int i=1;i<=n;i++){
8          cin>>a[i];
9          __1_____________;//1初始化
10     }
11     for(int i=2;i<=n;i++){
12         for(int j=__2_____________){
13             if(__3_____________)//2状态转移
14                 dp[i]=_4_____________;
15         }
16     }
17     int maxn=-1e9;
18     for(int i=1;i<=n;i++)
19         maxn=max(maxn,dp[i]);
20     cout<<"max="<<maxn;
```

从后往前

```
#include<iostream>
using namespace std;
int main()
{
   int n,i,j,maxl,k,a[200],b[200];
   cin>>n;
   for (i=1;i<=n;i++) //输入序列的初始值
    {
      cin>>a[i];
      b[i]=1;
    }
   for (i=n-1;i>=1;i--)//从后往前
    {
        maxl=0;k=0;
        for (j=i+1;j<=n;j++)
         if ((a[j]>a[i])&&(b[j]>maxl))
             maxl=b[j];
        b[i]=maxl+1;
    }
     k=1;//下面找出最长不下降序列
     for (j=1;j<=n;j++)//求最长不下降序列的起始位置
      if (b[j]>k) k=b[j];
     cout<<"max="<<k<<endl;
}
```

[2816] 合唱队形

N位同学站成一排，音乐老师要请其中的(N-K)位同学出列，使得剩下的K位同学排成合唱队形。

合唱队形是指这样的一种队形：设K位同学从左到右依次编号为1,2,…,K，他们的身高分别为T1,T2,…,TK，则他们的身高满足T1<T2<…<Ti,Ti>Ti+1>…>TK(1≤i≤K)。

你的任务是，已知所有N位同学的身高，计算最少需要几位同学出列，可以使得剩下的同学排成合唱队形。

输入：输入的第一行是一个整数N（2≤N≤100），表示同学的总数。第二行有n个整数，用空格分隔，第ii个整数Ti（130≤Ti≤230）是第i位同学的身高（厘米）。

输出：输出包括一行，这一行只包含一个整数，就是最少需要几位同学出列。

样例输入

8

186 186 150 200 160 130 197 220

样例输出

4

分析

这道题是找出一个最长的子序列，该子序列需要满足一个类似于山的形状的大小关系，思路就是算出从每个点为起点向左的下降子序列向右的上升序列，然后遍历每个点找到左边长度加上右边长度的最大值，再用总数减掉最大值即答案。

1	2	3	4	5	6	7	8	9	10	11	12	13	14
13	7	9	16	38	24	37	18	44	19	21	22	63	15
1	1	2	3	4	4	5	4	5	5	6	7	8	3
2	1	1	2	4	3	3	2	3	2	2	2	2	1

```cpp
#include<bits/stdc++.h>
using namespace std;
int a[300],dp1[501],dp2[501];
int main(){
    int i,j,n,m=0;
    cin>>n;
    for(i=1;i<=n;i++)
        cin>>a[i];
    for(i=1;i<=n;i++){
        dp1[i]=1;
        for(j=1;j<i;j++){
            if(a[j]<a[i])
                dp1[i]=max(dp1[i],dp1[j]+1);
        }
    }
```

```
16      for(i=n;i>=1;i--){
17          dp2[i]=1;
18          for(j=i+1;j<=n;j++){
19              if((a[j]<a[i]))
20                  dp2[i]=max(dp2[i],dp2[j]+1);
21          }
22      }
23      for(i=1;i<=n;i++)
24          if(dp1[i]+dp2[i]>m)
25              m=dp1[i]+dp2[i];
26      cout<<n-m+1<<endl;
27  }
```

[2833] 最长上升子序列2

给定一个长度为N的数列，求数值严格单调递增的子序列的长度最长是多少。

输入格式

第一行包含整数N。

第二行包含N个整数，表示完整序列。

输出格式

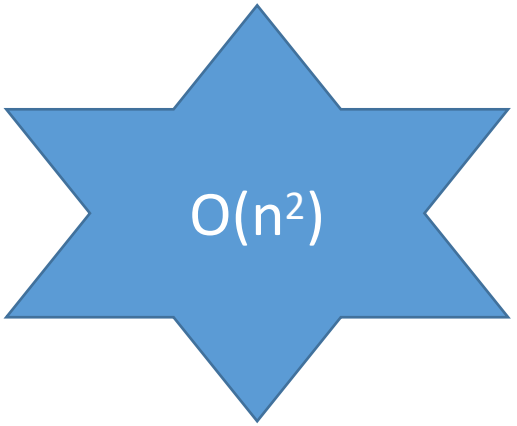
输出一个整数，表示最大长度。

数据范围

$1 \leqslant N \leqslant 100000$，

$-10^9 \leqslant$ 数列中的数 $\leqslant 10^9$

```cpp
#include<bits/stdc++.h>
using namespace std;
int dp[205],a[205];
int main(){
    int n;
    cin>>n;
    for(int i=1;i<=n;i++){
        cin>>a[i];
        dp[i]=1;
    }
    for(int i=2;i<=n;i++){
        for(int j=1;j<i;j++){
            if(a[j]<a[i])
                dp[i]=max(dp[i],dp[j]+1);
        }
    }
    sort(dp+1,dp+1+n);
    cout<<"max="<<dp[n];
    return 0;
}
```



1	2	3	4	5	6	7	8	9	10	11	12	13	14
13	7	9	16	38	24	37	18	44	19	21	22	63	15

17	9	16	38	37	41	22	65						

思路

首先数组a中存输入的数（原本的数），开辟一个数组f用来存结果，最终数组f的长度就是最终的答案；假如数组f现在存了数，当到了数组a的第i个位置时，首先判断a[i] > f[cnt] ？ 若是大于则直接将这个数添加到数组f中，即f[++cnt] = a[i];这个操作时显然的。

当a[i] <= f[cnt] 的时,我们就用a[i]去替代数组f中的第一个大于等于a[i]的数，因为在整个过程中我们维护的数组f 是一个递增的数组，所以我们可以用二分查找在 logn 的时间复杂的的情况下直接找到对应的位置，然后替换，即f[l] = a[i]。

我们用a[i]去替代f[i]的含义是：以a[i]为最后一个数的严格单调递增序列,这个序列中数的个数为l个。

这样当我们遍历完整个数组a后就可以得到最终的结果。

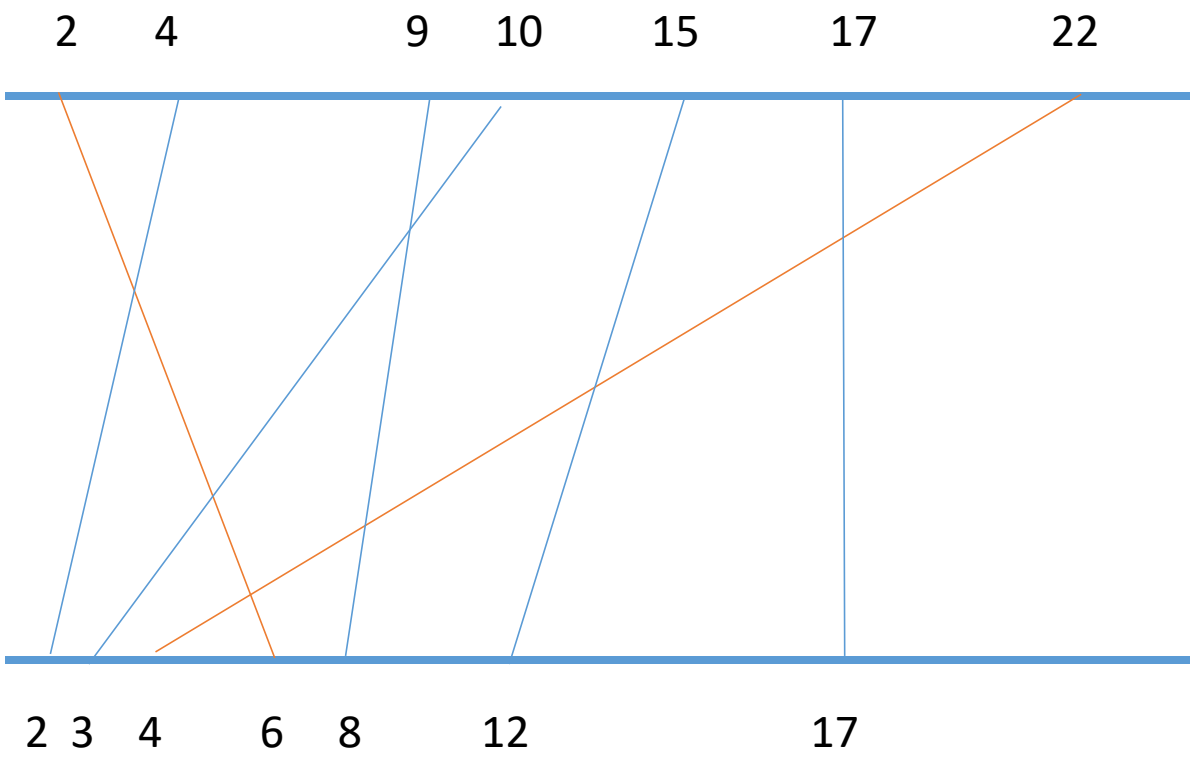
```cpp
#include<bits/stdc++.h>
using namespace std;
int a[100005], f[100005];
int cnt;
int find(int x) {
//二分查找f数组中第一个比a[i]大的数的下标
//用a[i]去替代数组f中的第一个大于等于a[i]的数
    int l = 1, r = cnt;
    int ans=1;
    while(l <= r) {
        int mid = l+r >> 1;
        if(f[mid] >= x) {
            ans=mid;
            r = mid-1;//可以更小
        }
        else l = mid + 1;
    }
    return ans;
}
```

```
20  int main(){
21      int n;
22      scanf("%d", &n);
23      for(int i = 1; i <= n; i ++) scanf("%d", &a[i]);
24      f[++cnt] = a[1];
25      for(int i = 2; i <= n; i ++)
26          if(a[i] > f[cnt]) f[ ++ cnt] = a[i];
27              //成立则末尾插入 不成立则前面替换
28          else {
29              int tmp = find(a[i]);
30              f[tmp] = a[i];
31          }
32      printf("%d\n", cnt);
33      return 0;
34  }
```

[2815] 友好城市

Palmia国有一条横贯东西的大河，河有笔直的南北两岸，岸上各有位置各不相同的N个城市。北岸的每个城市有且仅有一个友好城市在南岸，而且不同城市的友好城市不相同。
每对友好城市都向政府申请在河上开辟一条直线航道连接两个城市，但是由于河上雾太大，政府决定避免任意两条航道交叉，以避免事故。
编程帮助政府做出一些批准和拒绝申请的决定，使得在保证任意两条航线不相交的情况下，被批准的申请尽量多。
输入
第1行，一个整数N(1≤N≤5000)，表示城市数。
第2行到第n+1行，每行两个整数，中间用1个空格隔开，分别表示南岸和北岸的一对友好城市的坐标。(0≤xi≤10000)
输出
仅一行，输出一个整数，表示政府所能批准的最多申请数。

样例输入
7
22 4
2 6
10 3
15 12
9 8
17 17
4 2
样例输出
4



```cpp
int dp[5005];
int main(){
    int n;
    cin>>n;
    for(int i=1;i<=n;i++){
        cin>>cc[i].n>>cc[i].s;
    }
    sort(cc+1,cc+1+n,cmp);
    for(int i=1;i<=n;i++){
        dp[i]=1;
        for(int j=1;j<i;j++){
            if(cc[j].n<cc[i].n)
                dp[i]=max(dp[i],dp[j]+1);
        }
    }

    int maxn=dp[1];
    for(int i=2;i<=n;i++)
        maxn=max(dp[i],maxn);
    cout<<maxn;
    return 0;
}
```

```cpp
struct city{
    int n;
    int s;
}cc[5005];

int cmp(city a, city b){
    return a.s<b.s;
}
```

2.7 [2841] 拦截导弹问题

某国为了防御敌国的导弹袭击，开发出一种导弹拦截系统，但是这种拦截系统有一个缺陷：虽然它的第一发炮弹能够到达任意的高度，但是以后每一发炮弹都**不能高于前一发的高度**。某天，雷达捕捉到敌国的导弹来袭，由于该系统还在试用阶段。所以一套系统有可能不能拦截所有的导弹。

输入导弹依次飞来的高度（雷达给出的高度不大于30000的正整数）。计算要拦截所有导弹最小需要配备多少套这种导弹拦截系统。

输入 n颗依次飞来的高度（1≤n≤1000）.

输出要拦截所有导弹最小配备的系统数k。

【输入样例】389　207　155　300　299　170　158　65

【输出样例】2

分析 1989 1220 616 1619 1606 1981 118 123 456 789

1	1989	1220	616	118	
2	1619	1606	123		
3	1981	456			
4	789				
5					

按照题意，被一套系统拦截的所有导弹中，最后一枚导弹的高度最低。

若导弹i的高度高于所有系统的最低高度，则断定导弹i不能被这些系统所拦截，应增设一套系统来拦截导弹I

若导弹i低于某些系统的最低高度，那么导弹i均可被这些系统所拦截。

分析

设：

k为当前配备的系统数；

len[k]为被第k套系统拦截的最后一枚导弹的高度，简称系统k的最低高度（1≤k≤n）。

我们首先设导弹1被系统1所拦截（k←1,len[k]←导弹1的高度）。然后依次分析导弹2，…，导弹n的高度。

贪心策略: 选择其中最低高度最小（即导弹i的高度与系统最低高度最接近）的一套系统p(len[p]=min{len[j]| len[j]>导弹i的高度}。这样可使得一套系统拦截的导弹数尽可能增多。依次类推，直至分析了n枚导弹的高度为止。此时得出的k便为应配备的最少系统数。

框架

```
k=1;len[k]=导弹1的高度;
for (i=2;i<=n;++i)
{     p=0;//一套系统p
       for (j=1;j<=k;j++)
       if (len[j]>=导弹i的高度) {
                  if (p==0) p=j;
                  else if (len[j]<len[p]) p=j;
        }   //贪心
       if (p==0) {
             k++;len[k]=导弹i的高度;
        }                 //增加一套新系统
        else
             len[p]=导弹i的高度; //贪心,更新第p套系统的最低高度
}
```

输出应配备的最少系统数K。

1989 1220 616 1619 1606 1981 118 123 456 789

1	1989	1220	616	118	
2	1619	1606	123		
3	1981	456			
4	789				
5					

程序

```
1   #include<bits/stdc++.h>
2   using namespace std;
3   int s[1005];//存储导敌国的导弹高度
4   int len[1005];//第k套系统拦截的最后一枚导弹的高度
5   int main()
6   {
7       int n;
8       n=1;
9       memset(s,0,sizeof(s));
10      memset(len,0,sizeof(len));
11      while(cin>>s[n]){
12          n++;//计算导弹数量
13      }
14      int k=1;//当前配备的系统数
15      len[k]=s[1];//把第一个导弹的高度作为第一套系统的初始高度
16      int p;//表示某一套系统
17      for(int i=2;i<n;i++)
18      {//遍历每一颗导弹
39      cout<<k<<endl;
40      return 0;
41  }
```

程序

```
17      for(int i=2;i<n;i++)
18      {//遍历每一颗导弹
19          p=0;//记录当前导弹j属于哪套系统
20          for(int j=1;j<=k;j++)//一共有k套系统
21          {
22              if(len[j]>=s[i])//从k套系统选择
23              {
24                  if(p==0)p=j;//p赋值为第一个满足条件的第j套系统
25                  else if(len[j]<len[p]) p=j;
26                  //贪心，选择最小系统中最小的值，
27                  //然后后面会把这个比最小系统还小的值赋值给
28                  //这个最小系统的最小值
29              }
30          }
31          if(p==0)
32          {//没有找到合适的系统，生成新的导弹系统
33              k++;
34              len[k]=s[i];//把s[i]作为第k套系统的最小值
35          }
36          else
37              len[p]=s[i];//到合适的系统,把s[i].....
38      }
39      cout<<k<<endl;
```

填空

```
17      for(int i=2;i<n;i++)
18      {//遍历每一颗导弹
19          p=0;//记录当前导弹j属于哪套系统
20          for(int j=1;j<=k;j++)//一共有k套系统
21          {
22              if(_____________)//从k套系统选择
23              {
24                  if(p==0)_____________;//p赋值为第一个满足条件的第j套系统
25                  else if(_____________) p=j;
26                  //贪心，选择最小系统中最小的值，
27                  //然后后面会把这个比最小系统还小的值赋值给
28                  //这个最小系统的最小值
29              }
30          }
31          if(p==0)
32          {//没有找到合适的系统，生成新的导弹系统
33              _____________
34              _____________//把s[i]作为第k套系统的最小值
35          }
36          else
37              _____________//到合适的系统,把s[i].....
38      }
39      cout<<k<<endl;
```

2.7 [2838] 怪盗基德的滑翔翼

怪盗基德是一个充满传奇色彩的怪盗，专门以珠宝为目标的超级盗窃犯。而他最为突出的地方，就是他每次都能逃脱中村警部的重重围堵，而这也很大程度上是多亏了他随身携带的便于操作的滑翔翼。

有一天，怪盗基德像往常一样偷走了一颗珍贵的钻石，不料却被柯南小朋友识破了伪装，而他的滑翔翼的动力装置也被柯南踢出的足球破坏了。不得已，怪盗基德只能操作受损的滑翔翼逃脱。

2.7 [2838] 怪盗基德的滑翔翼

假设城市中一共有N幢建筑排成一条线，每幢建筑的高度各不相同。初始时，怪盗基德可以在任何一幢建筑的顶端。他可以选择一个方向逃跑，但是不能中途改变方向（因为中森警部会在后面追击）。因为滑翔翼动力装置受损，他只能往下滑行（即：只能从较高的建筑滑翔到较低的建筑）。他希望尽可能多地经过不同建筑的顶部，这样可以减缓下降时的冲击力，减少受伤的可能性。请问，他最多可以经过多少幢不同建筑的顶部(包含初始时的建筑)。

输入数据第一行是一个整数K(K<100)，代表有K组测试数据。每组测试数据包含两行：第一行是一个整数N(N<100)，代表有N幢建筑。第二行包含N个不同的整数，每一个对应一幢建筑的高度h(0<h<10000)，按照建筑的排列顺序给出。

对于每一组测试数据，输出一行，包含一个整数，代表怪盗基德最多可以经过的建筑数量。

2.7 [2838] 怪盗基德的滑翔翼

样例输入

3
8
300 207 155 299 298 170 158 65
8
65 158 170 298 299 155 207 300
10
2 1 3 4 5 6 7 8 9 10

样例输出

6
6
9

这道题根据题意，其实就是求最长上升子序列的长度，但是又是任意选择起点和方向，事实是就是去最长上升子序列和最长下降子序列的最大值。

[1140] 字母排序

XXXX年突然有外星人造访，但大家语言不通，不过科学家们经过研究发现外星人用26个英文字母组成的单词中最长不降子序列的长度来表述数字，且英文字母的排列顺序不同，现给出其排列顺序，再给出外星人说的每个数字（其实是每个英文单词，用空格隔开），翻译出外星人所说的数字（连续输出，最后加回车）。
(因为是最长不降子序列，所以数字中没有0，也就是说外星人的数字是>=1的数字)
例如
我们正常的字母排列顺序是abcdefg.....xyz，
代表a< b< c< .....< x< y< z
abcd efg hhh ihg 四个字符串的最长不降子序列的长度分别为
4 3 3 1

字母排序

输入
第1，2行为字符串 含义如题描述

输出
输出答案 含义如题描述

样例输入
```
abcdefghijklmnopqrstuvwxyz
abcd efg hhh ihg
```

样例输出
```
4331
```

分析—字典序样例

假设按照字典序abcdefghijklmnopqrstuvwxyz(a< b< c< …..< x< y< z)作为计算基值。

有外星数字：

a	b	c	d	e	f	g
1	2	3	4	5	6	7

a　ab　abc　abcd　abcde　abcdef　abcdefg

下表

有外星数字zzzhqgs，根据字典可得到外星数字截止每个字母可构成的最长不降子序列长度如下表

z	z	z	h	g	g	s
1	2	3	1	1	2	3

z　zz　zzz　h　g　gg　ggs

有外星数字zyxwvba，根据字典可得到外星数字截止每个字母可构成的最长不降子序列长度如下表

z	y	x	w	v	b	a
1	1	1	1	1	1	1

z　y　x　w　v　b　a

分析— 自定义字典样例

假设按照字母排序stuvwxyzaqrdefghijklbcmnop （s<t<u<v…<n<o<p)作为计算基值。

有外星数字:abcdefg,根据字典可得到外星数字截止每个字母可构成的最长不降子序列长度如下表

c[i]	a	b	c	d	e	f	g
f[j]	1	2	3	2	3	4	5

a　　ab　　abc　　ad　　ade　　adef　　adefg

记f[i]为第i个字母能够成的最长子序列
有f[i]=max(f[0..j])+1 且 c[i]>=c [j]
即当前位置i能组成的最长不降子序列等于0到i之间最长子序列+1

分析：问题分解

根据问题描述及样例分析，可以发现题目需要解决的核心问题有三个

一、字母顺序重排列

二、找到字符串中的最长不降子序列

三、处理多组字符串

字母的顺序的排布

将字符顺序重新编码，可以设置一个数组pow,数组的下标为字符本身的ascii码值。数组内容存放字母出现的顺序

对于stuvwxyzaqrdefghijklbcmnop

可以如下记录新的字符顺序

```
pow['s']=1,pow['t’]=2 … pow['o']=25,pow['p']=26
```

或者

```
pow['s'-'a']=1,pow['t'-'a']=2 … pow['o'-'a']=25,pow['p'-'a']=26
```

找到字符串中的最长子序列

根据对样例的分析，我们已经拿到了如何计算每一个字符构成的最长不降子序列的方式。

记f[i]为第i个字母能够成的最长子序列
有f[i]=max(f[0...i-1])+1 且 c[i]>=c [j]
即当前位置i能组成的最长不降子序列等于0到i之间最长子序列+1

可以先不考虑自定义字典序，假设外星人使用的数字字母就是普通的字典序，求外星人数字就可以简化为求一串字符串的最长不降子序列

```
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
    int zh[27]={0};
    string str ;
    cin >> str;
    for (int i=0; i<=25;i++)//转化为数值
        zh[str[i]-'a'+1]=i+1 ;
    string s ;
    int dg[2005]={0} ;
    int kk=1;
    while (cin>>s)
    {
    for (int i=1;i<= kk-1; i++)
    {
        cout<<dg[i] ;
    }
}
```

```
13      int kk=1;
14      while (cin>>s)
15      {
16          int a[299]={0} ;
17          int b[299]={0} ;//保存序列长度
18          memset(b,0,sizeof(b)) ;
19          int max=-1 ;
20          for (int i=0; i<= s.length()-1 ;i++)
21              a[i+1]=zh[s[i]-'a'+1] ;//
22          for (int i =1; i<= s.length();i++ )
23          {
24              b[i]=1 ;
25              for (int j=1 ; j <= i - 1 ; j ++)
26              {
27                  if (a[i] >= a[j] && b[j]+1>b[i])
28                  {
29                      b[i]=b[j]+1 ;
30                  }
31              }
32          }
```

```
33          for (int i=1 ; i<= s.length();i++)
34          {
35              if (b[i]>max)
36              {
37                  max=b[i] ;
38              }
39
40          }
41          dg[kk]=max ;
42          kk ++ ;
43
44      }
45      for (int i=1;i<= kk-1; i++)
46      {
47          cout<<dg[i] ;
48      }
49  }
```

[3412] 猴子与香蕉

一组研究人员正在设计一个测试猴子IQ的实验。他们把香蕉吊在屋顶上，同时给猴子提供了砖块。如果猴子够聪明，它会把砖块一个个叠起来做成一个塔，然后爬上去拿到自己喜爱的食物。

研究人员有n种不同的砖块，而且每种砖块都是取之不尽的。每种砖块都是长方体，第i种砖块的大小是（xi,yi，zi）。砖块能够翻转，可以将任意两边当作底面，剩下的那边作为高。

他们想确定用砖块搭成的最高塔，能否帮助猴子够着屋顶。问题是，在叠塔过程中，要放的那块砖，其底面两条边都要小于下面那块砖的两条边，这是为了留个空间给猴子踩脚。例如，底面相同尺寸的砖块不能相叠。

现给定砖块，请计算猴子能够叠塔的最大高度。

输入描述：

输入包含多组测试数据。每组输入的第一行是一个整数n，表示砖块的种类数。n的最大值是30。

接着n行，每行输入三个整数xi，yi和zi。

当n=0时，输入结束。

输出描述：

对于每组输入，输出一行：测试例编号case（从1开始编号），塔能够达到的最大高度height。

输出格式为：“Case case: maximum height = height”。

第一组数据：

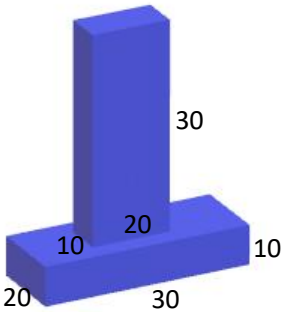
一个砖块转来转去的很麻烦，那我们为什么不能把一个砖块的六种状态当成六个不同的砖块呢？

再规定宽的长度比长要小，每一类砖块有三种不同状态。

	1	2	3
宽(w)	10	10	20
长(l)	20	30	30
高(h)	30	20	10

样例输出：

```
Case 1: maximum height = 40
Case 2: maximum height = 21
Case 3: maximum height = 28
Case 4: maximum height = 342
```



样例输入：

```
1
10 20 30
2
6 8 10
5 5 5
7
1 1 1
2 2 2
3 3 3
4 4 4
5 5 5
6 6 6
7 7 7
5
31 41 59
26 53 58
97 93 23
84 62 64
33 83 27
0
```

第二组数据：

两类砖块，每一类排列出3种，共6种排列方式

```
2
6 8 10
5 5 5
```

	1	2	3	4	5	6
宽(w)	6	6	8	5	5	5
长(l)	8	10	10	5	5	5
高(h)	10	8	6	5	5	5

先宽从小到大排列，再按长从小到大排

	1	2	3	4	5	6
宽(w)	5	5	5	6	6	8
长(l)	5	5	5	8	10	10
高(h)	5	5	5	10	8	6

建立height数组，表示当前第i个砖块能达到的最大高度

然后遍历所有砖块，对于每个砖块，检查从这个砖头往前的所有砖块是否满足长宽都小于当前砖块，找出里面的最大height值，累加到当前的height中

	1	2	3	4	5	6
宽(w)	5	5	5	6	6	8
长(l)	5	5	5	8	10	10
高(h)	5	5	5	10	8	6
height	5	5	5	5+10	5+8	15+6

最后我们所要求的是：从所有砖块的最大高度中找出最大值即可

思路

1.先定义一个结构体，用来盛放砖块（包括宽w、长l、高h），然后需要按照砖块的宽、长来排序。

2.用一个数组c[i]，保存一类砖块的长宽高，然后演变成3种状态，保存到结构体中，再按照宽长进行排序。

3.用一个数组height[i]存放每一种砖块能达到的最大高度，然后根据要求找出砖块前的最大高度值，累加上自身高度存放到数组height[i]中，最后算出数组height[i]的最大值即可。

一、建立结构体

```
#include<bits/stdc++.h>
using namespace std;
struct Bricks {
    int w,l,h;
    //w表示宽，l表示长，h表示高，规定宽的长度比长要小
}a[91];
bool cmp(Bricks qian, Bricks hou)
{
    //先宽度排序，再按长度排序
    if (qian.w==hou.w) return qian.l<hou.l;
    else return qian.w<hou.w;
}
```

二、输入砖块，建立不同状态，保存到结构体后排序

```
13  int main()
14  {
15      int n,i,j,maxn,c[4];
16      //数组c用来保存某一类砖块的长，宽，高的三个数据
17      int k, num=1;  //测试数据组数，初始值为1
18      while (cin>>n) {
19          if (n==0) break;
20          k=1; //k表示不同类型的砖块数量
21          int maxn=0; //maxn保存所求结果，初始为0
22          for (j=1; j<=n; j++)
23          {
24              for (i=1;i<=3;i++)cin>>c[i];
25              sort(c+1, c+4);
26              //长宽高的三个数据从小到大排序，这样一个砖块可以有三种排列方式，
27              //把他当成三个不同类型的砖块 //三种排列方式赋值到结构体中
28              a[k].w=c[1];a[k].l=c[2];a[k].h=c[3];k++;
29              a[k].w=c[2];a[k].l=c[3];a[k].h=c[1];k++;
30              a[k].w=c[1];a[k].l=c[3];a[k].h=c[2];k++;
31          }
32          sort(a+1, a+k,cmp);  //所有类型的砖块按宽跟长从小到大排序
```

三、求每一种砖块能达到的最大高度height[i]

```
33          int dp[3*n+1];  //数组h用来保存每一类砖块能达到的最大高度
34          for (i=1; i<k;i++)
35          {
36              dp[i]=a[i].h;
37              for (j=1;j<i;j++)
38              {
39                  if(a[j].w<a[i].w && a[j].l<a[i].l)
40                      dp[i]=max(dp[i],dp[j]+a[i].h);
41                      //累加到当前的高度中
42              }
43              if (dp[i]>maxn)maxn=dp[i];
44          }
45          cout << "Case " <<num<< ": maximum height = "<<maxn << endl;
46          num++;
47      }
48      return 0;
49  }
```

股票交易

最近越来越多的人都投身股市，阿福也有点心动了。谨记着“股市有风险，入市需谨慎”，阿福决定先来研究一下简化版的股票买卖问题。

假设阿福已经准确预测出了某只股票在未来N天的价格，他希望**买卖两次**，使得获得的利润最高。为了计算简单起见，利润的计算方式为卖出的价格减去买入的价格。

同一天可以进行多次买卖。但是在第一次买入之后，必须要先卖出，然后才可以第二次买入。

现在，阿福想知道他最多可以获得多少利润。

输入描述： 输入的第一行是一个整数T(T≤50)，表示一共有T组数据。

接下来的每组数据，第一行是一个整数N(1≤N≤100,000)，表示一共有N天。第二行是 N 个被空格分开的整数，表示每天该股票的价格。该股票每天的价格的绝对值均不会超过1,000,000。

输出描述： 对于每组数据，输出一行。该行包含一个整数，表示阿福能够获得的最大的利润。

样例输入：

```
3
7
5  14  -2  4  9  3  17
6
6   8  7  4  1  -2
4
18  9  5  2
```

样例输出：

```
28
2
0
```

样例讲解

样例输入：

```
3
7
5 14 -2 4 9 3 17
6
6 8 7 4 1 -2
4
18 9 5 2
```

样例输出：

```
28
2
0
```

对于第一组样例，阿福可以第1次在第1天买入（价格为5），然后在第2天卖出（价格为14）。第2次在第3天买入（价格为-2），然后在第7天卖出（价格为17）。一共获得的利润是(14-5)+(17-(-2))=28

对于第二组样例，阿福可以第1次在第1天买入（价格为6），然后在第2天卖出（价格为8）。第2次仍然在第2天买入，然后在第2天卖出。一共获得的利润是8-6=2。

对于第三组样例，由于价格一直在下跌，阿福可以随便选择一天买入之后迅速卖出。获得的最大利润为0。

算法分析

题目要求：股票需要进行两次交易。

我们先假设，只继续一次交易后能够获取到的最大利润

对于第一组数据：5　14　-2　4　9　3　17

1，对于gp [1]来说，由于它是第一个数，同时设置为最低的价格min

所以lr[1]开始计算时，只有一个价格，所以利润为0

2，如从gp[2] 开始查找时，则存在下面的两种可能性：

1)若gp[2]<=min，为了不亏损，利润依旧为0，并且最低价格min需要更换

2)若gp[2]>min,　　这时存在利润：a[2]-min，最低价格min依旧不变

3，　一般若从gp[i] 开始，此时求能够获取到的最大利润应该按下列方法求出：

在gp[i+1],gp[i+2],……,gp[n]中，找到一个比lr[i]大，并且是利润中最大的值，作为它的后继。

求解过程

定义gp[i], lr[i],min;

1) gp[i] 表示第i天的股票价格

2) lr[i] 表示从第1天~第i天中，能够获取到的最大利润

3) min表示从第1天~第i天中，股票交易的最低价格

1） 从第二天开始计算，前面仅有1项，比较一次，因14>5，有利润可图，所以lr[2]=14-5=9， min=5不变。

2） 第三天，需要第三天的股票价格与股票的最小价格比较，如下表：

gp[i]	**5**	**14**	**-2**	**4**	**9**	**3**	**17**
lr[i]	0	9	9	9	11	11	19
min	5	5	-2	-2	-2	-2	-2

最后输出 数组 lr[i] 的最后一项，即为一次交易后的能够获取到的最大利润

算法分析

弄明白 一次交易后能够获取到的最大利润

接下来题目要求，需进行两次的股票交易后能够获取到的最大利润

过程： 第一次买入之后，必须要先卖出，然后才可以第二次买入

从这里得到结论：两次的股票交易，都是独立的交易

第一次交易： 1 ~ i 天； 第二次交易： i ~ n 天

难点： 如何处理第二次交易的交易范围？

与第一次处理相反，我们从后往前进行计算

为算法上的需要，定义整数类型gp[i], lr2[i],max;

1) gp[i] 表示第i天的股票价格

2) lr2[i] 表示从第i天~第n天中，能够获取到的最大利润

3) max表示从第i天~第n天中，股票的最高价格

求解过程

1） 从第n-1天开始计算，后面仅有1项，比较一次，因3<17，有利润可图，所以lr2[n-1]=17-3=14， max=17不变。

2） 倒数第三天，需要当天的股票价格与股票的最大价格比较，如下表：

gp[i]	**5**	**14**	**-2**	**4**	**9**	**3**	**17**
lr2[i]	19	19	19	14	14	14	0
max	17	17	17	17	17	17	17

算法分析

从前往后：

gp[i]	**5**	**14**	**-2**	**4**	**9**	**3**	**17**
lr[i]	0	9	9	9	11	11	19
min	5	5	-2	-2	-2	-2	-2

从后往前：

gp[i]	**5**	**14**	**-2**	**4**	**9**	**3**	**17**
lr2[i]	19	19	19	14	14	14	0
max	17	17	17	17	17	17	17

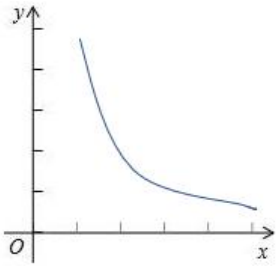
两次交易后能够获取的最大利润：

第1～i天的利润 + 第i ～ n天的利润 之和的最大值

```cpp
#include<bits/stdc++.h>
#include<algorithm>
#include<cstring>
using namespace std;
int ans,t,n,gp[100001],Min,Max,lr[100001],lr2[100001];
int main()
{
    scanf("%d",&t);
    while (t--)
    {
        memset(lr,0,sizeof(lr));        //初始化
        memset(lr2,0,sizeof(lr2));
        scanf("%d",&n);
        for (int i=1;i<=n;i++)
        scanf("%d",&gp[i]);             //输入每天的股票价格
        Min=gp[1];
        for (int i=2;i<=n;i++)          //从前往后计算
        {
            Min=min (gp[i],Min);
            lr[i]=max(lr[i-1],gp[i]-Min);
        }
        Max=gp[n];
        for (int i=n-1;i>=1;i--)        //从后往前计算
        {
            Max=max(gp[i],Max);
            lr2[i]=max(lr2[i+1],Max-gp[i]);
        }
        ans=0;
        for (int i=1;i<=n;i++)          //计算最大利润
        {
            ans=max(ans,lr[i]+lr2[i]);
        }
        printf("%d\n",ans);
    }
    return 0;
}
```

分析

4
18 9 5 2



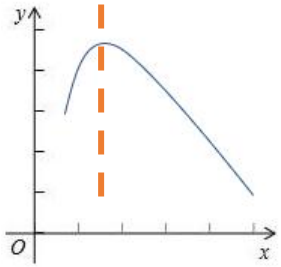
第三组样例：已知4天的价格，但由于价格一直在下跌，即后一天都要比前一天的价格要低

为了不亏损，当天购买的股票不能保留到后面几天

最后相当于没有在进行股票交易，所以最大利润为0

分析

6
6　8 7 4 1 -2



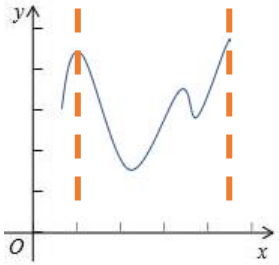
第二组样例：已知6天的价格，并且从第二天股票价格达到最高值，但是同样从第二天开始，呈现 下跌趋势

所以第一天买入股票，并在后面最高价格的时候(第二天) 卖出股票
8-6=2
为了不亏损，后续当天购买的股票同样也不能保留到后面几天

最后相当于只进行了一次有盈利的股票交易，所以最大利润为2

分析

7
5　14 -2 4 9 3 17



第一组样例：已知7天的价格，但是价格上有出现了3个高峰

选择第一天买入股票，并在后面第一个小高峰 (第二天) 卖出股票
第三天最低价格时再买入股票，第七天第三个高峰时卖出股票
即 (14-5) + (17- (-2)) =28

进行了两次有盈利的股票交易，最大利润为28

问题：当出现多个价格高峰时，如何选择？

[1120] trs滑雪

trs喜欢滑雪。他来到了一个滑雪场，这个滑雪场是一个矩形，为了简便，我们用r行c列的矩阵来表示每块地形。为了得到更快的速度，滑行的路线必须向下倾斜。例如样例中的那个矩形，可以从某个点滑向上下左右四个相邻的点之一。例如24-17-16-1，其实25-24-23…3-2-1更长，事实上这是最长的一条。

输入

输入文件 第1行: 两个数字r，c(1< =r,c< =100)，表示矩阵的行列。第2..r+1行:每行c个数，表示这个矩阵。

输出

输出文件 仅一行: 输出1个整数，表示可以滑行的最大长度。

样例输入

5 5

1 2 3 4 5

16 17 18 19 6

15 24 25 20 7

14 23 22 21 8

13 12 11 10 9

样例输出

25

什么是记忆化搜索呢？搜索的低效在于没有能够很好地处理重叠子问题；动态规划虽然比较好地处理了重叠子问题，但是在有些拓扑关系比较复杂的题目面前，又显得无奈。记忆化搜索正是在这样的情况下产生的，它采用搜索的形式和动态规划中递推的思想将这两种方法有机地综合在一起，扬长避短，简单实用，在信息学中有着重要的作用。

用一个公式简单地说：记忆化搜索=搜索的形式+动态规划的思想。

事实上，记忆化搜索与动态规划只是一种优化性能的手段，的确如你所说的那样是 一个是自顶向下，另一个是自底向上。
递推和记忆化搜索并不能相互转化，例如：第一种情况：记忆化搜索可以实现然而普通递推不能实现的问题（或实现起来非常麻烦的问题）最为典型的代表是“滑雪” 。

我们假设f[i][j]表示滑到坐标(i,j)所能滑到的最长长度。那么对于状态f[i][j]而言，它可以由f[i-1][j],f[i][j-1],f[i+1][j],f[i][j+1]四个状态推得，然而我们使用普通的递推（两个for）只能得到上、左两个方向的状态，右、下两个方向的状态却无从得知，因此使用递推就不能满足我们的要求，如果再补上两个for覆盖右、下状态，那么时间复杂度就变为了N^4，很明显会TLE。

这题如果采用记忆化搜索，在搜索的过程中发现f[i][j]在以前的某个时刻已经被计算过，直接return，可以保证时间复杂度为N^2。

思路：给出一个二维数组，让你求出最长递减序列长度，可以四个方向行走，起点任意。要对于每个点，都算出到达1的最长路径。
典型的动态规划题目，采用记忆化搜索，利用一个数组保存每个点的最大值，每次第一次访问一个点，就记录它到达1的最长路径，当下次访问时，就直接返回记录的值(动态规划的优点，避免重复计算子问题)，对每个点进行上下左右的求解，该点的最大值肯定是从四个方向中最大的+1。即用一个数组dp[110][110]存储每个节点的最长路径，先初始化为0，每当访问一个节点时，就判断dp值是否大于0，大于0则此点的值就是dp的值，直接返回。按照这个思想，就可求解。

```cpp
#include <bits/stdc++.h>
using namespace std;
#define MAXN  110
int a[MAXN][MAXN];
int dp[MAXN][MAXN];  //用来记录以点（i,j）为最大时的路径长度
int r, c;
int xx[4] = {1, -1, 0, 0};  //规定方向为下上右左
int yy[4] = {0, 0, 1, -1};
int dfs(int x,int y){
    int i,nx,ny;
    if(dp[x][y]!=0)   //该点还未记录过
        return dp[x][y];
    int ret=0;
    for(i=0;i<4;i++){ //进行搜索，寻找比该点小的点
        nx=x+xx[i];
        ny=y+yy[i];
        if(nx>=0&&nx<r&&ny>=0&&ny<c&&a[nx][ny]<a[x][y]){
            ret=max(ret,dfs(nx,ny));
        }
    }
    return dp[x][y]=ret+1;//记录该点开始的路径长度
}
```

```
23 int main(){
24     int ans=0,i,j;
25     scanf("%d%d",&r,&c);
26     for(i=0;i<r;i++)
27         for(j=0;j<c;j++)
28             scanf("%d",&a[i][j]);
29     memset(dp,0,sizeof(dp));//清空数组dp，表示所有点都未记录过
30     for(i=0;i<r;i++)
31         for(j=0;j<c;j++)
32             if(dp[i][j]==0)
33                 ans=max(ans,dfs(i,j));//记录下最长路径
34     printf("%d",ans);
35     return 0;
36 }
```

今天的课程结束啦……

下课了…
同学们再见！