# 第一章

## 神族文字

【题目描述】神族文字 (dictionary.cpp/c/pas) POJ 2503

楚继光发现图书馆里收藏有许多上古时代的魔法书，这些上古时代的魔法书使用一种传说中的“神族文字”来书写，幸运的是，楚继光手边恰巧有一本词典可以帮助他。

【输入格式】

输入的词典内容最多包含有100,000个词条，每一个词条包含一个英文单词，其次是一个空格和一个对应的“神族文字”。没有一个“神族文字”在词典中出现一次以上。词典词条全部输入完毕后是一个空行，之后是需要翻译的“神族文字”，每一个词一行，每个单词是一个最多为10个小写字母的字符串。

【输出格式】

输出翻译好的英文，每行一个字。若词典中查找不到，输出“eh”。

【输入样例】

dog ogday

cat atcay

pig igpay

froot ootfray

loops oopslay

atcay

ittenkay

oopslay

【输出样例】

cat

eh

loops

【算法分析】

由于数据量很大，需要用scanf 和 printf读写数据。标准库函数qsort+二分查找。即先对字典根据外语来进行qsort，然后通过二分查找即可找到匹配的项。

更进一步地，可以考虑哈希表。

### STL版

下面的参考程序使用stl模板，但训练时不建议使用stl模板。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51 | //神族文字－ＳＴＬ版  #include<iostream>  #include<string>  #include <cstdlib>  #include<map>  using namespace std;  int main(void)  {  freopen("dictionary.in","r",stdin);  freopen("dictionary.out","w",stdout);  char english[11],foreign[11];  map<string,string>translate;//记录foreign到english的映射  while(true)//输入字典  {  char t; //临时变量  if((t=getchar())=='\n') //判定是否输入了空行  break;  else //输入english  {  english[0]=t;  int i=1;  while(true)  {  t=getchar();  if(t==' ')  {  english[i]='\0';  break;  }  else  english[i++]=t;  }  }  cin>>foreign;  getchar(); //处理 输入foreign后的 回车符  translate[foreign]=english;  }  char word[11];  while(cin>>word)//开始翻译  {  if(translate[word]!="")  cout<<translate[word]<<endl;  else  cout<<"eh"<<endl;  }  return 0;  } |

### 非STL版

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66 | //神族文字－非STL版  #include <iostream>  #include <stdio.h>  #include <stdlib.h>  #include <math.h>  #include <string.h>  #include <algorithm>  using namespace std;  #pragma warning(disable:4996)  struct word  {  char get[11];  char have[11];  }dic[100001];  char str[25];  int n;  int cmp(const void \* x, const void \* y)  {  return strcmp((\*(word \*)x).have, (\*(word \*)y).have);  }  int judge()  {  int l = 1, r = n;  int ans;  int mid;  while (l <= r)  {  mid = (l + r) >> 1;  ans = strcmp(str, dic[mid].have);  if (ans == 0)  return mid;  else if (ans == 1)  l = mid + 1;  else  r = mid - 1;  }  return -1;  }  int main()  {  freopen("dictionary.in", "r", stdin);  freopen("dictionary.out", "w", stdout);  int a;  while (gets(str))  {  if (str[0] == '\0')  break;  n++;  sscanf(str, "%s%s", dic[n].get, dic[n].have);  }  qsort(dic + 1, n, sizeof(dic[0]), cmp);  int l;  while (scanf("%s", &str) != EOF)  {  a=judge();  if (a == -1)  printf("eh\n");  else  printf("%s\n", dic[a].get);  }  } |

## 近似整数

【题目描述】近似整数(Approximation.cpp/c/pas) POJ 1650

给定一个浮点数Ａ和一个整数L，求在范围[1，L]内的两个整数n和d，使得n/d能近似等于Ａ，且使误差|Ａ-n/d|最小。

【输入格式】

第一行为一个浮点数Ａ，第二行为一个整数Ｌ。

【输出格式】

两个整数n和d。

【输入样例】

3.14159265358979

10000

【输出样例】

355 113

【算法分析】

　　除了二分法外，还可尝试其他解法，例如：

解法一：不断比较得出min（A-n/d），在n/d>A时d++,相反则n++，即基本的追赶问题。

解法二：暴力枚举。

### 暴力枚举法

下面的程序采用暴力枚举，但训练时不建议使用暴力枚举。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | //近似整数－暴力枚举  #include<iostream>  #include<cmath>  #include<cstdlib>  using namespace std;  int main()  {  double min,a,temp;  int i,j,n,d,l;  while(scanf("%lf%d",&a,&l)!=EOF)  {  min=10.0;  for(i=1;i<=l;i++)  {  j=(int)(i/a);//找到最接近的j  if(j>l)  continue;  temp=fabs(a-(double)i/(double)j);//计算差值是多少  if(temp<min)//如果差值更小，更新一下n和j  {  min=temp;  n=i;  d=j;  }  j++;//因为之前取值，j值略小，所以j再加一，试一试会不会差值更小  temp=fabs(a-(double)i/(double)j);  if(temp<min)  {  min=temp;  n=i;  d=j;  }  }  printf("%d %d\n",n,d);  }  system("pause");  return 0;  } |

### 追赶法

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | //近似整数－追赶法  #include <iostream>  #include <stdlib.h>  #include <math.h>  #include <string.h>  #include <algorithm>  using namespace std;  int L;  double f;  int main()  {  freopen("Approximation.in", "r", stdin);  freopen("Approximation.out", "w", stdout);  scanf("%lf%d", &f, &L);  int ansn, ansd;  int n = 1, d = 1;  double min = 99999999, cha;  while(n <= L && d <= L)  {  cha = f - (double)n / d;  if (min > fabs(cha))  {  min = fabs(cha);  ansn = n;  ansd = d;  }  if (cha > 0)  n++;  else  d++;  }  printf("%d %d\n", ansn, ansd);  } |

### 二分法

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | //近似整数－二分法  #include<iostream>  #include<cstdio>  #include<cstring>  #include<cstdlib>  #include<cmath>  using namespace std;  int D,N,ll;  const double eps=1e-9;  double a,ans,k=1e7;  int main()  {  cin>>a>>ll;  for(int j=1; j<=ll; ++j)//i从1到l扫描  {  int l=1,r=ll,mid;  while(l<=r)  {  mid=(l+r)>>1;  ans=double(mid)/double(j);  if(fabs(ans-a)<k)  {  k=fabs(ans-a);  D=j,N=mid;  }  if(ans>a+eps)//此处进行二分  r=mid-1;//如值偏大，则分子范围选择左侧  else l=mid+1;//否则分子范围选择右侧  }  }  cout<<N<<' '<<D;  system("pause");  return 0;  } |

## 花费

【题目描述】花费（Expense.cpp/c/pas）POJ 3273

修罗王和邪狼的逃亡天数为N(1 ≤ N ≤ 100,000)，每天需要花的钱已经分配好，请把这些天分成M(1 ≤ M ≤ N)份（每份都是连续的天），则第i段的和为sum[i]（i=1,2...M），求max{sum[i]}最小为多少？

【输入格式】

【输出格式】

【输入样例】

7 5（表示N=7，M=5）

200 300 300 200 500 221 420 (表示每天的花费)

　　【输出样例】

500

【算法分析】

二分法，容易证明，分成组越少，花费越高，分成组越多，花费越少，呈线性关系，故可以采用二分法求解单调函数极值（最值）。一开始二分的上界为n天花费的总和（相当于分成1份），下界为每天花费的最大值（相当于分成n份），然后二分，每次的mid值为（上界 + 下界）/ 2，然后根据mid值遍历n天花费，对n天的花费进行累加，每当超过mid值 份数++，看看这个mid值能把n天分成几份，如果份数大于m，表示mid偏小，下界 = mid + 1，反之小于等于mid，上界 = mid - 1，然后输出最后的mid值即可，复杂度为 O(nlogM)。

除了二分法外，还可以尝试使用动态规划算法解决。

设best[i][j] 表示把j个单位分成i份使其中最大的一份最小的最优解的最大值，又假设第i份（最后１份）为第k+1个单位到第j个单位的和，显然前面的i-1份包括k个单位。

则best[i][j] = min{max(best[i - 1][k], sum(k + 1, j))}

其中k<j，sum(l, r)为l到r之间所有单位的和。

### 动规算法

下面的程序采用动态规划算法，但由于数据规模很大，动态规划算法显然会超时。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50 | //花费－动规算法  #include <iostream>  #include <stdio.h>  #include <string.h>  using namespace std;  int f[1000][1000];  int n,m;  int sum[10000],money[10000];  // f[i][j]=min(f[i-1][k],sum[j]-sum[k])  int main()  {  freopen("Expense.in","r",stdin);  freopen("Expense.out","w",stdout);  int i,j,k,minx,t,maxn;  while(~scanf("%d%d",&n,&m) && n)  {  memset(f,0,sizeof(f));  memset(sum,0,sizeof(sum));  memset(money,0,sizeof(money));  for(i=1;i<=n;i++)  {  scanf("%d",&money[i]);  f[1][i]=f[1][i-1]+money[i];  sum[i]=sum[i-1]+money[i];  }  for(i=2;i<=m;i++)  {  maxn=-1;  for(j=1;j<=i;j++)  maxn=max(money[j],maxn);  for(j=1;j<=i;j++)  f[i][j]=maxn;  for(j=i+1;j<=n;j++)  {  minx=9999999;  for(k=i-1;k<j;k++)  {  t=max(f[i-1][k],sum[j]-sum[k]);  minx=min(t,minx);  }  f[i][j]=minx;  }  }  printf("%d\n",f[m][n]);  }  return 0;  } |

### 二分法

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57 | //二分法  #include <iostream>  #include <stdio.h>  #include <stdlib.h>  #include <math.h>  #include <string.h>  #include <algorithm>  using namespace std;  int n, m;  int data[100001];  int high, low, mid;  bool judge()  {  int i;  int temp = 0;  int plan = 1;  for (i = 1; i <= n; i++)  {  temp += data[i];  if (temp > mid)  {  plan++;  temp = data[i];  }  }  if (plan > m)  return 0;  else  return 1;  }  int main()  {  freopen("Expense.in","r",stdin);  freopen("Expense.out","w",stdout);  scanf("%d%d", &n, &m);  int i;  for (i = 1; i <= n; i++)  {  scanf("%d", &data[i]);  high += data[i];  if (low < data[i])  low = data[i];  }  while (low <= high)//==  {  mid = (low + high) >> 1;  if (!judge())  low = mid + 1;  else  high = mid - 1;  }  printf("%d\n", low);  return 0;  } |

## 快速模幂

【题目描述】快速模幂（Modulo.cpp/c/pas）

试求ab%n（这就是著名的RSA公钥的加密方法）的值，其中a，b，n均为整数范围内的数。

【输入格式】

三个整数即a，b，n。

【输出格式】

输出结果。

【输入样例】

1 1 1

【输出样例】

0

【算法分析】

如果先计算ab再求模，两次运算都需要高精度运算，但运算结果却在整型数范围内。故通常是将幂模运算转换为乘模运算（蒙哥玛利模幂算法）。有以下两个公式：

（1）a×b%n=（a%n×b%n)%n

（2）（a+b）%n=（a%n+b%n)%n

算法1：利用公式a×b%c=((a%c) ×b)%c,这样每一步都进行这种处理，这就解决了ab可能太大存不下的问题，但这个算法的时间复杂度依然没有得到优化。

代码如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | //快速模幂  #include<iostream>  #include <cstdlib>  using namespace std;  int a,b,n;  int a\_b\_mod\_n(int a,int b,int n)  {  int result = 1;  while (b--)  {  result = a \* result % n;  }  return result;  }  int main()  {  cin>>a>>b>>n;  cout<<a\_b\_mod\_n(a,b,n);  system("pause");  return 0;  } |

算法2：使用快速幂二分优化，可以达到O(logn)。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | #include <iostream>  #include <cstdlib>  #include <cstdio>  using namespace std;    int a\_b\_mod\_n(int a,int b,int n)  {  int result=1;  int tmp=a;  while(b)  {  if(b & 1)  result=result\*tmp%n;  tmp=tmp\*tmp%n;  b>>=1;  }  return result;  }    int main()  {  freopen("Modulo.in","r",stdin);  freopen("Modulo.out","w",stdout);  int a,b,n;  cin>>a>>b>>n;  cout<<a\_b\_mod\_n(a,b,n)<<endl;  return 0;  } |

例如求C15 %n，可分解为6 个乘模运算：

（1）C1 =C×C % n =C2 % n

（2）C2 =C1×C % n=C3 % n

（3）C3 =C2×C2 % n =C6 % n

（4）C4 =C3×C % n =C7 % n

（5）C5=C4×C4 % n =C14 % n

（6）C6 =C5×C % n =C15 % n

故可以用递归或递推方法来实现。

### 递推算法

递推算法的参考代码如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | //快速模幂  #include<iostream>  using namespace std;  int a,b,n;  int a\_b\_mod\_n(int a,int b,int n)  {  int digit[32]={};  int i,k,result=1;  i=0;  //把b化成2进制,这是因为要知道b何时能整除2，并不需要反复进行  //减一或除二的操作，只需验证b的二进制各位是0 还是1 就可以了，  //从左至右或从右至左验证都可以，从左至右会更简洁  while(b)//例如b=103,二进制为1100111,则digit[]存为11100110000......  {  digit[i++]=b%2;  b>>=1;  }    for(k=i-1;k>=0;k--) //计算a^b mod n  {  result=(result\*result)%n;  if(digit[k]==1)  result=(result\*a)%n;  }  return result;  }  int main()  {  cin>>a>>b>>n;  cout<<a\_b\_mod\_n(a,b,n);  system("pause");  return 0;  } |

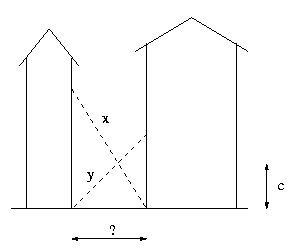
### 递归算法

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | //快速模幂 －递归  #include <iostream>  #include <stdio.h>  #include <stdlib.h>  #include <math.h>  #include <string.h>  #include <algorithm>  using namespace std;  int a, b, c;  long long mi(long long a, int b)  {  if (b == 1)  return a % c;  long long t = mi(a, b / 2);  return (b & 1) ? (t\*t\*a) % c : (t\*t) % c;  }  int main()  {  freopen("Modulo.in", "r", stdin);  freopen("Modulo.out","w",stdout);  scanf("%d%d%d", &a, &b, &c);  int ans = mi(a, b);  printf("%d\n", ans%c);  return 0;  } |

## 交叉的梯子

【问题描述】交叉的梯子(ladders.cpp/c/pas) PKU 2507

如图所示，魔法学院有一个狭窄的街道上矗立着两栋楼，右楼有一个长为x的梯子搭在左楼，左楼有一个长为y的梯子搭在右楼，两梯子的交叉点的高度在c处。问这个街道的宽度。



图

【输入格式】

每行一组数据，表示x，y，c。

【输出格式】

输出街道的宽度。

【输入样例】

30 40 10

12.619429 8.163332 3

10 10 3

10 10 1

【输出样例】

26.033

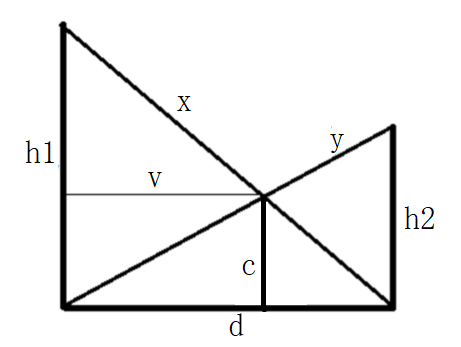
7.000

8.000

9.798

【算法分析】

计算几何题+二分。试做图如下：



可得：h1=sqrt(x2-d2)

h2=sqrt(y2-d2)

由三角形相似原理，得：

= ，

得=

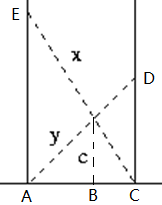
得c=(h1\*h2)/(h1+h2)

则二分求c的值比较即可。

参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | //½»²æµÄÌÝ×Ó  #include <bits/stdc++.h>  using namespace std;  double x,y,c;  bool judge(double t)  {  double h1=sqrt(x\*x-t\*t);  double h2=sqrt(y\*y-t\*t);  return h1\*c+h2\*c-h1\*h2>0? true:false;  }  int main()  {  freopen("ladders.in","r",stdin);  freopen("ladders.out","w",stdout);  while(~scanf("%lf%lf%lf",&x,&y,&c))  {  double low=0,mid; //ÏÂ½çÎª0  double high=(x>y?y:x); //ÉÏ½çÎªx,yÖÐµÄ½ÏÐ¡Õß  while(low<=high)  {  mid=(low+high)/2;  judge(mid)? high=mid-0.0001:low=mid+0.0001;  }  printf("%.3lf\n",low);  }  return 0;  } |

算法2：试作图如下：



图

根据相似性原理：易知，又有

故有

即

通分得：

令f(AC)= 

则下界是0，上界是x，y中的较小值，二分枚举ＡＣ，令f(AC)=0即为答案。

## 老板的又一道题

【问题描述】老板的又一道题（k4.cpp/c/pas）

有两个长度都为n的正整数序列A和B，从A和B中各取其中的一个数相加一共可以得到n2个和。要求输出这n2个和中最小的n个。

【输入格式】

第一行，一个正整数n。

第二行，n个用空格隔开的正整数，代表A序列。

第三行，n个用空格隔开的正整数，代表B序列。

【输出格式】

一行，依次是从小到大输出n个最小的和，每两个数之间用一个空格隔开。

【输入样例】

3

2 6 6

1 4 8

【输出样例】

3 6 7

【数据范围】

50%的数据：n<=500，0<Ai，Bi<=1000000000;

100%的数据：n<=100000。

类似于第K小数3（k3），参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67 | //老板的又一道题  #include <fstream>  std::fstream fin("k4.in", std::ios::in);  std::fstream fout("k4.out", std::ios::out);  int A[100000], B[100000], C[1000000], Cptr, n;  int dec(const void\* a, const void\* b)  {  return \*static\_cast<const int\*>(a) - \*static\_cast<const int\*>(b);  }  bool enough(int limit)//k3的原理,O(N)  {  int Aptr, Bptr, sum = 0;  for (Aptr = 0, Bptr = n - 1; Aptr < n; ++Aptr)  {  while (Bptr >=0 && A[Aptr] + B[Bptr] > limit)  --Bptr;  sum += Bptr + 1;  }  return sum >= n;  }  //找到第n大的数后，把所有比它小的数加入新数组，参见k3，O(N)  void push(int limit)  {  int Aptr, Bptr, i, sum = 0, ptr1, ptr2;  for (Aptr = 0, Cptr = 0; Aptr < n; ++Aptr)  for (Bptr=0;A[Aptr]+B[Bptr]<=limit && Bptr<n;++Bptr)  C[Cptr++] = A[Aptr] + B[Bptr];  for (ptr1 = 0, ptr2 = 0; ptr1 < Cptr; ++ptr1)  if (C[ptr1] != limit) C[ptr2++] = C[ptr1];  while (ptr2 < n)  C[ptr2++] = limit;  }  //参见k3, O(logN)  int find(int left, int right, bool (\*law)(int arg))  {  int mid;  while (left < right)  {  mid = int((unsigned(left)+unsigned(right))/2);  if (law(mid))  right = mid;  else  left = mid + 1;  }  return left;  }  int main()  {  fin >> n;  for (int i = 0; i < n; ++i)  fin >> A[i];  for (int i = 0; i < n; ++i)  fin >> B[i];  qsort(A, n, sizeof(A[0]), dec);//排序O(NlogN)  qsort(B, n, sizeof(B[0]), dec);//排序O(NlogN)  push(find(A[0]+B[0],A[n-1]+B[n-1],enough));  qsort(C, Cptr, sizeof(C[0]), dec);//最后排序输出  for (int i = 0; i < n; ++i)  fout << C[i] <<' ';  fout << std::endl;  } |

## 电脑组装

【问题描述】电脑组装(Assemble.cpp/c/pas)　POJ 3497

魔法世界的魔法师们近来讨论话题最多的无疑是最新型量子计算机的上市，量子计算机运算速度惊人，例如求解一个亿亿亿级变量的方程组，即便是用世界上最快的超级计算机也至少需要几百年。而量子计算机十秒钟就可解决。所以对于一直信奉“工欲善其事,必先利其器。”的张琪曼来说，购买一台最新型量子计算机是她的近期目标。但是由于量子计算机价格昂贵，她只能用一定的预算去买各种量子计算机的组件，组件每种买一个，其中电脑组件都有品质和价格两个参数。

求在不超过预算的情况下，能买到的所有组件的最差品质的最大值是多少。

【输入格式】

第一行为一个整数Ｎ，表示测试组数，Ｎ不超过100。

每组数据第一行有两个数，即组件数和预算，其中1≤组件数≤1 000, 1≤预算≤1 000 000 000。

以下各行为组件的类型、名称、价格、质量。

【输出格式】

能买到的所有组件的最差品质的最大值。每组测试数据为一行。

【输入样例】

1　　（表示测试组数，不超过100组）

18 800 (表示1≤组件数≤1 000, 1≤预算≤1 000 000 000)

processor 3500\_MHz 66 5 （类型，名称，价格，质量）

processor 4200\_MHz 103 7

processor 5000\_MHz 156 9

processor 6000\_MHz 219 12

memory 1\_GB 35 3

memory 2\_GB 88 6

memory 4\_GB 170 12

mainbord all\_onboard 52 10

harddisk 250\_GB 54 10

harddisk 500\_FB 99 12

casing midi 36 10

monitor 17\_inch 157 5

monitor 19\_inch 175 7

monitor 20\_inch 210 9

monitor 22\_inch 293 12

mouse cordless\_optical 18 12

mouse microsoft 30 9

keyboard office 4 10

【输出样例】

９

【算法分析】

一道排序+二分题，时间卡的比较紧所以输入不要一个字符一个字符处理，排序的时候尽量避免字符串比较，可以用STL中的map<string，int>把“种类”的字符串映射成整数或者把组件直接归类均可。

对每一类组件按照品质从小到大排序，然后二分答案quality，每一种组件中选一个品质不低于quality的最便宜的，跟预算比较即可。

参考程序略。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84 | //电脑组装  #include <iostream>  #include <cstdio>  #include <map>  #include <string>  #include <cstring>  #include <algorithm>  using namespace std;  struct compute  {  int type,price,quality;  }d[1005];  int init[1005],n,w,Count;  map<string,int> MAP;//将string映射为数字  int cmp(compute a,compute b)  {  return a.quality<b.quality;  }  int weight(int middle)  {  int i,j,s;  memset(init,-1,sizeof(init));  for(i=middle;i<n;i++)  {  if(init[d[i].type]==-1)//如果该类型还没有统计  init[d[i].type]=d[i].price;//则添加该类型的一个组件的价值  else if(init[d[i].type]>d[i].price)  init[d[i].type]=d[i].price;//选取部件选给定quality的最便宜的部件  }  for(s=i=0;i<Count;i++)//看是否所有类型都选上  if(init[i]==-1)//若有没选的，则失败  return 0;  else  s+=init[i];//否则累加  if(s>w) //若超过预算，失败  return 0;  else //否则成功  return 1;  }  int binary()//二分  {  int left=0,right=n-1,middle;  while(left<right)  {  middle=(left+right+1)/2;//要加1，否则遇到8,9就成死循环  if(weight(middle))  left=middle;  else  right=middle-1;  }  return d[left].quality;  }  int main()  {  freopen("Assemble.in","r",stdin);  freopen("Assemble.out","w",stdout);  int t,i,j;  string s; char a[25];  cin>>t;  while(t--)  {  Count=0;  scanf("%d%d",&n,&w);//组件数，预算  for(i=0;i<n;i++)  {  scanf("%s",a);//类型  s=a;  scanf("%s%d%d",a,&d[i].price,&d[i].quality);  if(MAP.find(s)==MAP.end())//如果该类型还没有  MAP[s]=Count++;//该类别映射为某个数字  d[i].type=MAP[s];//标记归类  }  MAP.clear();//映射任务完成  sort(d,d+n,cmp);//按质量排序  printf("%d\n",binary());  }  return 0;  } |

# 第二章　递归算法

## 棋子移动２

【题目描述】棋子移动２（piece2.cpp/c/pas）

现在，修罗王和邪狼已经陷入了魔法棋阵中，魔法棋阵有2N个棋子(N>=4)排成一行，开始位置为白子全部在左边，黑子全部在右边，例如当N=4时，棋子排列情况为〇〇〇〇●●●●。允许每一步将相邻两个棋子交换位置，最后形成黑白相间的排列。只要求出移动步骤即可将修罗王和邪狼抓捕归案。

【输入格式】

一个整数Ｎ。

【输出格式】

输出移动步骤。

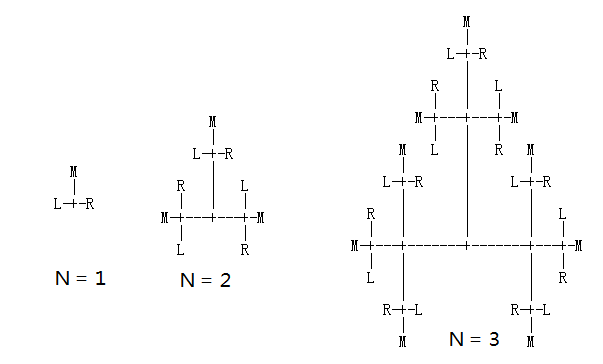
【算法分析】

非常简单的移动策略，自制测试数据验证即可。

## 分形图３

【题目描述】分形图３（Fractal3.cpp/c/pas）

图是一种简单的用字符打印出的分形图的几个例子。



图

可以看出，对于给定的Ｎ，图中有3N个字母，Ｌ，Ｍ，Ｒ各有３Ｎ-1个，现要求把数字１，２，３，...３Ｎ代表字母，并要求代替Ｌ的字母的数字和、代替Ｍ的字母的数字和及代替Ｌ的字母的数字和相等，你能做到吗？

如果有方案，把代替Ｌ的字母的数字输出到文件第一行，把代替Ｍ的字母的数字输出到文件第二行，把代替Ｒ的字母的数字输出到文件第三行。

【输入格式】

输入Ｎ的值，Ｎ>=2。

【输出格式】

输出三行答案，每个数字之间以一个空格间隔。

【输入样例】

２

【输出样例】

1 6 8

5 7 3

9 2 4

首先，生成图中分形图的程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60 | //生成分形图  #include <iostream>  #include <cstdlib>  using namespace std;  const int d[4][2]={{-1,0},{0,1},{1,0},{0,-1}};//4个方向变化量  const char ch[4]={'|','-','|','-'};//４个方向线型  const char cha[4]={'M','R','x','L'};//４种结点打印符号  int n;  char g[1500][1500];//保存图形数组  int len[10];//各分支大小  void draw(int x,int y,int dir1,int dir,int level)//x,y为中心坐标，  {//dir为子树“正”方向，dir1为结点字符编号，level为子树“层数”  int i,j,x1,y1;  if(level==0)  {  g[x][y]=cha[dir1];  return;  }  g[x][y]='+';  for(i=-1;i<2;i++)//分别递归处理３个方向  {  x1=x;  y1=y;  for(j=0;j<=len[level];j++)  {  x1+=d[(i+dir+4)%4][0];  y1+=d[(i+dir+4)%4][1];  if(j==len[level])  break;  g[x1][y1]=ch[(i+dir+4)%4];  }  draw(x1,y1,(i+4)%4,(i+dir+4)%4,level-1);  }  }  void out(int s,int m)  {  for(int i=s;i<m;i++)  {  for(int j=s;j<m;j++)  if(g[i][j])  cout<<g[i][j];  else cout<<" ";  cout<<endl;  }  }  int main()  {  cin>>n;  len[1]=1;  for(int i=2;i<=n;i++)  len[i]=2\*len[i-1]+1;  draw(len[n]\*2+1,len[n]\*2+1,0,0,n);  out(0,len[n]\*4+2);  system("pause");  return 0;  } |

我们会发现，把数学１，２，３，...,3N分成３行,使各行各列的数字和相等。如Ｎ＝２时，一种答案为：

1 6 8

5 7 3

9 2 4

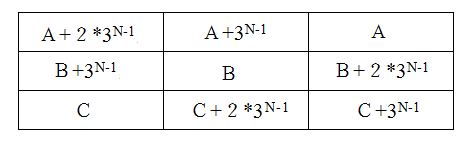
同样类似分形思想，对于Ｎ>２的图形，可以这样分析：

如果Ｎ－１的方案已知，为Ａ，Ｂ，Ｃ三行，

第一步：把每个数加3N-1，得到新的三行，记为Ａ+3N-1，Ｂ+3N-1，Ｃ+3N-1，；

第二步：把每个数加２\*3N-1，得到另外三行，为Ａ+２\*3N-1，Ｂ+２\*3N-1，Ｃ+２\*3N-1。

第三步：调整如下图所示：



图

此时，每行和相等，每行列相等，并且每块（“叶子”或“分支”）中的数字和都相等。

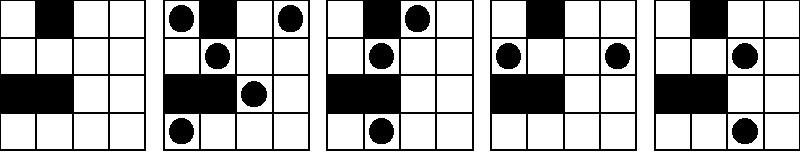
本题的参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51 | //分形图３  #include <iostream>  #include <cstdlib>  using namespace std;  const int maxn=1000;  int A[maxn],B[maxn],C[maxn],n;  void move(int D[],int len,int p1,int p2)  {  for(int i=0;i<len;i++)  swap(D[i+p1],D[i+p2]);  }  void out(int D[],int len)  {  for(int i=0;i<len;++i)  cout<<D[i]<<" ";  cout<<endl;  }  int main()  {  A[0]=1,A[1]=6,A[2]=8;  B[0]=5,B[1]=7,B[2]=3;  C[0]=9,C[1]=2,C[2]=4;  cin>>n;  int len=3,len2=3\*3;  for(int i=3;i<=n;i++)  {  for(int j=1;j<3;j++)  {  for(int k=0;k<len;k++)  {  A[k+len\*j]=A[k]+len2\*j;  B[k+len\*j]=B[k]+len2\*j;  C[k+len\*j]=C[k]+len2\*j;  }  }  move(A,len,0,len\*2);  move(B,len,0,len);  move(C,len,len,len\*2);  len=len2;  len2=3\*len2;  }  out(A,len);  out(B,len);  out(C,len);  system("pause");  return 0;  } |

## 冲突

【题目描述】冲突(Conflict.cpp/c/pas) POJ 1315

监狱的每间牢房是一个不超过4×4的正方形，里面设有一些障碍，牢房里住着的犯人脾气都很大，只要两个犯人位于同一行或同一列即会发生冲突，但障碍物可以阻挡同行同列犯人的冲突。问最多可放几个犯人而不会发生冲突。如图所示，左边表示初始牢房样，右边四个显示了摆放方案，当然，最后两个方案是错误的。



图

【输入格式】

有多组测试数据，每组数据第一行为一个整数Ｎ表示牢房大小。随后Ｎ行描述牢房，其中X表示障碍。

所有测试数据结束的标志为０。

【输出格式】

输出最多可放的犯人数。

【输入样例】

4　　（表示牢房大小）

.X..　（描述牢房，Ｘ表示障碍）

....

XX..

....

3

.X.

X.X

.X.

3

...

.XX

.XX

0　　（表示结束）

【输出样例】

5

5

2

【算法分析】

其实数据规模暗示了算法。

方法一：递归算法。

方法二：非递归算法－DFS。

方法三：二分图（需掌握二分图的相关知识，请查阅相关资料）。

此处仅提供DFS算法以做参考，请尝试完成递归算法：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  54  65 | //冲突 －ＤＦＳ  #include <iostream>  #include <cstdlib>  #include <cstring>  using namespace std;  #define maxn 10  int n,ans;  char map[maxn][maxn];  bool rook[maxn][maxn];  int dir[4][2] ={{ 0, 1 },{ 1, 0 },{ 0, -1 },{ -1, 0 } };  bool in(int x, int y)  {  return x < n && y < n && x >= 0 && y >= 0;  }  bool ok(int sx, int sy)  {  if (map[sx][sy] == 'X')  return false;  for (int i = 0; i < 4; i++)  {  int x = sx;  int y = sy;  while (in(x, y) && map[x][y] == '.')  {  if (rook[x][y])  return false;  x += dir[i][0];  y += dir[i][1];  }  }  return true;  }  void dfs(int a, int cnt)  {  ans = max(ans, cnt);  for (int i = a; i < n \* n; i++)  {  int x = i / n;  int y = i % n;  if (ok(x, y))  {  rook[x][y] = true;  dfs(i + 1, cnt + 1);  rook[x][y] = false;  }  }  }  int main()  {  while (scanf("%d", &n), n)  {  for (int i = 0; i < n; i++)  scanf("%s", map[i]);  memset(rook, 0, sizeof(rook));  ans = 0;  dfs(0, 0);  printf("%d\n", ans);  }  return 0;  } |

## 放苹果

【题目描述】放苹果（apple.cpp/c/pas）POJ 1664

把M个同样的苹果放在N个同样的盘子里，允许有的盘子空着不放，问共有多少种不同的分法？（用K表示）5，1，1和1，5，1 是同一种分法。

【输入格式】

第一行为一个整数，表示测试数据的数目，第二行为Ｍ和Ｎ。

【输出格式】

输出有多少种不同分法。

【输入样例】

1　　（测试数据的数目t（0 <= t <= 20））

7 3　　(M和N，以空格分开。1<=M，N<=10)

【输出样例】

8

【算法分析】

　　令f(m,n)表示m个苹果放到n个盘子里有多少种放法，下面对不同的情况给予讨论：

　　(1)当盘子数为1的时候，只有一种放法就是把所有苹果放到一个盘子里。

　　(2)当苹果数为1的时候，也只有一种放法，注意题目中说明，盘子之间并无顺序,所以不管这个苹果放在哪个盘子里，结果都算一个。

　　(3)当m<n时，因为此时最多只能放到m个盘子中去（一个盘里放一个），实际上就相当于把m个苹果放到m个盘子里一样，也就是f(m，m)。

　　(4)当m>=n时，也分两种情况讨论，一种是至少有一个盘子里不放苹果，这样子就相当于f(m，n-1),第二种是，先取出n个苹果一个盘子里放一个，再将剩下的m-n个苹果放到n个盘子里去，即f(m-n，n)。

　　则递归表达式：

　　f(m，n)=1 当 m=1或n=1

　　f(m，n)=f(m，m) 当m<n

　　f(m，n)=f(m-n，n)+f(m，n-1)

参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | //放苹果  #include <iostream>  #include <cstdlib>  using namespace std;  int f( int m, int n)  {  if( m == 1 || n ==1 || m== 0)  return 1;  else if( m < n )  return f( m, m);  else  return f( m - n, n) + f( m, n - 1);  }  int main()  {  freopen("apple.in","r",stdin);  freopen("apple.ans","w",stdout);  int t, m, n;  cin>>t;  while( t-- )  {  cin>>m>>n;  cout<<f( m, n)<<endl;  }  return 0;  } |

也可以抽象为数学问题。

# 第三章　排列组合问题

## 巡视

【题目描述】巡视（Patrol.cpp/c/pas）POJ 2907

　　典狱长每天要到监狱的n个地方巡视一般，监狱可以看作是一个row×col的矩阵（均不超过20），典狱长起点的位置(x，y)，还有其他n个目标点，问从起点出发，走过每个目标点之后返回到起点，典狱长只能沿x，y轴移动，不能走对角线，请问最短的路径是多少？

【输入格式】

第一行为一个整数，表示测试数据的组数。以后每组数据的第一行为两个整数，表示矩阵大小，第二行为起始位置坐标，第三行为一个整数即目标点数n，随后n行为各点坐标。

【输出格式】

输出最短路径，每组测试数据一行。

【输入样例】

　　1（表示测试数据组数）

　　10 10（表示矩阵大小）

　　1 1（起始位置）

　　4　　（目标点数）

　　2 3　　（各目标点坐标）

　　5 5

　　9 4

　　6 5

【输出样例】

The shortest path has length 24

【算法分析】

由于点不多，可以直接全排列+dfs解决。参考代码如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69 | //巡视  #include <iostream>  #include <stdlib.h>  #include <math.h>  using namespace std;  #define INF 0x3fffffff  struct node  {  int x,y;  }g[11],save[11];  int n,m,k;  int sx,sy;  int mark[11];  int mi;  void cmp()  {  int sum=0;  sum=fabs(save[0].x-sx)+fabs(save[0].y-sy);  int tx,ty;  tx=save[0].x;  ty=save[0].y;  for(int i=1;i<k;i++)  {  sum+=fabs(save[i].x-tx)+fabs(save[i].y-ty);  tx=save[i].x;  ty=save[i].y;  }  sum+=(fabs(tx-sx)+fabs(ty-sy));  if(sum<mi)  mi=sum;  }  void dfs(int s)  {  if(s==k)  cmp();  for(int i=1;i<=k;i++)  {  if(mark[i]==0)  {  mark[i]=1;  save[s].x=g[i].x;  save[s].y=g[i].y;  dfs(s+1);  mark[i]=0;  }  }  }  int main()  {  int t;  scanf("%d",&t);  while(t--)  {  mi=INF;  scanf("%d%d",&n,&m);  scanf("%d%d",&sx,&sy);  scanf("%d",&k);  for(int i=1;i<=k;i++)  scanf("%d%d",&g[i].x,&g[i].y);  dfs(0);  printf("The shortest path has length %d\n",mi);  }  return 0;  } |

## 因子数

　　【题目描述】因子数（FactorNumber.cpp/c/pas）POJ 2992

试计算C(n，k)的因子个数。

【输入格式】

有多组数据，每行两数即n，k，其中0 ≤ k ≤ n ≤ 431)

【输出格式】

【输入样例】

5 1

6 3

10 4

【输出样例】

2

6

16

【算法分析】

一个数学问题。由于k，n比较大，所以不可能直接求组合数。

n的因子个数求法：由n=p1a×p2b......pnc; 即质因数乘积的形式，其中p1，p2……pn为质数，那么n的因子的个数就是sum=(a+1)×(b+1)×...×(c+1)。

参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59 | //因子数  #include<stdio.h>  #include<string.h>  int p[90];  int v[432];  int pn;  int e[432][90];  void pri()  {  int i,j;  pn=0;  memset(v,0,sizeof(v));  for(i=2;i<=431;i++)  {  if(v[i]==0)  {  p[pn++]=i;  for(j=i;j<=431;j+=i)  v[j]=1;  }  }  }  void fac()  {  int i,j,k;  for(i=2;i<=431;i++)  {  k=i;  for(j=0;j<pn;j++)  while(k>1&&k%p[j]==0)  {  e[i][j]++;  k/=p[j];  }  }  for(i=3;i<=431;i++)  for(j=0;j<pn;j++)  e[i][j]+=e[i-1][j];  }  int main()  {  freopen("FactorNumber.in","r",stdin);  freopen("FactorNumber.out","w",stdout);  int i,n,k;  long long sum;  pri();  fac();  while(scanf("%d%d",&n,&k)!=EOF)  {  sum=1;  for(i=0;i<pn;i++)  sum\*=(1+e[n][i]-e[n-k][i]-e[k][i]);  printf("%lld\n",sum);  }  return 0;  } |

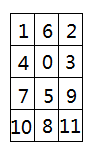
# 第五章　排序算法

## 绝境求生

【题目描述】绝境求生（mnPuzzle.cpp/c/pas）POJ 2893

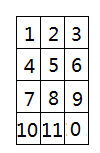
“天道酬勤”是针对每一个人的，而不在于他是“好人”还是“坏人”。换句话说，如果“好人”不够努力，不够勤奋，他其实是很难战胜勤奋而“勇敢”的“坏人”的。

所以当修罗王和邪狼陷入了绝境时，他们仍在拼命地寻找一线生机。所谓绝境是由一个M × N的矩形，其中M和N至少有一个数是奇数。矩形中有1到MN－1个可以滑动的方块，0代表空地，例如当M＝4和N＝3时，绝境可能如图所示。



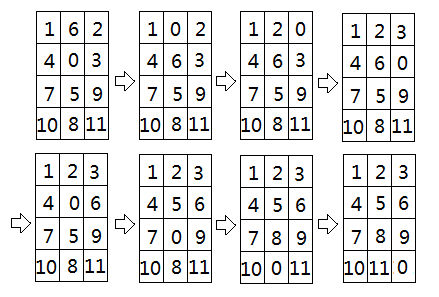
图

通过移动空地周围的方块，移成如图所示的状态才可逃脱。



图

例如M＝4和N＝3时的绝境移动顺序如图所示。



现在给一个M× N的矩形绝境，试计算能否逃脱。

【输入格式】

包含多组测试数据，每组数据第一行为两个整数M和N (2 ≤ M, N ≤ 999)，随后M行为各组数据。全部测试数据结束后，以0 0结尾。

【输出格式】

每组测试数据给出答案，即是否能逃脱。能则“YES”，否则“NO”。

【输入样例】

3 3

1 0 3

4 2 5

7 8 6

4 3

1 2 5

4 6 9

11 8 10

3 7 0

0 0

【输出样例】

YES

　　NO

【算法分析】

　　本题实质是逆序对（不包括０）的奇偶性判断，有关详细的逆序对的奇偶性判断，请参见本书搜索算法中八数码问题的ＩＤＡ\*算法讲解部分。此题的目标状态为偶数。

　　首先，0的左右移动不改变奇偶性。

　　当N为奇数时，上下移动不改变奇偶性，故逆序数为偶的为YES；

　　当N为偶数时，上下移动逆序数变化为±1，此时还要考虑0的竖直距离，逆序数%2 == 距离%2 时为YES。

参考程序如下所示：

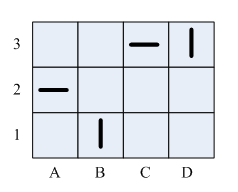
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55 | //绝境求生  #include <stdio.h>  #include <string.h>  #define lowbit(i) (i) & (-i)  #define N 1000007  int a[N],b[N],k;  void update(int i)  {  for(; i <=k; i += lowbit(i))  b[i] ++;  }  int sum(int i)  {  int ans = 0;  for(; i > 0; i -= lowbit(i))  ans += b[i];  return ans;  }  int cal()  {  int i,res = 0;  for(i = 0; i < k; ++i)  {  update(a[i]);  res += i + 1 - sum(a[i]);  }  return res;  }  int main()  {  int n,m,cnt,step,i,j,s;  while(scanf("%d%d",&m,&n) && m)  {  memset(b,0,sizeof(b));  k = 0;  for(i = 0; i < m; ++i)  for(j = 0; j < n; ++j)  {  scanf("%d",&s);  if(s == 0)  step = m - i - 1;  else  a[k++] = s;  }  cnt = cal();  if(n & 1)  step = 0;  printf(step % 2 == cnt % 2? "YES\n" : "NO\n");  }  return 0;  } |

# 第六章　穷举算法

## 纵横火柴棒

【题目描述】纵横火柴棒（matches2.cpp/c/pas）2012蓝桥杯软件大赛决赛

这是一个纵横火柴棒游戏。在3×4的格子中，游戏的双方轮流放置火柴棒。



图

其规则是：

1.不能放置在已经放置火柴棒的地方（即只能在空格中放置）。

2.火柴棒的方向只能是竖直或水平放置。

3.火柴棒不能与其它格子中的火柴“连通”。所谓连通是指两根火柴棒可以连成一条直线，且中间没有其它不同方向的火柴“阻拦”。

例如：上图所示的局面下，可以在C2位置竖直放置（为了方便描述格子位置，图中左、下都添加了标记），但不能水平放置，因为会与A2连通。同样道理，B2，B3，D2此时两种方向都不可以放置。但如果C2竖直放置后，D2就可以水平放置了，因为不再会与A2连通（受到了C2的阻挡）。

4.游戏双方轮流放置火柴，不可以弃权，也不可以放多根。直到某一方无法继续放置，则该方为负（输的一方）。

游戏开始时可能已经放置了多根火柴。

你的任务是：编写程序，读入初始状态，计算出对自己最有利的放置方法并输出。

【输入格式】

第一行一个整数，表示有多数数据，以下各行描述初始状态。

【输出格式】

输出答案。

【输入样例】

2 (表示有两组数据)

0111（“0”表示空，“1”表示竖直放置，用“-”表示水平放置。）

-000

-000

1111(这是第二组数据)

----

0010

【输出样例】

00-　（对第一个局面，在第0行第0列水平放置,答案非唯一，输出一种即可）

211 （对第二个局面，在第2行第1列垂直放置）

【算法分析】

暴力递归出解的深度，即判断每一个可能的位置，直到找不到位置放置。此时若深度为0，说明一开始就不能放置，输掉比赛。若深度为偶数（>0），说明这样走也会输掉比赛，将初始位置走法记入到输的队列。若深度为奇数，则说明这样走会赢取比赛，将初始位置走法记入到赢的队列。

选取最佳位置，判断赢的队列的每一个初始位置是否存在于输的队列中，若不存在，该初始位置为最佳位置（可以找多个），若存在，该初始位置为次最佳位置（当没有最佳位置时，它就为最佳）。

参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152 | //纵横火柴棒算法2  #include<iostream>  using namespace std;  int map[4][5],i,j,k,x,y,z;  int num,mx,my,mk,m,Min=10000,Max=0,N,u,v=1;  char temp[4][5];//临时存输入的值  int win[4][5],lose[4][5];//赢队列，输队列，用数组仿真  int zui[4][5],ci[4][5];//存最优解，存次优解  int place(int x, int y,int t) //判断某一位置是否能否安置火柴棒，x,y坐标，t为横竖状态  {  int m,b=1;  if (map[x][y]!=0)  return 0;  else  if (t==2) //如果竖直状态  {  for (m=x;m<=3;m++)  {  if (map[m][y]==1)  break;  if (map[m][y]==2)  b=0;  }  for (m=x;m>=1;m--)  {  if (map[m][y]==1)  break;  if (map[m][y]==2)  b=0;  }  }  if (t==1) //如果水平状态  {  for (m=y;m<=4;m++)  {  if (map[x][m]==2)  break;  if (map[x][m]==1)  b=0;  }  for (m=y;m>=1;m--)  {  if (map[x][m]==2)  break;  if (map[x][m]==1)  b=0;  }  }  return b;  }  void Try(int x,int y,int k,int d)//求出解的深度 ，x,y,k为初始状态，d为深度  {  int a,b,c,e=0,temp; //temp恢复map数组  if (d>Max)  Max=d;  for (b=1;b<=4;b++)  for (a=1;a<=3;a++)  for (c=1;c<=2;c++)  {  if (place(a,b,c)==1) //如果可以放置  {  e++;  temp=map[a][b];  map[a][b]=c;  Try(x,y,k,d+1);  map[a][b]=temp;  }  if (e==0) //如果不能放置任何火柴  {  if ((d%2)==1)  win[x][y]=k; //如果奇数入赢队列  if ((d%2)==0)  lose[x][y]=k; //如果偶数入输队列  }  }  }  int main()  {  int t;  freopen("matches2.in","r",stdin);  freopen("matches2.out","w",stdout);  cin>>N;  while (N>0)  {  for (i=1;i<=3;i++)  for (j=1;j<=4;j++)  cin>>temp[i][j];  for (j=1;j<=4;j++)  for (i=1;i<=3;i++)  {  if (temp[i][j]=='1') //1代表横，2代表竖  map[i][j]=2;  if (temp[i][j]=='-')  map[i][j]=1;  }  for (i=1;i<=3;i++) //穷举  for (j=1;j<=4;j++)  for (k=1;k<=2;k++)  {  if (place(i,j,k)==1)  {  t=map[i][j];  map[i][j]=k;  Try(i,j,k,1);  map[i][j]=t; //恢复  }  }  for (i=1;i<=3;i++)  for (j=1;j<=4;j++)  {  if ((win[i][j]!=0) && (lose[i][j]==0)) //如果赢队列里面答案元素不在输队列中为最优解  zui[i][j]=win[i][j];  if ((win[i][j]!=0) && (lose[i][j]==win[i][j])) //如果在，就是次优解  ci[i][j]=win[i][j];  }  for (i=1;i<=3;i++)  for (j=1;j<=4;j++)  if (zui[i][j]!=0) //如果有最优解，则输出  {  v=0;  cout<<i-1<<j-1;  if (zui[i][j]==2)  cout<<1;  if (zui[i][j]==1)  cout<<"-";  return 0;  }  if (v==1); //如果没有最优解，输出次优解  {  for (i=1;i<=3;i++)  for (j=1;j<=4;j++)  if (ci[i][j]!=0)  {  cout<<i-1<<j-1;  if (ci[i][j]==2)  cout<<1;  if (ci[i][j]==1)  cout<<"-";  return 0;  }  }  /\*for (i=1;i<=3;i++)  for (j=1;j<=4;j++)  cout<<i<<' '<<j<<' '<<win[i][j]<<' '<<lose[i][j]<<' '<<endl;\*/  N--;  }  return 0;  } |

## 推理游戏

【问题描述】推理游戏（BlackVienna.cpp/c/pas）POJ 3915

让李旭琳大跌眼镜的是，小墨老师对学习魔法知识并不是很上心，他天天沉迷于各种游戏之中，例如，他现在又和同学们玩起了Black Vienna游戏，该游戏的规则是这样的：有18张牌，分别标着A-R，游戏有三个人。游戏开始的时候每个人拿5张牌，剩下三张牌隐藏起来，大家都只能看见自己手中的牌。之后，从第一个人到第三个人轮流拿一张“审问牌”去问另外的一个人，“审问牌”上有三个字母，那个人必须诚实地回答自己手中有多少“审问牌”中的字母。哪个人能够最先推断出隐藏起来的牌是什么，那个人就取得胜利。现在你的任务是，给出三个人手上的牌和询问情况，你需要计算出最早在哪一次询问之后，有人能够推断出隐藏的牌。

例如三个玩家的初始状态为：

玩家1: DGJLP; 玩家2: EFOQR; 玩家3: ACHMN; 隐藏牌: BIK

第1回合:玩家１审问玩家２有无BJK; 回答为0

第2回合:玩家2审问玩家３有无ABK; 回答为1

第３回合:玩家３审问玩家２有无 DEF;回答为2

第4回合:玩家 1审问玩家2有无 EIL; 回答为1

第5回合:玩家2审问玩家3有无 FIP; 回答为0

第6回合:玩家3审问玩家１有无GMO; 回答为1

第7回合:玩家 1审问玩家２有无OQR;回答为 3

第8回合:玩家 2 审问玩家３有无ADQ; 回答为1

第9回合:玩家 3审问玩家1有无EGJ;回答为2

事实上，玩家１能在第８回合判断出隐藏牌是BIK。

【输入格式】

输入最多有12组测试数据，所有测试数据结束以０表示。每组测试数据第一行为一整数t，表示审问次数。且2 ≤ t ≤ 15。

下一行包含四个以空格分隔的字符串，表示三个玩家的牌和隐藏牌。

接下来的t行数据为每回合的数据，即审问者和被审问者，审问字母的字符串及被审问者提供的答案。

所有的字符串为从A到R的唯一字母，按严格递增的顺序排列。同样的查询字符串可能出现在多个回合的游戏。

【输出格式】

输出最早第几回合，即有玩家判断出隐藏牌。若无玩家能判断出，则输出字符串“?”。

【输入样例】

9

DGJLP EFOQR ACHMN BIK

2 BJK 0

3 ABK 1

2 DEF 2

2 EIL 1

3 FIP 0

1 GMO 1

2 OQR 3

3 ADQ 1

1 EGJ 2

3

ABCDE FGHIJ KLMNO PQR

3 BKQ 1

1 ADE 3

2 CHJ 2

0

【输出样例】

8

?

【算法分析】

如果我们能让计算机记下所有状态再排除，等到某人心中的状态少到足以推断出隐藏牌来，那就可以算出答案。

现在的问题在于怎样表示一个状态……如果只表示隐藏牌的话，那么这个逻辑推理是很无力的。举个例子说，如果某人对于ABC回答了2，对于ABF回答了1，然后另外一个人手中有F。那么对于此人来说，隐藏牌CDE这个状态与ABC为2不矛盾，与ABF为1也不矛盾，但是CDE就是不能成立。原因在于此人手中有F，那人手中一定没有F，但是他对ABF回答了1，那么A与B这两张牌他一定有且仅有一张。然而他又对ABC回答了2，那么此人手中一定有C，所以说隐藏牌中一定没有C，所以说CDE是可以排除的。这也就是说，状态里面一定要包含对另外两个人的牌的推断，这样才能处理这样的状况。

这样的话状态有C(13, 5) × C(8, 5) = 72072种，还是不多的。搜状态的时候加上一些可行性剪枝就可以很快搜出状态，然后按照顺序模拟一遍就行了。

参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106 | //推理游戏  #include<iostream>  using namespace std;    const int o = 3,h = 15, q = 5, m=13;//通过定义常数防止栈溢出  char c[h][3+1],d[o + 1][5+1];//审问内容,实际手牌  char card[o + 1][q+1],eson[m];//可能手牌  int a[h],b[h];//被审问的人,审问答案  int mas[o+1];  int v,w;//审问次数  int i,j,p;    int Max(int a,int b)//返回最大值  {  if(a>b)  return a;  else  return b;  }    int msn(char a1[],char a2[])  {  int n=0;  for(i=0;i<5;i++)  {  for(j=0;j<3;j++)  {  if(a1[i]==a2[j])  n++;  }  }  return n;  }  int mqn(char jook[][q+1])  {  int t=0;  while(t<w &&msn(jook[a[t]],c[t])==b[t])  t++;  return t;  }  void men(int r,char l[])  {  if(r==(18-5))  {  if (card[o][0] == l[r-o])  return;  v = Max(v, mqn(card));  }  else  {  int j;  for(j=0;j<=o;j++)  {  if((j<o && mas[j]<5) || (j==o && mas[j]<3))  {  card[j][mas[j]]=l[r];  mas[j]++;  men(r+1,l);  mas[j]--;  }  }  }  }  int main()  {  int k,t;  while(scanf("%d",&w)!=EOF && w>0)  {  for(i=0;i<=o;i++)  scanf("%s",&d[i]);  for(t=0;t<w;t++)  {  scanf("%d",&a[t]);  a[t]--;  scanf("%s",&c[t]);  scanf("%d",&b[t]);  }//输入  k=h;  for(p=0;p<o;p++)  {  for(int z=0;z<13;z++)  eson[z]=0;//将数组中元素初始为0  for(j=0;j<=o;j++)  {  if(j!=p)  strcat(eson,d[j]);//连接两数组  }  strcpy(card[p],d[p]);  mas[p] = q;  v = 0;  men(0,eson);  memset(card[p],'\0',sizeof(card[p]));//如上方的清0方法  mas[p]=0;  if(v<k)  k=v;  }  if(k<w)  printf("%d\n",k+1);  else  printf("?\n");  }  return 0;  } |

# 第七章　贪心问题

## 闭区间问题

【题目描述】闭区间问题（ClosedInterval.cpp/c/pas）FOJ 1230

通过魔法钟回来的张琪曼和魔法学院的其他学员一起研究营救李旭琳脱离“时空陷”的方法。他们建立了n个对历史时间线的监控点，每个监控点可监控历史上的一个时间段，我们可以简单的看作是 x 轴上 n 个闭区间。但有些监控点监控的时间段是重叠的，这会干扰监控的准确性。请尝试去掉尽可能少的闭区间，使剩下的闭区间都不相交。

【输入格式】

第一行为闭区间的个数n，随后n行为闭区间的２个端点。

【输出格式】

输出去掉尽可能少的闭区间的个数。

【输入样例】

　　3 　　(区间个数n,1<=n<=40,000)

　　10 20 (以下为闭区间的 2 个端点)

　　15 10

　　20 15

　　【输出样例】

２

【算法分析】

可用结构体来保存各个区间的始点和终点，Reg[i].left 和 Reg[i].right 。对每个区间的终点进行排序。统计时如果第一个区间的终点小于第二个区间的始点，即说明它是独立的,依此类推则可解决。

最后用总的区间数-独立的区间数=要求的删除最少的区间数

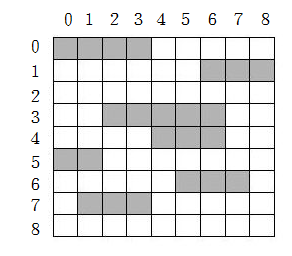
参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45 | //闭区间问题  #include <iostream>  #include <cstdio>  #include <algorithm>  using namespace std;  const int maxn=40002;  struct segment  {  int begin, end;  segment(int \_b=0, int \_e=0):begin(\_b),end(\_e){};  inline bool operator<( const segment& ss ) const  {//按照区间的右端点排序  return end<ss.end || (end==ss.end)&&(begin<ss.begin);  }  inline void input()  {  scanf("%d %d",&begin, &end);  if(begin>end)//交换，保证左端点值不比右端点大  begin^=end, end^=begin, begin^=end;  }  }seg[maxn];  int main()  {  freopen("ClosedInterval.in","r",stdin);  freopen("ClosedInterval.out","w",stdout);  int n;  while(scanf("%d",&n)!=EOF)  {  int i, res=1, limit;  for(i=0; i<n; i++)  seg[i].input();  sort(seg,seg+n);  limit=seg[0].end;  for(i=1; i<n; i++)  {//seg[i].begin<=limit的所有区间都是相互相交的，  //因为这些区间必然有公共点limit，即某一个区间的右端点  if(seg[i].begin>limit)  res++, limit=seg[i].end;  }  printf("%d\n",n-res);  }  return 0;  } |

## 穿越时空

【问题描述】穿越时空（Siworae.cpp/c/pas）POJ 1230

李旭琳要回到原来的时空，如图所示，在时空中有一些时空乱流（灰色区域）。时空乱流平行于X轴，宽度为一个单位，但长度各不相同，并且同一区域上不会有两个时空障碍。现在她要从上方沿Y轴方向，走到下方。途中可以穿越部分时空乱流，但会消耗一部分魔法力，所以穿越数有限制，不能超过一个值k。所以，要保证李旭琳无论从X轴哪点出发，都能走到下方，必须湮灭某些时空乱流，使得每条路上的时空乱流数都不超过穿越的限定值。现给定时空乱流的分布与穿越限定值，问至少湮灭多少时空乱流，可保证每条路上的时空乱流数不超过该限定值。例如此图中，当穿越限定值k=3时，李旭琳除了Ｘ轴为６的点外，可以从Ｘ轴的任何一点出发。



图

【输入格式】

输入有多组测试数据，第一行为组数t(1 <= t <= 10)，随后是每组测试数据，第一行为两个整数n (1 <= n <= 100)， 表示时空乱流数，和穿越限定值k (0 <= k <= 100)，以下n行表示时空乱流的起始坐标和结束坐标。

【输出格式】

每组一行数据，表示最少湮灭的时空乱流数。

【输入样例】

2

3 1

2 0 4 0

0 1 1 1

1 2 2 2

7 3　　（此例即为图中所示）

0 0 3 0

6 1 8 1

2 3 6 3

4 4 6 4

0 5 1 5

5 6 7 6

1 7 3 7

【输出样例】

１

１

【算法分析】

每次湮灭一个时空乱流，需选取对超过限定值的列影响尽量大的时空乱流。即移走后能使尽量多的列符合穿越限定值。由于数据范围比较小，所以可以从左向右扫描场地的每一列是否合法。若不合法，贪心的找出从该列起向右延伸最长的m道时空乱流，移除这m道时空乱流使得该列合法。时间复杂度为O(n2)。

注意输入的时空乱流的两个坐标未必是从左到右，也有从右到左的，此外，时空乱流有可能相邻，故不能直接以01标记有无时空乱流。

参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81 | //穿越时空  #include <iostream>  #include <cstdlib>  #include <algorithm>  using namespace std;  #define M 113  int k,n;  int s[M][M],v[M];  int main()  {  freopen("Siworae.in","r",stdin);  freopen("Siworae.out","w",stdout);  int i,j,t;  int x1,y1,x2,y2,mx,my;  int ans,tx;  scanf("%d",&t);  while(t--)  {  scanf("%d%d",&n,&k);  memset(v,0,sizeof(v));//记录每列时空乱流数  memset(s,0,sizeof(s));//记录时空乱流分布  mx=0;my=0;//统计当前图区域  for(i=0;i<n;i++)  {  scanf("%d%d%d%d",&x1,&y1,&x2,&y2);  if(x1<=x2)  for(j=x1;j<=x2;j++)  {  s[j][y1]=i+1;  v[j]++;  }  else  for(j=x2;j<=x1;j++)  {  s[j][y1]=i+1;  v[j]++;  }  if(x1>mx)  mx=x1;  if(x2>mx)  mx=x2;  if(y1>my)  my=y1;  if(y2>my)  my=y2;  }  ans=0;  for(i=0;i<=mx;i++)  {  if(v[i]<=k)  continue;  for(j=0;j<v[i]-k;j++)//该行需移走的时空乱流数  {  y1=0,y2=-1;  for(x1=0;x1<=my;x1++)  {  if(s[i][x1]==0)  continue;  for(x2=i,tx=0;s[x2][x1]==s[i][x1];x2++)  if(v[x2]>k)  tx++;  if(tx>y1)  {  y1=tx;  y2=x1;  }  }  for(x2=i+1,x1=y2;s[x2][x1]==s[i][x1];x2++)  {  s[x2][x1]=0;  v[x2]--;  }  ans++;  }  }  printf("%d\n",ans);  }  return 0;  } |

## 预算

【题目描述】预算（budget.cpp/c/pas）NOIP 1999　旅行家的预算

邪狼：“天哪！这么多魔法石都不够用！”

修罗王：“没办法，通货膨胀，物价太高了。”

修罗王两人为了节省花费，决定驾驶最老式的燃油汽车以最少的费用从一个城市到另一个城市(假设出发时油箱是空的)。给定两个城市之间的距离D1、汽车油箱的容量C(以升为单位)、每升汽油能行驶的距离D2、出发点每升汽油价格P和沿途油站数N(N可以为零)，油站i离出发点的距离Di、每升汽油价格Pi(i＝1，2，…，N)。计算结果四舍五入至小数点后两位。如果无法到达目的地，则输出“No Solution”。

【输入格式】

输入五个数，即D1，C，D2，P，N。

【输出格式】

输出最小费用，如无法到达目的地，则输出“No Solution”。

【输入样例】

275.6 11.9 27.4 2.8 2 (分别表示D1，C，D2，P，N)

102.0 2.9 (以下共N行，分别表示油站i离出发点的距离Di和每升汽油价格Pi)

220.0 2.2

【输出样例】

26.95(该数据表示最小费用)

【算法分析】

枚举显然是不行的。

从后向前找到油价最低的加油站，显然车至该站油箱应为空，则以该加油站为界将路程划分为两段分别求其最小费用，二者之和即为总费用。

依此法划分，分以下几种情况：

（１）若某段只有起点与终点两个加油站时无需再分;

（２）若某一段油价最低的加油站为起点，如能一次加油到达该段终点最好，否则，则加满油再考虑油箱有油情况下的二分法。即考虑起点之外所有的加油站中从后往前油价最低的加油站。分两种情况：

若该加油站位于起点加满油后不能到达之处，则到达该站时油箱应该为空，以该加油站为界将全程分为两个独立段考虑，前半段为有油情况，后半段为无油情况。

若该加油站处于起点加满油后能到达之处，则将该段总路程缩短为该加油站至终点的情况，该加油站在该段路程中最便宜，若从该站加满油仍不能到达终点，则继续分治即可。

下面的程序用贪心算法解决，算法伪代码为：

寻找距离当前站最近的比当前站便宜的站点

　　如果找到了，油量够就直接开过去，油量不够就加油到刚好可以开过去；

　　如果找不到，就到前面找一个充满油量能到得了的最便宜的站点，充满油开过去；

　　如果加满油找不到任何站点，那就输出No Solution。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98 | //预算－贪心法  #include <stdio.h>  #include <stdlib.h>  struct dot  {  double dis, p;  }dots[100001];  double have = 0, used = 0;  double d1, c, d2, p;  int place = 0;    int com(const void \*a, const void \*b)  {  struct dot i = \*(struct dot \*)a, j = \*(struct dot \*)b;  return i.dis - j.dis;  }    void driveto(int id)  {  place = id;  }    int main()  {  freopen("budget.in","r",stdin);  freopen("budget.out","w",stdout);  int i;  int n, id;  double small;  scanf("%lf%lf%lf%lf%d", &d1, &c, &d2, &p, &n);  dots[0].p = p;  dots[n + 1].dis = d1;  for(i = 1; i <= n; i++)  {  scanf("%lf%lf", &dots[i].dis, &dots[i].p);  }  qsort(dots, n + 2, sizeof(struct dot), com);  while(dots[place].dis < d1)  {  id = -1;  for(i = place + 1; i <= n + 1; i++)  {  if(dots[place].dis + c \* d2 >= dots[i].dis)  {  if(dots[i].p <= dots[place].p)  {  id = i;  break;  }  }  else  break;  }  if(i == place + 1 && id == -1)  {  printf("No Solution\n");  return 0;  }  if(id != -1)  {  if(dots[place].dis + have \* d2 >= dots[id].dis)  {  have -= (dots[id].dis - dots[place].dis) / d2;  }  else  {  used += ((dots[id].dis - dots[place].dis) / d2 - have) \* dots[place].p;  have = 0;  }  }  else  {  small = 100000000;  id = -1;  for(i = place + 1; i <= n + 1; i++)  {  if(dots[place].dis + c \* d2 >= dots[i].dis)  {  if(small >= dots[i].p)  {  small = dots[i].p;  id = i;  }  }  else  break;  }  used += (c - have) \* dots[place].p;  have = c;  have -= (dots[id].dis - dots[place].dis) / d2;  }  place = id;  }  printf("%.2lf\n", used);  return 0;  } |

# 第八章　递推

## 曲线分割

【问题描述】曲线分割（curve.cpp/c/pas）

设有n条封闭曲线画在平面上，而任何两条封闭曲线恰好相交于两点，且任何三条封闭曲线不相交于同一点，问这些封闭曲线把平面分割成的区域个数。

【输入格式】

输入整数n。

【输出格式】

输出分割的区域个数。

【输入样例】

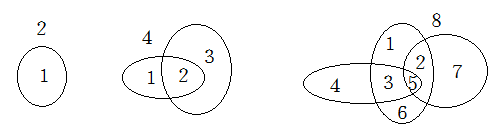
2

【输出样例】

4

【算法分析】

下图为n=1，n=2，n=3的情形：



设满足条件的n条封闭曲线所分割成的域的数目为 an ，其中 n-1 条封闭曲线所分割成的域的数目为 an-1 第 n 条封闭曲线和这些曲线相交于 2(n-1) 个点，这 2(n-1) 个点把第 n 条封闭曲线截成 2(n-1) 条弧，每条弧把 2(n-1) 个域中的每个域一分为二。故新增加的域数为 2(n-1)

an = an-1 + 2(n-1)

a1 = 2

参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | #include <iostream>  using namespace std;  int f(int n)  {  if(n==1)  return 2;  else  return f(n-1)+2\*(n-1);  }  int main()  {  int n;  while(1)  {  cin>>n;  cout<<f(n)<<endl;  }  return 0;  } |

继续化简后得到：an=n×n-n+2，参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | //曲线分割  #include <iostream>  #include <cstdlib>  using namespace std;  int main()  {  freopen("curve.in","r",stdin);  freopen("curve.out","w",stdout);  int n;  cin>>n;  cout<<n\*n-n+2<<endl;  return 0;  } |

# 第九章　搜索算法

## 骑士遍历超级版

【题目描述】骑士遍历超级版（knight3.cpp/c.pas)

楚继光判断邪狼可能藏在一个n×n的（n≦1000）正方形区域，楚继光骑着战马从任一点A（x，y）开始，试图找出一条路径，使马不重复地走遍区域的每一个点。马走的规则是走日字，可向任意方向走。

【输入格式】

三个整数n，x， y。 n代表棋盘大小，x，y代表A点坐标，棋盘下标从（0，0开始）。

【输出格式】

任一种棋盘路径。

【输入样例】

5 2 1

【输出样例】

23 4 13 8 21

12 7 22 3 14

17 24 5 20 9

6 11 18 15 2

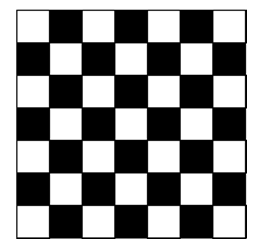
25 16 1 10 19

【算法分析】

对于本问题，用深度优先搜索法求解由于算法复杂度是指数级的，当棋盘较大时就会超时，如果只要求骑士遍历问题的一个解，这里采用Warnsdoff策略求解，这也是一种贪婪法，其选择下一出口的贪婪标准是在那些允许走的位置中，选择出口最少的那个位置。如马的当前位置（i，j）只有三个出口，他们是位置（i+2，j+1）、（i-2，j+1）和（i-1，j-2），如分别走到这些位置，这三个位置又分别会有不同的出口，假定这三个位置的出口个数分别为4、2、3，则程序就选择让马走向（i-2，j+1）位置。

由于程序采用的是一种贪婪法，整个找解过程是一直向前，没有回溯，所以能非常快地找到解。但是，对于某些开始位置，实际上有解，而该算法不能找到解。对于找不到解的情况，程序只要改变8种可能出口的选择顺序，就能找到解。改变出口选择顺序，就是改变有相同出口时的选择标准。

另外，如图所示，当棋盘大小为奇数时，马从黑格出发是不能走遍棋盘的。



图

因为从图中可以看出，在国际象棋中黑格里的马跳一步必定到达一个白格，同样白格里的马跳一步必定到达黑格，而当棋盘边长n为奇数格时，棋盘上所有的格子的总和也是奇数，此时白格比黑格多出一格，马从某一黑格出发连跳n×n-2步后，到达一个白格，此时盘上还剩下最后一个白格，无论这个白格位于何处，都不可能一步到达。

演示代码如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111 | //骑士遍历超级版  #include <stdio.h>  #define INF 100  int board[8][8],n;  int start;  int mover[]={-2,-1,1,2, 2, 1,-1,-2};  int movec[]={ 1, 2,2,1,-1,-2,-2,-1};  int numable(int r, int c, int nexta[])//返回下次可以找到的增量  {  int i, k, a, b;  int num = 0;  for(i = 0;i <= 7;i++)  {  k = (i + start) % 8;//1-8的循环滚动  a = r + mover[k];  b = c + movec[k];  if(a<=7 && a>=0 && b>=0 && b<=7 && board[a][b]==0)  {  nexta[num] = k;  num++;  }  }  return num;  }  int number(int r, int c)/\*返回下次可以找到的增量的个数\*/  {  int i, k, a, b;  int num = 0;  for(i = 0;i <= 7;i++)  {  k = (i + start) % 8;  a = r + mover[k];  b = c + movec[k];  if(a<=7 && a>=0 && b>=0 && b<=7 && board[a][b]==0)  num++;  }  return num;  }  int next(int r, int c)//找到最少的增量步，若是-1则无路  {  int nexta[8], num, num1 = 0, minNum, i, k;  minNum = INF;  num = numable(r, c, nexta);//计算（r,c)有多少出路  if(num == 0)  return -1; //没有出口    for(i = 0;i <= num - 1;i++)//找到最少出口  {  num1 =number(r+mover[nexta[i]],c+movec[nexta[i]]);  if(num1 <= minNum)  {  minNum = num1;  k = nexta[i];  }  }  return k;  }  int main()  {  int x, y;  int step, caseCounter;  int r, c;  int i, j, k;  scanf("%d%d", &x, &y);//接收起点  start = 0;  caseCounter = 1;  while(start <= 7)  {  for(i = 0;i <= 7;i++)//初始化  for(j = 0;j <= 7;j++)  board[i][j] = 0;  r = x;  c = y;  board[r][c] = 1;//起点  step = 2;//下一个位置  while(1)  {  if(step > 64)//走满64步，则打印结果  {  printf("Case %d:\n", caseCounter++);  for(i = 0;i <= 7;i++)  {  for(j = 0;j <= 7;j++)  printf("%2d ", board[i][j]);  printf("\n");  }  start++;  break;  }    k = next(r, c);  if(k == -1)  {  start++;  break;  }  r = r + mover[k];  c = c + movec[k];  board[r][c] = step;  step++;  }  }  return 0;  } |

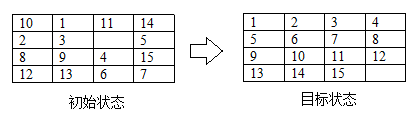
该程序只计算了棋盘大小为8时的走法，且未对棋盘格数为奇数的情况进行优化，请参照该演示代码写出完整的程序。

## 15数码问题

【题目描述】15数码问题（Puzzle15.cpp/c/pas）uva 10181

“呵呵，你真以为这世上有无所不能的神？实话告诉你，很久以前，宇宙中的几大黑暗势力就在你们星球上通过创立各种宗派收罗信众，并暗中挑动了不计其数的宗派战争以收割你们人类的生命力（命源）。所谓的创世神传说，也不过是我们在远古时代就布局抛给你们的诱饵而已。当然我们比他们要仁慈，象他们那种通过战争来收割生命力（命源）的低级手段就象杀鸡取卵一样愚蠢，但再愚蠢，也比不过你们彼此残杀数十万年而不醒悟的愚蠢。看到这道15谜问题了吗，当你解开它，你就会知道我们的来历了。”所谓的守护者在吸收了赢来的魔法石上足够的能量后，对修罗王说。

15数码问题是在一个4×4的方格棋盘上，将数字1，2，3，…，14，15以任意顺序置入棋盘的各个方格中，空出一格，通过有限次移动，把一个给定的初态变成目标状态，如下图。移动规则是：每次只能在空格周围的四个数字中任选一个移入空格。可以证明的是，一共16！的初态中，有一半是不可能移成目标状态的。



【输入格式】

第一行为一个整数Ｎ，表示有Ｎ组数据，随后是Ｎ组4×4的棋盘初始状态描述。

【输出格式】

若在５０步内不能完成，输出“This puzzle is not solvable.”，否则输出步数如样例所示。其中'R'，'L'，'U'和'D'分别代表左，右，上和下。

【输入样例】

2

2 3 4 0

1 5 7 8

9 6 10 12

13 14 11 15

13 1 2 4

5 0 3 7

9 6 10 12

15 8 11 14

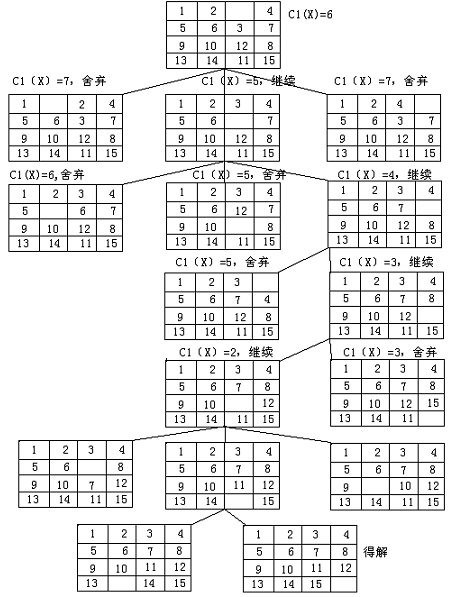
【输出样例】

LLLDRDRDR

This puzzle is not solvable.

【算法分析】

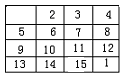
这道题如果搜索整个空间，工作量太大，我们可以采取分支定界的方法，找到某个判定函数，例如，建立判定函数C1（X）,意思是在节点X的状态下，没有达到目标状态下正确位置的数字个数。请看下图：



我们还可以再设一个判定函数



第二个估值函数如果当搜索过程出现下图的情形时，会更加符合实际情况，估值函数选得好坏，对算法效率有很大影响，很难肯定应怎样建立合理的估值函数，因为它涉及问题的固有特性和数学描述方式等。只有对问题实质了解得越透彻，数字描述越适当，建立的估值函数就可能更好些。对于某些类型的问题，常用问题的目标函灵敏本身做估值函数。



实际上使用IDA\*的八数码程序即可完美解决该题，参考程序如下所示：

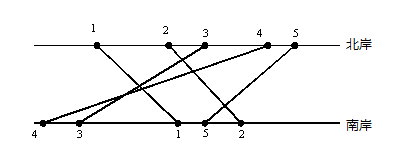
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155 | //１５数码问题 -IDA\*算法  #include<iostream>  #include<cstdlib>  #include<cmath>  #define size 4  using namespace std;  int move[4][2]={{-1,0},{0,-1},{0,1},{1,0}};//上,左,右,下增量  char op[4]={'U','L','R','D'};  int map[size][size],map2[size\*size],limit,path[100];  int flag,length;  //int goal\_st[3][3]={{1,2,3},{4,5,6},{7,8,0}};//目标状态  //goal存储目标位置，即０存在（３,３）,1存在（0,0）...  int goal[16][2]= {{3,3},{0,0},{0,1}, {0,2},  {0,3},{1,0},{1,1}, {1,2},  {1,3},{2,0},{2,1}, {2,2},  {2,3},{3,0},{3,1}, {3,2}};    int h(int a[size\*size])//求逆序数  {  int i,j,num,w,x,y;  num=0;  for(i=0;i<size\*size;i++)  {  if(a[i]==0)  w=i;  for(j=i+1;j<size\*size;j++)  {  if(a[i]>a[j])  num++;  }  }  x=w/size;  y=w%size;  num+=abs(x-3)+abs(y-3);  if(num%2==1)  return 1;  else  return 0;  }  int manhattan(int a[][size])//计算曼哈顿距离，小等于实际总步数  {  int i,j,cost=0;  for(i=0;i<size;i++)  for(j=0;j<size;j++)  {  int w=map[i][j];  cost+=abs(i-goal[w][0])+abs(j-goal[w][1]);  }  return cost;  }  void swap(int\*a,int\*b)  {  int tmp;  tmp=\*a;  \*a=\*b;  \*b=tmp;  }  void dfs(int sx,int sy,int dep,int pre\_move)//sx,sy是空格的位置  {  int i,j,nx,ny;  if(flag)  return;  int dv=manhattan(map);  if(dep==limit)  {  if(dv==0)  {  flag=1;  length=dep;  return;  }  else  return;  }  else if(dep<limit)  {  if(dv==0)  {  flag=1;  length=dep;  return;  }  }  for(i=0;i<4;i++)//４个方向尝试  {  if(i+pre\_move==3&&dep>0)//不和上一次移动方向相反，对第二步以后而言  continue;  nx=sx+move[i][0];  ny=sy+move[i][1];  if(0<=nx && nx<size && 0<=ny&&ny<size)//如果可以移动  {  swap(&map[sx][sy],&map[nx][ny]);//交换两位置  int p=manhattan(map);  if(p+dep<=limit&&!flag)  {  path[dep]=i;  dfs(nx,ny,dep+1,i);  if(flag)  return;  }  swap(&map[sx][sy],&map[nx][ny]);  }  }  }  int main()  {  freopen("Puzzle15.in","r",stdin);  freopen("Puzzle15.out","w",stdout);  int i,j,k,l,m,n,sx,sy;  char c,g;  i=0;  scanf("%d",&n);  while(n--)  {  flag=0;length=0;  memset(path,-1,sizeof(path));  for(i=0;i<16;i++)  {  scanf("%d",&map2[i]);  if(map2[i]==0)  {  map[i/size][i%size]=0;  sx=i/size;sy=i%size;  }  else  {  map[i/size][i%size]=map2[i];  }  }  if(h(map2)==1)//该状态可达  {  limit=manhattan(map);  while(!flag&&length<=50)//题中要求50步之内到达  {  dfs(sx,sy,0,0);  if(!flag)  limit++; //得到的是最小步数  }  if(flag)  {  for(i=0;i<length;i++)  printf("%c",op[path[i]]);  printf("\n");  }  }  else if(!h(map2)||!flag)  printf("This puzzle is not solvable.\n");  }  return 0;  } |

# 第十一章　动态规划

## 友好城市

【题目描述】友好城市（Ship.cpp/c/pas）

一条河从东向西流过，并把魔法世界分为南北两个部分。河的两岸各有n个城市，且北岸的每一个城市都与南岸的某个城市是友好城市，而且对应的关系是一一对应的。如图所示。



现在要求在两个友好城市之间建立一条航线，但为了安全起见，所有航线都不能相交，因此，不是所有的友好城市都能建立航线。请问，最多能建多少航线？

【输入格式】

第一行两个由空格分隔的整数x,y，10<=x<=6000,10<=y<=100。X表示河的长度而y表示宽，第二行是一个整数n(1<=n<=5000),表示分布在河两岸的城市对数。接下来的n行每行有两个由空格分隔的正数c,d(c,d<=x),描述每一对友好城市与河起点的距离， c表示北岸城市的距离，而d表示南岸城市的距离。在河的同一边，任何两个城市的位置都是不同的。

【输出格式】

在安全条件下能够开通的最大航线数目。

【输入样例】

30 4

5

4 5

2 4

5 2

1 3

3 1

【输出样例】

3

【算法分析】

只需对南岸的编号中找出最长不下降序列即可。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77 | //友好城市  #include<iostream>  #include<fstream>  #define M 10000  using namespace std;  ifstream fin("city.in");  ofstream fout("city.out");  int a[M+1];  int L[M+1];  int m,n;  struct pp  {  int n,s;  }s[M+1];  int cmp(const void \*p,const void \*q)  {  pp \*a=(pp \*)p,\*b=(pp \*)q;  if(a->n > b->n) return 1;  if(a->n < b->n) return -1;  return 0;  }  int half(int k)  {  int begin,end,mid;  begin=0;end=m;  while(begin<=end)  {  mid=begin+end>>1;  if(s[L[mid]].s>=k) end=mid-1;  else begin=mid+1;  }  if(s[L[mid]].s>=k) return mid-1;  else return mid;  }  void init()  {  fin>>m>>m;  fin>>n;  for(int i=1;i<=n;++i)  fin>>s[i].n>>s[i].s;  }  void dp()  {  int i,j,k=0;  m=1;  L[1]=1;  a[1]=1;  for(i=2;i<=n;++i)  {  j=half(s[i].s);  a[i]=a[L[j]]+1;  L[a[i]]=i;  if(a[i]>m) m=a[i];  }  for(i=1;i<=n;++i)  if(a[i]>k)  k=a[i];  fout<<k<<endl;  }  int main()  {  init();  s[0].n=-1;  qsort(s,n+1,sizeof(s[1]),cmp);  dp();    fin.close();  fout.close();  return 0;  } |

## 合唱团

【问题描述】合唱团（chorus.cpp/c/pas）NOIP 2004

N位同学站成一排，墨老师要请其中的(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<...<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。

【输入文件】

输入文件chorus.in的第一行是一个整数N(2<=N<=100)，表示同学的总数。第一行有n个整数，用空格分隔，第i个整数Ti(130<=Ti<=230)是第i位同学的身高(厘米)。

【输出文件】

输出文件chorus.out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

【数据规模】

对于50％的数据，保证有n<=20；

对于全部的数据，保证有n<=100。

【算法分析】

动态规划。最基本的想法是：枚举中间最高的一个人，接着对它的左边求最长上升序列（注意序列中最高的同学不应高过基准），对右边求最长下降序列（同样的，序列中最高的同学不应高过基准）。时间复杂度为O(n3)，算法实现起来也很简单。

接着对这个算法进行分析，我们不难发现，假如还是基于枚举一个同学的话，设Incsq[i]表示了1 - i的最长上升序列，Decsq[i]表示了i - n的最长下降序列，那么，

Current[i] = Incsq[i] + Decsq[i] - 1（两个数组中i被重复计算了）

那么，我们只需要先求好最长上升和下降序列，然后枚举中间最高的同学就可以了。

算法还可进一步优化：

求最长上升序列的经典状态转移方程为：

opt[i] = max{opt[j]+1, 其中i<j<=n, 且list[j]>list[i]}

我们对状态转移方程稍微做一些修改：

opt[i] = max{opt[i+1], min{j, rec[j]>=list[i]}}

rec[j] = list[i]

很明显可以看出，在opt[i]的寻找j的过程当中，查询序列是单调的，于是可以用二分法，就十分巧妙地在logn的时间内找到指定的j，而问题的总体复杂度为O(nlogn)。这样，这个问题的算法效率就得到了大幅度的提升，即便n是106，也可以轻松应对。

参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | //合唱团  #include<stdio.h>  #include<stdlib.h>  int T[201];  int f[201],g[201];  int main()  {  freopen("chorus.in","r",stdin);  freopen("chorus.out","w",stdout);  int n,i,j,min,k,R,L;  scanf("%d",&n);  for(i=1;i<=n;i++)  scanf("%d",&T[i]);  for(i=1;i<=n;i++)  {  f[i]=1;  for(j=1;j<=i-1;j++)  if(T[j]<T[i]&&f[j]+1>f[i])  f[i]=f[j]+1;  }  for(i=n;i>=1;i--)  {  g[i]=1;  for(j=i+1;j<=n;j++)  if(T[j]<T[i]&&g[j]+1>g[i])  g[i]=g[j]+1;  }  min=n;  for(k=1;k<=n;k++)  {  L=f[k],R=g[k];  if(n-L-R+1<min)  min=n-L-R+1;  }  printf("%d\n",min);  } |

## 采药

　　【题目描述】采药（medicine.cpp/c/pas)　noip 2005普及组

山洞里有一些不同的草药，采每一株都需要一些时间，每一株也有它自身的价值，在一段时间内如何让采到的草药价值最大。

【输入格式】

第一行有两个用空格隔开的整数T和M（1<=T,M<=100），T代表总共采药时间，M代表草药数目。接下来的M行每行包括两个在1到100之间（包括1和100）的整数，分别表示采摘某种草药的时间和这株草药的价值。

【输出格式】

只包含一个整数，表示在规定的时间内可以采到的草药的最大总价值。

【输入样例】

70 3

71 100

69 1

1 2

【输出样例】

3

参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | //采药  #include <stdio.h>  #include <stdlib.h>  main()  {  freopen("medic.in","r",stdin);  freopen("medic.out","w",stdout);  int t,m,i,j,a[101],b[101],f[101][1001];  scanf("%d%d",&t,&m);  for (i=1;i<=m;i++)  scanf("%d%d",&a[i],&b[i]);  for (i=0;i<=t;i++)  f[0][i]=0;  for (i=1;i<=m;i++)  for (j=0;j<=t;j++)  {  f[i][j]=f[i-1][j];  if ((j>=a[i])&&(f[i][j]<f[i-1][j-a[i]]+b[i]))  f[i][j]=f[i-1][j-a[i]]+b[i];  }  printf("%d",f[m][t]);  return(0);  } |

## 简单背包问题２

【题目描述】(pack2.cpp/c/pas) NOIP 2001

张琪曼和李旭琳有一个背包容量为v(正整数，o≤v≤20000)，同时有n个魔法石(o≤n≤30)，每个魔法石有一个体积 (正整数)。要求从m个魔法石中，任取若干个装入包内，使背包的剩余空间为最小。

【输入格式】

第一行为一个整数，表示背包容量，第二行为一个整数，表示有n个魔法石，接下来n行，分别表示这n个魔法石的各自体积。

【输出格式】

只有一个整数，表示背包剩余空间。

【输入样例】

24

6

8

3

12

7

9

7

【输出样例】

0

### 穷举法★

最朴素的算法是穷举法，但不能通过全部数据。

因为n个魔法石中的每个都有装和不装两种选择，故可以1和0表示。穷举所有的n位二进制数进行统计即可。

参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | //简单背包问题２  #include <stdio.h>  int a[30+1];//状态为0-1，装入为1，不装入为0  long v,n,i,j,s,f;  long t[30+1];//保存每物品的体积  int main()  {  scanf("%ld%ld",&v,&n);  for(i=1;i<=n;i++)  scanf("%ld",&t[i]);  memset(a,0,sizeof(a));  s=0;  while(a[0]==0)//穷举  {  f=0;  for(i=1;i<=n;i++)  f=f+a[i]\*t[i];//计算不同状态的总重量  if((f>s)&&(f<=v))  s=f;  j=n;  while(a[j]==1)//进位  j=j-1;  a[j]=1;  for(i=j+1;i<=n;i++)  a[i]=0;  }  printf("%ld\n",v-s);  return 0;  } |

### 动态规划１★

使用二维数组f[i][j]，表示前i个物品装入容量为j的背包能获得的最大体积，则动态转移方程为： f[i][j]=max{f[i-1][j]，f[i-1][j-a[i]]+a[i]}。该方程的意思是，要么第i件物品不放，即f[i-1][j]；要么放第i件物品，这样就必须从j容量的背包里腾出第i件物品的空间才行，即f[i-1][j-a[i]]，然后加上a[j]的重量，即f[i-1][j-a[i]]+a[i]。

参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | //简单背包问题２－动规１  #include <iostream>  using namespace std;  int f[31][20001];  int main()  {  freopen("pack2.in","r",stdin);  freopen("pack2.out","w",stdout);  int V,n,i,j;  cin>>V>>n;  int a[n+1];  for(i=1;i<=n;i++)  cin>>a[i];  for(i=1;i<=n;i++)  for(j=1;j<=V;j++)  if(j<a[i])  f[i][j]=f[i-1][j];  else  f[i][j]=max(f[i-1][j],f[i-1][j-a[i]]+a[i]);  cout<<V-f[n][V]<<'\n';  return 0;  } |

### 动态规划２★

前一种方法占用内存空间较大，可以改用一维数组f[j]，仍表示前i个物品装包能获得的最大体积。

　　参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | //简单背包问题２-动规２  #include <iostream>  #include <cstdlib>  using namespace std;  int f[20001];//f[j]表示前i个物品装入获得的最大体积  int main()  {  freopen("pack2.in","r",stdin);  freopen("pack2.out","w",stdout);  int V,n,i,j;  cin>>V>>n;  int a[n+1];  for(i=1;i<=n;i++)  cin>>a[i];  for(i=1;i<=n;i++)  for(j=V;j>=1;j--)  if(j<a[i])  f[j]=max(f[j-1],f[j]);//虽然f[j]总是大于f[j-1],但这句不能少  else  f[j]=max(f[j],f[j-a[i]]+a[i]);  cout<<V-f[V];  return 0;  } |

### 动态规划３★

　继续压缩空间，设布尔数组f[20001]，f[i]=1表示容量为i的背包空间可被装满，f[i]=0即表示容量为i的背包空间无法装满。若f[j]=1，下一个物品为a[i]，若该物品可装，显然f[j+a[i]]也为1。

　　参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | //简单背包问题２－动规３  #include <iostream>  #include <fstream>  using namespace std;  ifstream fin("pack2.in");  ofstream fout("pack2.out");  bool f[20001];  int main()  {  int v,n,a[31],i,j,d;  fin>>v>>n;    for(i=1;i<=n;i++)  fin>>a[i];    for(i=1;i<=n;i++)//对n种物品  {  for(j=v;j>0;j--)//j为剩余空间  if(f[j]==1 && j+a[i]<=v)  f[j+a[i]]=1;  f[a[i]]=1;  }  d=v;  while(d>0 && f[d]!=1)//找出最大装载空间  d--;    fout<<v-d<<endl;  fin.close();  fout.close();  return 0;  } |

## 预算

【题目描述】预算（budget.cpp)

叶妍霜等人要为太空战指挥中心购置设备，魔法学院的院长昨天说：“指挥中心需要购买哪些设备，你们研究了算，只要不超过N元钱就行”。所以今天一早，叶妍霜就开始做预算了，她把想买的物品分为两类：主件与附件，附件是从属于某个主件的，表3.5就是一些主件与附件的例子：

表3.5

|  |  |
| --- | --- |
| 主件 | 附件 |
| 广播系统 | 音箱，麦克风 |
| 资料柜 | 密码锁 |
| 控制台 | 电子地图，激光笔 |
| 记录仪 | 无 |

如果要买归类为附件的物品，必须先买该附件所属的主件。每个主件可以有0个、1个或2个附件。附件不再有从属于自己的附件。指挥中心想配备的东西很多，肯定会超过院长限定的N元。于是，她把每件物品规定了一个重要度，分为5等：用整数1~5表示，第5等最重要。她还从互联网上查到了每件物品的价格（都是10元的整数倍）。她希望在不超过N元（可以等于N元）的前提下，使每件物品的价格与重要度的乘积的总和最大。

设第j件物品的价格为v[j]，重要度为w[j]，共选中了k件物品，编号依次为j1，j2，…，jk，则所求的总和为：

v[j1]×w[j1]＋v[j2]×w[j2]+ …＋v[jk]×w[jk]。

请你帮助叶妍霜设计一个满足要求的购物单。

　　【输入格式】

第1行为两个正整数，用一个空格隔开：N m（其中N＜32000表示总钱数，m＜60为希望购买物品的个数。）

从第2行到第m＋1行，第j行给出了编号为j－1的物品的基本数据，每行有3个非负整数v p q（其中v表示该物品的价格（v＜10000），p表示该物品的重要度（1~5），q表示该物品是主件还是附件。如果q＝0，表示该物品为主件，如果q＞0，表示该物品为附件，q是所属主件的编号）

　　【输出格式】

输出只有一个正整数（＜200000），为不超过总钱数的物品的价格与重要度乘积的总和的最大值。

【输入样例】

1000 5

800 2 0

400 5 1

300 5 1

400 3 0

500 2 0

【输出样例】

2200

【算法分析】

本题是有依赖的01背包问题，每个主件有四种选择方案：

1. 即只选主件;
2. 选主件和附件1;
3. 选主件和附件2;
4. 选主件和全部附件。

可以将这四种选择方案看成四块石头（物品），但这四块石头只能任取一种方案即可。

在掌握了树型动态规划算法后，还可以尝试使用树型动态规划算法解决该题。

参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61 | //预算  #include <bits/stdc++.h>  using namespace std;  const int Maxn = 40000,Maxm = 70;  int V[Maxm][3],P[Maxm][3];//物品的价值，重要度  int F[Maxm][Maxn];  int n,m,v,p,q;  void Init()  {  cin>>n>>m;  for(int i=1; i<=m; i++)  {  cin>>v>>p>>q;  if(q==0)//主件  {  V[i][0]=v;  P[i][0]=p;  }  else if(V[q][1]==0) //第一个附件  {  V[q][1]=v;  P[q][1]=p;  }  else//第二个附件  {  V[q][2]=v;  P[q][2]=p;  }  }  }  void DP()  {  for(int i=1; i<=m; i++)//枚举各物品  for(int j=1; j<=n; j++)//钱可以看成是背包容量  if(j>=V[i][0])//如果钱够买该物品  {  F[i][j]=max(F[i-1][j],F[i-1][j-V[i][0]]+V[i][0]\*P[i][0]);//只购买主件  if(j-V[i][0]-V[i][1]>=0)//购买主件和第一附件  F[i][j] = max(F[i][j],F[i-1][j-V[i][0]-V[i][1]]+V[i][0]\*P[i][0]+V[i][1]\*P[i][1]);  if(j-V[i][0]-V[i][2]>=0)//购买主件和第二附件  F[i][j] = max(F[i][j],F[i-1][j-V[i][0]-V[i][2]]+V[i][0]\*P[i][0]+V[i][2]\*P[i][2]);  if(j-V[i][0]-V[i][1]-V[i][2]>=0)//购买主件和全部附件  F[i][j] = max(F[i][j],F[i-1][j-V[i][0]-V[i][1]-V[i][2]]+V[i][0]\*P[i][0]+V[i][1]\*P[i][1]+V[i][2]\*P[i][2]);  }  else  F[i][j]=F[i-1][j];  }  int main()  {  freopen("budget.in","r",stdin);  freopen("budget.out","w",stdout);  Init();  DP();  cout<<F[m][n]<<endl;  return 0;  } |

## 货币面值

　　【题目描述】货币面值（Currency.cpp/c/pas）九度OJ 1531

魔法世界发行了很多不同面额的纸币，试求出用这些纸币进行任意的组合不能表示的最小面额是多少。

【输入格式】

输入包含多个测试用例，每组测试用例的第一行输入一个整数N（N<=100）表示流通的纸币面额数量，第二行是N个纸币的具体表示面额，取值[1，100]。

【输出格式】

对于每组测试用例，输出一个整数，表示已经发行的所有纸币都不能表示的最小面额（已经发行的每个纸币面额最多只能使用一次 ，但面值可能有重复）。

【输入样例】

5

1 2 3 9 100

5

1 2 4 9 100

5

1 2 4 7 100

【输出样例】

7

8

15

一种方法是０１背包，参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48 | //货币面值 -01背包  #include <iostream>  #include <cstdlib>  using namespace std;    int N,MAX;  int value[101],dp[10001];    int Compare(const void \* p, const void \* q)  {  return \*(int \*)p - \*(int \*)q;  }    int Max(int a, int b)  {  return (a > b) ? a : b;  }    int ZeroOnePack()  {  int i, j;  memset(dp, 0, sizeof(dp));  for (i = 1; i <= N; ++i)  for (j = MAX; j >= value[i]; --j)  dp[j] = Max(dp[j], dp[j-value[i]] + value[i]);  for (i = 1; i <= MAX; ++i)  if (dp[i] != i)  return i;  }    int main()  {  freopen("Currency.in","r",stdin);  freopen("Currency.out","w",stdout);  int i;  while (scanf("%d", &N) != EOF)  {  MAX = 0;  for (i = 1; i <= N; ++i)  {  scanf("%d", &value[i]);  MAX += value[i];  }  qsort(value, N, sizeof(int), Compare);  cout<<ZeroOnePack()<<endl;  }  return 0;  } |

使用动态规划的思想, 对于从第1个到第i个数的和total，如果第i+1个数大于total+1则不会组成total+1。

具体来讲，就是计算已排好序的数据，计算到第i个的时候，total记录的是前面i – 1个数的和。那么第i个数如果大于total + 1，而第i个数后面的数已知都比第i个数大，都不可能构成total + 1，那么对于读到的这个i，显然total + 1是最小的无法构成的数。

再来看更普遍的：其实对于一个范围内的数，如果要用最少的数构成所有这个范围内的数，只要满足得到的数都是关于2的幂次,比如要构成16以内所有的数，只用1，2，4，8便足够了，这是很容易证明的，因为所有的数都可以表示成若干个2次幂数的和，其中构成数的最大2次幂数一定小于这个数，所以得证。

知道了以上这个规律，再观察2次幂数的特点， 对于1，2，4，8，16 这样的数列，会发现前i – 1项的和都恰好比第i项小1，那么，对于一个有序数列，如果满足上述这个规律，或者更普遍点，不用正好小1，小更多，只要是前提已经是有序，那么都会有这个数列的特性，即构成value[i]范围内所有的数，最小不能构成的数就是total + 1。

参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | //货币面值－动规２  #include <stdio.h>  #include <stdlib.h>  int Compare(const void \* p, const void \* q)  {  return \*(int \*)p - \*(int \*)q;  }  int main()  {  freopen("Currency.in","r",stdin);  freopen("Currency.out","w",stdout);  int N,i,ans,value[100];  while (scanf("%d", &N) != EOF)  {  for (i = 0; i < N; ++i)  scanf("%d", &value[i]);  qsort(value, N, sizeof(int), Compare);  ans = 0;  for (i = 0; i < N; ++i)  if (value[i] > ans + 1)  break;  else  ans += value[i];  printf("%d\n", ans + 1);  }  return 0;  } |

## 收益

【问题描述】收益（Profit.cpp/c/pas) POJ 2063

“建太空梯进入太空要1兆亿？”魔法学院的院长瞪大了眼睛。

“这只是基础设施的费用，后期还要......”墨老师掰着手指算。

“哎呀，现在地主也很穷啊，学院的钱批下来就这么多，你想办法用这笔钱在债券市场上获得最大收益吧。”院长皱着眉头。

简单来说，就是你有一笔钱，你要将这笔钱去投资债券，现在有d种债券，每种债券都有一个价值和年收益，债券的价值是1000的倍数，问你如何投资在n年后的获得最大收益。

【输入格式】

第一个为一个整数M，表示有M组数据。

每组数据第一行有两个整数，表示初始资金(不超过50000)和年数n。

每组数据第二行为一个整数d(1 <= d <= 10)，表示债券种类。

随后d行每行有两个整数，表示该债券的价值和年收益。年收益不会超过债券价值的10%。

所有数据不超过整型取值范围。

【输出格式】

每组数据，输出n年后获得的最大收益。

【输入样例】

1

10000 4

2

4000 400

3000 250

【输出样例】

14050

【算法分析】

完全背包问题，因为每种股票可以无限投资，而问n年之后的最大收益，我们将每一年的最大收益计算，将前i-1年的现有收益作为现在投资金额再次对第i年进行一次投资。

此外，由于债券的价值是1000的倍数，所以在算完全背包的时候，可以把总的钱除于1000，把每种债券的价格也除于1000，这样可以减少运算时间。但要注意原来的钱不一定是1000的整数倍的。

参考代码如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | //收益  #include <iostream>  using namespace std;  #define MAXV 50000  #define max(a,b) a>b?a:b  int main()  {  freopen("Profit.in","r",stdin);  freopen("Profit.out","w",stdout);  int dp[MAXV];  int v[11],c[11];  int n,m,year,t,i,j,sum,k;  scanf("%d",&n);  while(n--)  {  scanf("%d%d%d",&m,&year,&t);  for(i=1;i<=t;i++)  {  scanf("%d%d",&v[i],&c[i]);  v[i]=v[i]/1000;//因为价值是1000的倍数，所以除以1000  }  sum=0;  for(i=0;i<year;i++)//因为每年都要投资，所以每年都要计算  {  memset(dp,0,sizeof(dp)); //对数组进行初始化  sum=m/1000; //对现在第i年投资前，拥有多少钱  for(j=1;j<=t;j++) //完全背包DP，当今年投资在前j个债券的价值最大DP[k]  {  for(k=0;k<=sum;k++)  if(k>=v[j])  dp[k]=max(dp[k],dp[k-v[j]]+c[j]);  }  m+=dp[sum];//将今年赚到的作为下一年的初始资金  }  printf("%d\n",m);  }  return 0;  } |

## 均分魔法石

【问题描述】均分魔法石（dividing.cpp/c/pas）HDU 1059

张琪曼和李旭琳收集了很多魔法石，现在她们想把这些魔法石分开以装入太空防御系统的两个能量槽，要求是两个能量槽的魔法石能量必须要相等而且数量也必须相等。显然如果所有的魔法石都有相同的价值，那么分成均等的两份将很容易，但是不幸的是，魔法石的能量各不相同，其价值评估分别为1、2、3、4、5、6，另外如果魔法石数是奇数将分不开，即便是偶数，也不一定能分开，例如，有一个价值为1的，一个价值为3的，两个价值为4的，就不能分成相等的两部分，因此需要你写一个程序判断能不能将这些魔法石分成价值相等的两部分。

【输入格式】

有多组数据，每组数据每行包括6个非负数的整数，n1, n2, ..., n6，ni代表价值为i的魔法石有ni个，个数可能最大达到20000。

【输出格式】

每组数据输出“Collection #k:”，k是第几个样例数，然后输出“Can be divided.”或“Can't be divided.”。

【输入样例】

1 0 1 2 0 0

1 0 0 0 1 1

0 0 0 0 0 0

【输出样例】

Collection #1:

Can't be divided.

Collection #2:

Can be divided.

【算法分析】

可以转成01背包问题，也可以多重背包+二进制优化。

所谓二进制优化，简单的说一个数可以由一系列全是2的倍数或者 2的倍数加上一个非2的倍数组成。即有价值为i的物品有n个，可以把n拆成几个物品，分别是1，2，4，…，2(k-1)，num[i]-2k+1，且k是满足num[i]-2k+1>0的最大整数。例如num[13]可以拆成13=1+2+4+6, num[14]可以拆成1，2，4，7四件物品。

假设3本来是1+1+1，但是3还可以拆成1+2，这样时间复杂度由O(V×Σn[i])降到O(V×Σlog n[i])。

参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60 | //均分魔法石  #include <stdio.h>  #include <string.h>  #include <iostream>  using namespace std;  #define CLR(a,v) memset(a,v,sizeof(a))  int num[7],dp[60010];  int dig[]={1,2,4,8,16,32,64,128,256,512,1024,2048,4096,8192,16384,32768};  int main()  {  freopen("dividing.in","r",stdin);  freopen("dividing.out","w",stdout);  int v,ncase = 0;  while(1)  {  int sum = 0;  for(int i=1;i<=6;i++)  {  scanf("%d",&num[i]);  sum += i\*num[i];  }  if(sum == 0)  break;  printf("Collection #%d:\n",++ncase);  if(sum&1)  {  printf("Can't be divided.\n\n");  continue;  }  CLR(dp,0);  sum /= 2;  for(int i=1;i<=6;i++)  {  for(int d=0;num[i];d++)  {  if(num[i] >= dig[d])  {  v = i\*dig[d];  num[i] -= dig[d];  }  else  {  v = i\*num[i];  num[i] = 0;  }  for(int j=sum;j>=v;j--)  dp[j] = max(dp[j],dp[j-v]+v);  }  }  if(dp[sum] == sum)  printf("Can be divided.\n\n");  else  printf("Can't be divided.\n\n");  }  return 0;  } |

## 硬币问题

【问题描述】硬币问题(coin.cpp/c/pas)

给你n种硬币，知道每种的面值Ai和每种的数量Ci。问能凑出多少种不大于m的面值。

【输入格式】

有多组数据，每一组第一行有两个整数 n(1<=n<=100)和m(m<=100000)，第二行有2n个整数，即面值A1，A2，A3...An和数量C1，C2，C3，…，Cn (1<=Ai<=100000，1<=Ci<=1000)。所有数据结束以2个0表示。

【输出格式】

每组数据输出一行答案。

【输入样例】

3 10

1 2 4 2 1 1

2 5

1 4 2 1

0 0

【输出样例】

8

4

【算法分析】

把m当做背包，dp[i]代表要买价值为i的物品是否能恰好用硬币买到。

如果Ａ[i]×Ｃ[i]>=m，则在背包容量m内，第i个硬币可以取任意n个(n×a[i]<=m)都不会超过Ｃ[i]个，所以可以看成完全背包。

如果Ａ[i] ×Ｃ[i]<m，则在背包容量m内，第i个硬币最多可以取Ｃ[i]个,所以可以看成是多重背包。这样可以二进制优化，转为０１背包问题。

参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65 | //硬币问题－混合背包  #include<iostream>  #include<cstdlib>  #include<cstring>  #define INF 99999999  using namespace std;  const int MAX=100002;  bool dp[MAX];  int a[102],c[102];//a代表第i个硬币的价值,c代表个数.  int main()  {  freopen("coin.in","r",stdin);  freopen("coin.out","w",stdout);  int n,m;  while(cin>>n>>m,n||m)  {  for(int i=0;i<n;++i)  cin>>a[i];  for(int i=0;i<n;++i)  cin>>c[i];  if(m<=0)//会出现m为负数  {  cout<<"0\n";  continue;  }  memset(dp,false,sizeof(bool)\*(m+1));  dp[0]=true;  for(int i=0;i<n;++i)  {  if(a[i]\*c[i]>=m)//完全背包  {  for(int j=a[i];j<=m;++j)  {  if(dp[j-a[i]])  dp[j]=true;  }  }  else//多重背包  {  int k=0;  for(k=1;k\*2-1<=c[i];k=k\*2)//二进制优化，以01背包来做.  {  for(int j=m;j>=k\*a[i];--j)  {  if(dp[j-k\*a[i]])  dp[j]=true;  }  }  for(int j=m;j>=(c[i]-k+1)\*a[i];--j)  {  if(dp[j-(c[i]-k+1)\*a[i]])  dp[j]=true;  }  }  }  int sum=0;  for(int i=1;i<=m;++i)  if(dp[i])  sum++;  cout<<sum<<endl;  }  return 0;  } |

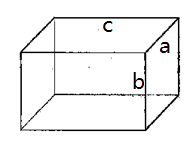
使用多重背包的参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51 | //硬币问题－多重背包  #include<iostream>  #include<cstdlib>  #include<cstring>  #define INF 99999999  using namespace std;  const int MAX=100002;  bool dp[MAX];  int a[102],c[102];//a代表第i个硬币的价值,c代表个数  int main()  {  int n,m,k;  while(cin>>n>>m,n||m)  {  for(int i=0;i<n;++i)  cin>>a[i];  for(int i=0;i<n;++i)  cin>>c[i];  if(m<=0)  {  cout<<"0\n";  continue;  }  memset(dp,false,sizeof(bool)\*(m+1));  dp[0]=true;  for(int i=0;i<n;++i)  {  for(k=1;k\*2-1<=c[i];k=k\*2)//二进制优化，以０１背包来做  {  for(int j=m;j>=k\*a[i];--j)  {  if(dp[j-k\*a[i]])  dp[j]=true;  }  }  for(int j=m;j>=(c[i]-k+1)\*a[i];--j)  {  if(dp[j-(c[i]-k+1)\*a[i]])  dp[j]=true;  }  }  int sum=0;  for(int i=1;i<=m;++i)  if(dp[i])  sum++;  cout<<sum<<endl;  }  return 0;  } |

## 积木游戏

【题目描述】积木游戏(juggle.cpp/c/pas)

太空梯的建造计划从一开始就引来了魔法界各派人物喋喋不休的争吵。各派从学术观点、历史观点、美学观点等各方面引经据典，争论的面红耳赤。后来，为了检验各派理论的正确性，项目主持者设计了一种积木游戏。每个游戏者有N块编号依次为1 ，2，…，N的长方体积木。对于每块积木，它的三条不同的边分别称为”a边”、“b边”和”c边”，如图所示。



图

游戏规则如下：

1、从N块积木中选出若干块，并将它们分成M（l<=M<=N） 堆，称为第1堆，第2 堆…，第M堆。每堆至少有1块积木，并且第K堆中任意一块积木的编号要大于第K+1堆中任意一块积木的编号(2<=K<=M)。

2、对于每一堆积木，游戏者要将它们垂直摞成一根柱子，并要求满足下面两个条件：

(1)除最顶上的一块积木外，任意一块积木的上表面同且仅同另一块积木的下表面接触，并且要求下面的积木的上表面能包含上面的积木的下表面，也就是说，要求下面的积木的上表面的两对边的长度分别大于等于上面的积木的两对边的长度。

(2)对于任意两块上下表面相接触的积木，下面的积木的编号要小于上面的积木的编号。

最后，根据每人所摞成的M根柱子的高度之和来决出胜负。

请你编一程序，寻找一种摞积木的方案，使得你所摞成的M根柱子的高度之和最大。

【输入格式】

文件的第一行有两个正整数N和M（1<=M<=N<=100），分别表示积木总数和要求摞成的柱子数。这两个数之间用一个空格符隔开。接下来N行依次是编号从1到N的N个积木的尺寸，每行有三个；至1000之间的整数，分别表示该积木a边，b边和c边的长度。同一行相邻两个数之间用一个空格符隔开。

【输出格式】

只有一行，为一个整数，表示M根柱子的高度之和。

【输入样例】

4 2

10 5 5

8 7 7

2 2 2

6 6 6

【输出样例】

24

【算法分析】

设dp[k][i][j][s] 表示在前 i 个积木分成 k 堆，并且第 k 堆最上面的积木编号为 j，j 的 s 面朝上（s = 0, 1, 2）;

则第 i 个积木有 3 种决策：

（１） dp[k][i][i][t] = min(dp[k][i-1][j][s]); 放在第 k 堆；

（２） dp[k][i][i][t] = min(dp[k-1][i-1][j][s]); 另起一堆；

（３） dp[k][i][j][s] = dp[k][i-1][j][s]; 不用；

由于摆在上面的边要小于等于摆在下面积木的边，所以事先可以对 a, b, c 边从小到大排序下，判断的时候更加方便。

参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62 | //积木游戏  #include <iostream>  #include <cstdlib>  using namespace std;  const int MAXN = 110;  int dp[MAXN][MAXN][MAXN][3];  int a[MAXN], b[MAXN], c[MAXN];// 0 bc, 1 ac, 2 ab  int main()  {  freopen("juggle.in","r",stdin);  freopen("juggle.out","w",stdout);  int n, m;  scanf("%d%d", &n, &m);  for (int i = 1; i <= n; i++)  {  cin>>a[i]>>b[i]>>c[i];  if (a[i] > b[i])  swap(a[i], b[i]);  if (b[i] > c[i])  swap(b[i], c[i]);  if (a[i] > b[i])  swap(a[i], b[i]);  }  //t, s 控制上、下积木朝向。每次交换边的值，只让 b, c 比较即可  for (int k = 1; k <= m; k++)  for (int i = 1; i <= n; i++)  for (int j = 0; j < i; j++)  for (int t = 0; t <= 2; t++)  {  if (t == 1)  swap(b[i], a[i]);  if (t == 2)  swap(c[i], b[i]), swap(a[i], b[i]);  for (int s = 0; s <= 2; s++)  {  if (s == 1)  swap(b[j], a[j]);  if (s == 2)  swap(c[j], b[j]), swap(a[j], b[j]);  if (b[i] <= b[j] && c[i] <= c[j])  dp[k][i][i][t] = max(dp[k][i][i][t], dp[k][i-1][j][s] + a[i]);  if (s == 2)  swap(a[j], b[j]), swap(b[j], c[j]);  if (s == 1)  swap(b[j], a[j]);  dp[k][i][i][t] = max(dp[k][i][i][t], dp[k-1][i-1][j][s] + a[i]);  dp[k][i][j][s] = dp[k][i-1][j][s];  }  if (t == 2)  swap(a[i], b[i]), swap(c[i], b[i]);  if (t == 1)  swap(b[i], a[i]);  }  int ans = 0;  for (int j = 1; j <= n; j++)  for (int s = 0; s <= 2; s++)  ans = max(ans, dp[m][n][j][s]);  cout<<ans<<endl;  return 0;  } |

## 花店橱窗设计

【题目描述】花店橱窗设计（shopwindow.cpp/c/pas）　ioi 99

魔法世界的人们对艺术的追求是无止境的，我们可以从路边花店的橱窗设计就可以看出来。假设以最美观的方式布置花店的橱窗，有F束花，每束花的品种都不一样，同时，至少有同样数量的花瓶，被按顺序摆成一行，花瓶的位置是固定的，并从左到右，从1到V顺序编号，V是花瓶的数目，编号为1的花瓶在最左边，编号为V的花瓶在最右边，花束可以移动，并且每束花用1到F的整数惟一标识，标识花束的整数决定了花束在花瓶中列的顺序，即如果I〈J，则花束I必须放在花束J左边的花瓶中。例如，假设杜鹃花的标识数为1，秋海棠的标识数为2，康乃馨的标识数为3，所有的花束在放入花瓶时必须保持其标识数的顺序，即：杜鹃花必须放在秋海棠左边的花瓶中，秋海棠必须放在康乃馨左边的花瓶中。如果花瓶的数目大于花的数目，则多余的花瓶必须空，即每个花瓶中只能放一束花。 每一个花瓶的形状和颜色也不相同，因此，当各个花瓶中放入不同的花束时，会产生不同的美学效果，并以美学值（一个整数）来表示，空置花瓶的美学值为0。在上述例子中，花瓶与花束的不同搭配所具有的美学值，可以用如下表格表示：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 花瓶1 | 花瓶2 | 花瓶3 | 花瓶4 | 花瓶5 |
| 杜鹃花 | 7 | 23 | -5 | -24 | 16 |
| 秋海棠 | 5 | 21 | -4 | 10 | 23 |
| 康乃馨 | -21 | 5 | -4 | -20 | 20 |

根据表格，杜鹃花放在花瓶2中，会显得非常好看，但若放在花瓶4中则显得很难看。 为取得最佳美学效果，必须在保持花束顺序的前提下，使花的摆放取得最大的美学值，如果具有最大美学值的摆放不止一种，则输出任何一种方案即可。题中数据满足下面条件：1〈=F〈=100，F〈=V〈=100，-50〈=Vij〈=50，其中Vij是花束I摆放在花瓶J中的美学值。输入整数F，V和矩阵（Vij），输出最大美学值和每束花摆放在各个花瓶中的花瓶编号。

【输入格式】　 第一行包含两个数：F，V。 随后的F 行中，每行包含V 个整数，Aij 即为输入文件中第（i+1 ）行中的第j 个数　　【输出格式】　　包含两行:第一行是程序所产生摆放方式的美学值。第二行必须用F 个数表示摆放方式，即该行的第K个数表示花束K所在的花瓶的编号。　　【输入样例】

　　3 5

　　7 23 –5 –24 16

　　5 21 -4 10 23

　　-21 5 -4 -20 20

　　【输出样例】

　　53

2 4 5

【算法分析】

设f[i][j]为将第i束花插进第j个花瓶所能获得的最大的分，则动态转移方程是:

f[i][j] = max(f[i - 1][k] + map[i][j]) (i -1=<k<j)

参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60 | //花店橱窗设计  #include <stdio.h>  #include <stdlib.h>  int f[101][101];  int map[101][101];  int p[101][101];  void output(int a, int b)  {  if(a == 1)  {  printf("%d", b);  return;  }  output(a - 1, p[a][b]);  printf(" %d", b);  }  int main(int argc, char \*\*argv)  {  freopen("flower.in","r",stdin);  freopen("flower.out","w",stdout);  int i, j, k;  int m, n;  scanf("%d%d", &n, &m);  for(i = 1; i <= n; i++)  {  for(j = 1; j <= m; j++)  {  scanf("%d", &map[i][j]);  }  }  for(i = 1; i <= n; i++)  {  for(j = i; j <= m - n + i; j++)  {  f[i][j] = -100000000;  for(k = i - 1; k < j; k++)  {  if(f[i - 1][k] + map[i][j] > f[i][j])  {  f[i][j] = f[i - 1][k] + map[i][j];  p[i][j] = k;  }  }  }  }  for(i = n, k = 0; i <= m; i++)  {  if(k < f[n][i])  {  k = f[n][i];  j = i;  }  }  printf("%d\n", f[n][j]);  output(n, j);  printf("\n");  return 0;  } |

## 选数统计

　　【题目描述】选数统计（ChoiceNum.cpp/c/pas）

现在从1 ~ M里可以让你从小到大选出n个数，设这些数为A1到An，要求每个数至少为它前一项的两倍，比如说当M = 10，n = 4时，下面是几种可能的选法：

1 2 4 8

1 2 4 9

1 2 4 10

1 2 5 10

求一共可以选出多少不同的数列？

【输入格式】

仅有一行，表示n (n <= 8)和M (M <= 500)

【输出格式】

一个数表示方案数。

【输入样例】

4 10

【输出样例】

4

【算法分析】

设a [ k ] [ i ] 代表长度为k时第k位是 i 的序列种数，那么 a [ k ] [ i ] = ∑ a [ k-1 ] [ j -1 ] ( j <= i )

边界条件：a[ 1 ] [ i ]＝１

最后将所有的 a[ N ] [ i ] 相加则是总和。时间复杂度是 O ( N × M2 )

参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | //选数统计  #include<iostream>  #include <stdlib.h>  using namespace std;  int T,k,i,j,t,N,M,p;  long long a[11][2001],ans;  int main()  {  freopen("ChoiceNum.in","r",stdin);  freopen("ChoiceNum.out","w",stdout);  scanf("%d%d",&N,&M);  memset(a,0,sizeof(a));  for (i=1;i<=M;i++)  a[1][i]=1;  p=1;  for (k=2;k<=N;k++)  {  for (i=p\*2;i<=M;i++)  for (j=p;j<=i/2;j++)  a[k][i]+=a[k-1][j];  p\*=2;  }  ans=0;  for (i=p;i<=M;i++)  ans+=a[N][i];  printf("%I64d\n",ans);  return 0;  } |

## 胜利大逃亡

【题目描述】胜利大逃亡（escape.cpp/c/pas）

天顶星人设下了无数的陷阱，魔法世界的一支侦察小分队就落到了一个迷宫陷阱之中，现在，小分队要从一个m列n行的迷宫中逃走，有p个石柱，另有b只吸血蝙蝠。

　　蝙蝠分三种类型：

　　１.当蝙蝠前方遇到石柱或墙，向左转。

２.当蝙蝠前方遇到石柱或墙，向后转。

３.当蝙蝠前方遇到石柱或墙，向右转。

　　人可以向上、下、左、右四个方向移动或在原位置等待，人运动一格或等待，都花费一个单位时间，在一个单位时间内，蝙蝠和人可以同时移动，蝙蝠也可以向上、下、左、右四个方向移动或旋转，注意旋转不花时间，也就是说蝙蝠和人都可以先旋转再移动。同一时刻，人和蝙蝠在同一地点时，人死亡。人和蝙蝠可以互相穿过，且蝙蝠可以重叠。

　　现在从（1，1）逃到（m，n）,仅有（m+n-1）单位时间，问在该时间内共有几种逃亡路线。

　　注：在第1个时刻，人进入（1，1），蝙蝠赋初始状态，在第（m+n-1）时刻，人要到达（m，n）。

　　若初始状态中蝙蝠与石柱重合，则认为蝙蝠在石柱上休息，不会动。

【输入格式】

输入文件第1行为m，n，第2行为石柱个数p，以下p行，每行两个整数，分别为石柱的横、纵坐标，第p+3行，为蝙蝠数b，以下b行，每行四个整数，分别为蝙蝠的横、纵坐标，蝙蝠方向d(上为1，左为2，下为3，右为4)，以及蝙蝠类型t用1，2，3表示。

2<=m<=100;2<=n<=100;0<=p<=100;0<=b<=100

【输出格式】

仅一行，为逃脱的方法数，若无解则输出0。

【输入样例】

3 3

1

1 2

1

2 3 1 1

【输出样例】

3

【算法分析】

这是一道比较特殊的动态规划问题，也是过河卒问题的加深，由于时间限制，人必须保证每一时刻都向右或向下街，因此在每一时刻，人所能到的点是固定的。但由于有了石柱和蝙蝠，这些位置人是无法到达的。

由于蝙蝠是不断运动，因此我们在划分阶段的时候应采用时刻作为阶段，按斜线划分。因此，对于迷宫中的每一个点，如果这个点没有柱子且在人到达这个点的时刻这个点上没有蝙蝠，可以到达这点的方法数就应为可到达它上方的方法数加上可到达它左方的方法数之和，否则为0。

本题还由于蝙蝠旋转时不花时间且蝙蝠一开始还可能在石柱上，因此还需考虑许多特殊情况。

设g[i，j]表示石柱位置；然后对每个蝙蝠进行动作模拟，算出每一时间蝙蝠的位置用c[j，x，y]=true来表示第j时刻在（x，y）处是否有蝙蝠，这个过程比较麻烦，O（B（m+n））时间可以完成，然后开始O(mn)的DP。

设f[I,j]表示到第i行j列得到的路线数目，则状态转移方程为：

f[i，j]=0 当c[(i+j-2)，i，j] or not g[i，j]

否则 f[i，j]=f[i-1，j]+f[i，j-1]

参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87 | //胜利大逃亡  #include <cstdio>  #include <string>  #include <cstdlib>  const long dx[] = {-1,0,1,0};  const long dy[] = {0,-1,0,1};  bool bat[120][120][300];  char map[120][120];  long f[120][120];  void swtch(long &d,long t)  {  if (t == 3)  d = (d+3)%4;  else if (t == 2)  d = (d+2)%4;  else if (t == 1)  d = (d+1)%4;  }  long getint()  {  long rs=0;bool sgn=1;char tmp;  do tmp = getchar();  while (!isdigit(tmp)&&tmp-'-');  if (tmp == '-')  {  tmp=getchar();  sgn=0;  }  do rs=(rs<<3)+(rs<<1)+tmp-'0';  while (isdigit(tmp=getchar()));  return sgn?rs:-rs;  }  int main()  {  freopen("escape.in","r",stdin);  freopen("escape.out","w",stdout);  long m = getint();  long n = getint();  long T = m+n-1;  for (long i=0;i<n+2;i++)  map[i][0] = map[i][m+1] = 2;  for (long i=0;i<m+2;i++)  map[0][i] = map[n+1][i] = 2;  long p = getint();  for (long i=1;i<p+1;i++)  {  long y = getint();  long x = getint();  map[x][y] = 1;  }  long b = getint();  for (long j=1;j<b+1;j++)  {  long y0 = getint();  long x0 = getint();  long d = getint()-1;  long t = getint();  long x = x0;  long y = y0;  if (x<1||y<1||x>n||y>m||map[x][y]>0)  continue;  bat[x][y][1] = true;  for (long i=1;i<T;i++)  {  long tt;  for (tt=0;tt<4&&map[x+dx[d]][y+dy[d]]>0;++tt)  swtch(d,t);  if (tt < 4)  x += dx[d];y += dy[d];  bat[x][y][i+1] = true;  }  }  f[1][1] = 1;  for (long i=1;i<n+1;i++)  for (long j=1;j<m+1;j++)  if (!bat[i][j][i+j-1]&&map[i][j]==0&&f[i][j]==0)  f[i][j] = f[i-1][j]+f[i][j-1];  printf("%ld",f[n][m]);  return 0;  } |

## 天上掉馅饼

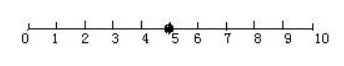
【题目描述】天上掉馅饼（pie.cpp/c/pas）HDOJ 1176

墨老师：“大家真的很辛苦，我会送给大家一个奖励的......”

众学生一脸不信：“老师你不是又在骗我吧，天上怎么会掉馅饼呢？”

墨老师满头大汗：“我这次说的绝对是真的，其实只要你们信，天上也是会掉馅饼的。”

如图所示，假设在接下来的一段时间里，所有的馅饼都掉落在0-10这11个位置。玩家每秒种只能在移动不超过一米的范围内接住坠落的馅饼（知道天天宅在家里玩游戏不运动的严重后果了吧）。



图

开始时游戏者站在5这个位置，因此在第一秒，他只能接到4，5，6这三个位置中其中一个位置上的馅饼。问最多可能接到多少个馅饼？

【输入格式】

输入数据有多组。每组数据的第一行为以正整数n(0<n<100000)，表示有n个馅饼掉下来。在结下来的n行中，每行有两个整数x，T(0<T<100000),表示在第T秒有一个馅饼掉在x点上。同一秒钟在同一点上可能掉下多个馅饼。n=0时输入结束。

【输出格式】

每一组输入数据对应一行输出。输出一个整数m，表示最多可能接到m个馅饼。

【输入样例】

6

5 1

4 1

6 1

7 2

7 2

8 3

0

【输出样例】

４

【算法分析】

设dp[i][j]表示i时刻在坐标j出最多能接到的馅饼数。则状态转移方程为：

dp[i][j] = Max(dp[i+1][j-1], dp[i+1][j], dp[i+1][j+1]) + dp[i][j]

dp[0][5]即为所求结果。

参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55 | //天上掉馅饼  #include <iostream>  #include <stdlib.h>  #include <math.h>  using namespace std;  #define MAX 100001  int dp[MAX][12];//dp[i][j]表示在i时刻落在j点的馅饼数量  int Max2(int a, int b)  {  return (a > b) ? a : b;  }  int Max3(int a, int b, int c)  {  int max = (a > b) ? a : b;  return (max > c) ? max : c;  }  int work(int max\_time)  {  int i, j, max;  for (i = max\_time - 1; i >= 0; --i)  {  dp[i][0] = Max2(dp[i+1][0], dp[i+1][1]) + dp[i][0];  for (j = 1; j < 10; ++j)  {  dp[i][j] = Max3(dp[i+1][j-1],dp[i+1][j],dp[i+1][j+1])+dp[i][j];  }  dp[i][10] = Max2(dp[i+1][10], dp[i+1][9]) + dp[i][10];  }  return dp[0][5];  }  int main()  {  freopen("pie.in","r",stdin);  freopen("pie.out","w",stdout);  int n,i,time,location,max\_time;  while (scanf("%d", &n) != EOF && n != 0)  {  memset(dp, 0, sizeof(dp));  max\_time = -1;  for (i=1; i<=n; ++i)  {  scanf ("%d%d", &location, &time);  ++dp[time][location];  if (max\_time < time)  max\_time = time;  }  printf ("%d\n", work(max\_time));  }  return 0;  } |

## 擦除游戏

【题目描述】擦除游戏

张琪曼喜欢和李旭琳玩这样一个游戏，首先她们在地上写出两个字符串，然后张琪曼指出两个字符串中的同一个字符，然后她得到这个字符所代表的一个奖励分数，这时李旭琳会擦除两个字符以及它们前面的所有字符，直到最后张琪曼找不出了为止。比如首先两个字符串分别为”abc”和”cab”，然后假设擦除任意一个字符的奖励分都为1，首先张琪曼会指出第一个串的第1个字符和第二个串的第2个字符a，然后擦除后两个串变成”bc”和”b”，然后她会指出b，然后游戏结束，她得两分。现在张琪曼想知道她能得到最多的奖励分是多少？

【输入格式】

第一行一个数字n(n <= 26)，表示串一共可能出现的字母数。

接下来n行，每行一个字母和一个数字，表示擦除该字母得到的奖励分。

接下来两行，表示游戏开始的字符串，字符串只会包含给出的字母。(长度 <= 100)

【输出格式】

一个数表示最大奖励分。

【输入样例】

3

a 1

b 10

c 100

abc

cab

【输出样例】

100

【算法分析】

一道求最长公共子串的问题，只不过字串的各个字符带上了权值。我们可以将一个字符的长度等于其权值，将其存储在一个辅助数组上。经过这样的转化，就是一道求最长字串的问题。

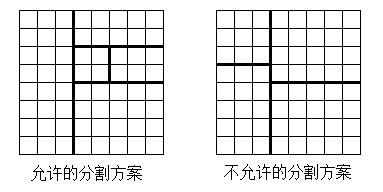
参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41 | //擦除游戏  #include <iostream>  #include <string>  #include <cstdlib>  using namespace std;  const int MAXN=110;  char A[MAXN], B[MAXN], ch;  int F[MAXN][MAXN], s[256];  int N;  int max (int a, int b)  {  return a > b ? a : b;  }  int main()  {  freopen ("erase.in", "r", stdin);  freopen ("erase.out", "w", stdout);  scanf ("%d\n", &N);  for (int i = 0; i < N; ++i)  {  scanf ("%c", &ch);  scanf ("%d\n", &s[ch]);  }  scanf ("%s\n", A);  scanf ("%s\n", B);  memset (F, 0, sizeof (F));  for (int i = 1; i <= strlen (A); ++i)  for (int j = 1; j <= strlen (B); ++j)  {  F[i][j] = max (F[i - 1][j], F[i][j - 1]);  if (A[i - 1] == B[j - 1])  F[i][j] = max (F[i][j], F[i - 1][j - 1] + s[A[i - 1]]);  }  printf ("%d\n", F[strlen (A)][strlen (B)]);  return 0;  } |

## 棋盘分割

【题目描述】棋盘分割（chessboard.cpp/c/pas）NOI 99

楚继光将一个８×８的棋盘进行如下分割：将原棋盘割下一块矩形棋盘并使剩下部分也是矩形，再将剩下的部分继续如此分割，这样割了(n-1)次后，连同最后剩下的矩形棋盘共有n块矩形棋盘。(每次切割都只能沿着棋盘格子的边进行)



图

原棋盘上每一格有一个分值，一块矩形棋盘的总分为其所含各格分值之和。现在需要把棋盘按上述规则分割成n块矩形棋盘，并使各矩形棋盘总分的均方差σ最小。

均方差σ，其中平均值，xi为第i块矩形棋盘的分。

请编程对给出的棋盘及n，求出σ的最小值。

【输入格式】

第1行为一个整数n(1<n<15)。

第2行至第9行每行为8个小于100的非负整数，表示棋盘上相应格子的分值。每行相邻两数之间用一个空格分隔。

【输出格式】

仅一个数，为σ（四舍五入精确到小数点后三位

【输入样例】

3

1 1 1 1 1 1 1 3

1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1

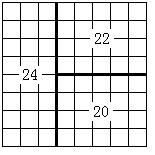
1 1 1 1 1 1 1 0

1 1 1 1 1 1 0 3

【输出样例】

1.633

【样例说明】



【算法分析】

将均方差公式变形，设x'表示最后矩形棋盘分值的平均值，可知平均值是一定的（它等于方格里的数和除以n）。

又由均方差公式推出σ2 ＝

可知只需要让每个矩形的总分的平方和尽量小即可以了。

可知若第i次分割为最佳分割，则第i－1次分割为且必为最佳，用f[k][x1][y1][x2][y2]作为状态，表示左上角坐标为(x1，y1)，右下角坐标为(x2，y2)的还要被切成k块的矩形中，总分和的平方的最小值。注意每次切割都会得到一块最终需要的矩形，每次都有横竖两个方向选择，则动态转移方程为：

水平切时，

dp[k][x1][y1][x2][y2]

=min(dp[k][x1][y1][x2][y2],dp[1][x1][y1][t][y2]+dp[k-1][t+1][y1][x2][y2])，

dp[k][x1][y1][x2][y2]

=min(dp[k][x1][y1][x2][y2],dp[k-1][x1][y1][t][y2]+dp[1][t+1][y1][x2][y2])。

纵向切时，

dp[k][x1][y1][x2][y2]

=min(dp[k][x1][y1][x2][y2],dp[1][x1][y1][x2][t]+dp[k-1][x1][t+1][x2][y2])，

dp[k][x1][y1][x2][y2]

=min(dp[k][x1][y1][x2][y2],dp[k-1][x1][y1][x2][t]+dp[1][x1][t+1][x2][y2])。

参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75 | //棋盘分割  #include <iostream>  #include <cstdlib>  #include <cmath>  #define N 20  #define INF 0x7ffffff  #define M 8  using namespace std;    int sum[N][N][N][N],dp[N][N][N][N][N];  int a[N][N];  int main()  {  freopen("chessboard.in","r",stdin);  freopen("chessboard.out","w",stdout);  int n;  while(scanf("%d",&n)!=EOF)  {  for(int i=0;i<=M-1;i++)  for(int j=0;j<=M-1;j++)  scanf("%d",&a[i][j]);    for(int x1=0;x1<=M-1;x1++)  for(int y1=0;y1<=M-1;y1++)  for(int x2=x1;x2<=M-1;x2++)  for(int y2=y1;y2<=M-1;y2++)  {  if(y2==y1&&x2==x1)  sum[x1][y1][x2][y2] = a[x1][y1];  else if(x2==x1)  sum[x1][y1][x2][y2]  = sum[x1][y1][x2][y2-1]+a[x2][y2];  else if(y2==y1)  sum[x1][y1][x2][y2]  = sum[x1][y1][x2-1][y2] + a[x2][y2];  else  sum[x1][y1][x2][y2]=sum[x1][y1][x2][y2-1]  +sum[x1][y1][x2-1][y2]-sum[x1][y1][x2-1][y2-1]+a[x2][y2];  dp[1][x1][y1][x2][y2]  = sum[x1][y1][x2][y2]\*sum[x1][y1][x2][y2];  }  for(int k=2;k<=n;k++)  for(int x1=0;x1<=M-1;x1++)  for(int y1=0;y1<=M-1;y1++)  for(int x2=x1;x2<=M-1;x2++)  for(int y2=y1;y2<=M-1;y2++)  {  dp[k][x1][y1][x2][y2]=INF;  for(int t = x1;t<=x2-1;t++) //hor  {  dp[k][x1][y1][x2][y2]=min(dp[k][x1][y1][x2][y2],  dp[1][x1][y1][t][y2]+dp[k-1][t+1][y1][x2][y2]);  dp[k][x1][y1][x2][y2]=min(dp[k][x1][y1][x2][y2],  dp[k-1][x1][y1][t][y2]+dp[1][t+1][y1][x2][y2]);  }  for(int t = y1;t<=y2-1;t++) //ver  {  dp[k][x1][y1][x2][y2]=min(dp[k][x1][y1][x2][y2],  dp[1][x1][y1][x2][t]+dp[k-1][x1][t+1][x2][y2]);  dp[k][x1][y1][x2][y2]=min(dp[k][x1][y1][x2][y2],  dp[k-1][x1][y1][x2][t]+dp[1][x1][t+1][x2][y2]);  }  }  double temp = (double)(sum[0][0][M-1][M-1])/(double)(n);  double res1 = temp\*temp;  double res = (double)(dp[n][0][0][M-1][M-1])/(double)n - res1;  res = sqrt(res);  res = res\*1000;  res+=0.5;  int R = (int)(res);  res = (double)(R)/1000;  printf("%.3lf\n",res);  }  return 0;  } |

## 三角形最大面积

【题目描述】三角形最大面积（TriangleArea.cpp/c/pas）

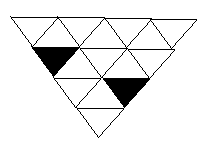
张琪曼：“圣诞节快到了，我们用这张三角形银纸来制作圣诞树上的大星星吧。”

李旭琳：“好啊，不过这纸上面怎么有许多洞？”

楚继光：“哈哈，不好意思，我已经事先剪下了一些小的三角形来制作小星星了。”

所以现在唯有寻找一个算法，计算出每张银纸上还能切出来的最大三角形面积。

如图所示，给定一张三角形银纸，黑色部分代表已被剪的部分，试求出白色区域中最大的三角形面积。



图

【输入格式】

输入文件包含若干个三角形银纸描述。每个三角形描述的第1行是一个整数n(1<=n<=100)，表示该三角形的高。接下来的n行每行包含由空格、“#”和“-”组成的字符串表示三角形的状况。其中“#”代表黑色的区域，“-”代表白色的区域，空格是用来填充输入的左边，从而使得整个输入构成一个三角形的形状。

对每个三角形，每行字符“#”和“-”的数目之和都是奇数，由2n-1递减到1。

全部输入以n=0结束。

【输出格式】

对输出的每个三角形，输出白色的区域中最大三角形的面积。注意，最大三角形可以是顶角朝上的，如同第2个样例输入所示。

样例输入

5

#-##----#

-----#-

---#-

-#-

-

4

#-#-#--

#---#

##-

-

0

样例输出

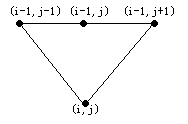
9

4

【算法分析】

对于三角形有两种情况：

1.尖向下时如图所示：



图

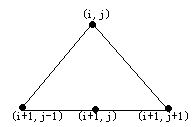
c[i,j]表示输入图形（i,j）这点的值；

s[i,j]表示以（i,j）这点为尖向下的顶点构成三角形的最大面积。

If(c[i-1,j-1]=’-’ and (c[i-1,j]=’-’) and (c[i-1,j+1]=’-’ then

s[i,j]=min(s[i-1,j-1],s[i-1,j+1])+1

2.尖向上时如图所示：



图

c[i,j]表示输入图形（i,j）这点的值

s[i,j]表示以(i,j)这点为尖向上的顶点构成三角形的最大面积。

If (c[i+1,j-1]=’-’) and (c[i+1,j]=’-’) and (c[i+1,j+1]=’-‘) then

s[i,j]=min(s[i+1,j-1],s[i+1,j+1])+1

而且一定要仔细看题目给的三角形，它有可能是正面朝前或向后三角形。

注意几点：

尖向上的情况，只能是一行中的偶数个。

尖向下的情况，只能是一行中的奇数个。

参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69 | //三角形最大面积  #include<iostream>  #include<fstream>  #define M 1000  #define MAX 99999999  #define M 1000  using namespace std;  ifstream fin("TriangleArea.in");  ofstream fout("TriangleArea.out");  bool s[M+1][M+1];  int a[M+1][M+1];  int n;  void dp()  {  int i,j,k,maxn=0;  for(i=1;i<=n;++i)  for(j=i;j<=n+n-i;j+=2)  if(s[i][j])  {  if(s[i-1][j-1]&&s[i-1][j]&&s[i-1][j+1])  a[i][j]=min(a[i-1][j-1],a[i-1][j+1])+1;  else  a[i][j]=1;  }  for(i=n-1;i>=1;--i)  {  for(j=i+1;j<=n+n-i;j+=2)  if(s[i][j])  {  if(s[i+1][j-1]&&s[i+1][j]&&s[i+1][j+1])  a[i][j]=min(a[i+1][j-1],a[i+1][j+1])+1;  else  a[i][j]=1;  }  }  for(i=1;i<=n;++i)  for(j=i;j<=n+n-i;++j)  if(maxn<a[i][j])  maxn=a[i][j];  fout<<maxn\*maxn<<endl;  }  void init()  {  int i,j;  char c;  fin>>n;  for(i=1;i<=n;++i)  for(j=i;j<=n+n-i;++j)  {  fin>>c;  s[i][j]=(c=='-');  }  }  int main()  {  init();  dp();    fin.close();  fout.close();  return 0;  } |

## 基因序列

【题目描述】基因序列（GEN.cpp/c/pas）

在将要到来的星际战争中，基因武器的杀伤威力极其巨大，因此研究天顶星人的基因十分必要。魔法学院发现天顶星人的基因是一个有限的基因序列，它是由大写的英文字母A-Z组成，不同的字母表示不同种类的基因。一个基因可以分化成为一对新的基因。这种分化被一个定义的规则集合所控制。每个分化的规则可以用三个大写字母A1A2A3表示，含义为基因A1可以分化成A2A3。我们用S代表特种基因，繁殖天顶星人是从特种基因序列开始。根据给定的规则，它由被选择控制规则对基因不断进行繁殖而成。

你需要从文本文件GEN.IN 读入一个定义的规则集和一个想生成的天顶星人单词序列。对每一个给定的基因，根据给定的分化规则，检查是否它能从某一个确定特种基因序列生成，如果能，找到最小的序列长度，将结果写入文本文件GEN.OUT。

【输入格式】

在文件GEN.IN 的第一行有一个整数n, 1 <= n <= 10000，下面n 每一行为一个分化规则，这些规则都由包含A – Z的三个大写字母组成.

接下来有一个整数k, 1 <= k <= 10000，接下来的k 行有一个基因序列，基因序列由没有空格的单词组成，最多100 个英文大写字母。

【输出格式】

在文件GEN.OUT中有k行，在第I行应写入：一个正整数――需要生成第I个基因的最小长度；或者单词 NIE， 如果不能生成对应的基因。

【输入样例】

6

SAB

SBC

SAA

ACA

BCC

CBC

3

ABBCAAABCA

CCC

BA

【输出样例】

3

1

NIE

【算法分析】

读取时用一个map[A][B]数组表示 字母AB能变成的字母，由于只有26个字母，可以用一个26位的二进制数表示。进行两次动归：

f[i][j]表示从字符串 从 i 到 j 能变成的字母，同理也是个二进制数，

f[i][j]=f[i][j] |map[c1][c2] 存在 (f[i][k]&char[c1]&&f[k+1][j]&char[c2])

不难得到那几段字符串能变成 ‘S’

再进行一次动归：设g[i]表示前 i 个字符能变成几个 ‘S’

则g[i]=min(g[j]+1) 存在(f[i][j+1]&char['S'])

复杂度O(len3×262)+O(len2)

　　参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64 | //基因序列  #include<algorithm>  #include<iostream>  #include<cmath>  #define LL long long  #define INF 99999999  #define Min(num1,num2) if(num1>num2) num1=num2  #define Max(num1,num2) if(num1<num2) num1=num2  using namespace std;  int n,f[101][101],g[101],num[101],map[101][101];  string s;  void work()  {  memset(f,0,sizeof f);  cin>>s;  int l=s.size();  for(int k=1,i=0;i<l;k++,i++)  f[k][k]=num[s[i]-'A'];  for(int p=1;p<=l;p++)  for(int i=1;i<=l;i++)  {  int j=i+p;  if(j>l)  break;  for(int k=i;k<j;k++)  for(int ci=0;ci<26;ci++)  for(int cj=0;cj<26;cj++)  if((f[i][k]&num[ci])&&(f[k+1][j]&num[cj]))  f[i][j]|=map[ci][cj];  }  int key='S'-'A';  for(int i=1;i<=l;i++)  g[i]=INF;  g[0]=0;  for(int i=1;i<=l;i++)  for(int j=1;j<=i;j++)  if((f[j][i]&num[key])&&g[j-1]!=INF)  Min(g[i],g[j-1]+1);  g[l]==INF ? printf("NIE\n") : printf("%d\n",g[l]);  }  int main()  {  freopen("GEN.in","r",stdin);  freopen("GEN.out","w",stdout);  scanf("%d\n",&n);  num[0]=1;  for(int i=1;i<=26;i++)  num[i]=num[i-1]<<1;  for(int a,b,c,i=1;i<=n;i++)  {  a=getchar()-'A';  b=getchar()-'A';  c=getchar()-'A';  map[b][c]|=num[a];  getchar();  }  int T;  scanf("%d\n",&T);  while(T--)  work();  } |

## 传纸条

【题目描述】传纸条（message.cpp/c/pas）NOIP 2008

张琪曼和李旭琳是好朋友也是同班同学，她们在一起总有谈不完的话题。在一次活动中，班上同学安排成一个m行n列的矩阵，而张琪曼和李旭琳被安排在矩阵对角线的两端，因此，她们就无法直接交谈了。幸运的是，她们可以通过传纸条来进行交流。纸条要经由许多同学传到对方手里，张琪曼坐在矩阵的左上角，坐标(1，1)，李旭琳坐在矩阵的右下角，坐标(m，n)。从张琪曼传到李旭琳的纸条只可以向下或者向右传递，从李旭琳传给张琪曼的纸条只可以向上或者向左传递。

在活动进行中，张琪曼希望给李旭琳传递一张纸条，同时希望李旭琳给他回复。班里每个同学都可以帮他们传递，但只会帮他们一次，也就是说如果此人在张琪曼递给李旭琳纸条的时候帮忙，那么在李旭琳递给张琪曼的时候就不会再帮忙。反之亦然。

还有一件事情需要注意，全班每个同学愿意帮忙的好感度有高有低（注意：张琪曼和李旭琳的好心程度没有定义，输入时用0表示），可以用一个0-1000的自然数来表示，数越大表示越好心。张琪曼和李旭琳希望尽可能找好心程度高的同学来帮忙传纸条，即找到来回两条传递路径，使得这两条路径上同学的好心程度之和最大。现在，请你帮助张琪曼和李旭琳找到这样的两条路径。

【输入格式】

　　输入文件的第一行有2个用空格隔开的整数m和n，表示班里有m行n列（2<=m，n<=50）。

接下来的m行是一个m×n的矩阵，矩阵中第i行j列的整数表示坐在第i行j列的学生的好心程度（不大于1000）。每行的n个整数之间用空格隔开。

【输出格式】

　　输出共一行，包含一个整数，表示来回两条路上参与传递纸条的学生的好心程度之和的最大值。

【样例输入】

　　3 3

　　0 3 9

　　2 8 5

　　5 7 0

【样例输出】

34

### 三维算法1

问题可以转化为纸条同时从1,1出发的两条道路，并注意两条道路不能交叉，可设四维数组f[i][j][k][l]表示从 (1, 1) 位置由两条不交叉的线路走到 (i, j)，(k, l) 位置时的最大好感度和。其方程为：

f[i][j][k][l]=max(f[i-1][j][k-1][l],f[i-1][j][k][l-1],f[i][j-1][k-1][l],f[i][j-1][k][l-1]) +a[i][j]+a[k][l]

仔细观察很容易得到一个这样的结论 纸条传的横坐标+纵坐标=走的步数; 通过这个结论便很简单的降一维，即l=i+j-k(l>0),因而时间复杂度可以从O（n4）降到O(n3)。

参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | //传纸条  #include<stdio.h>  #include<stdlib.h>  int f[52][52][52][52] = {0}, n, G[52][52];  int max(int a, int b, int c, int d)//返回4个值的最大值  {  if (a < b) a= b;  if (a < c) a= c;  if (a < d) a= d;  return a;  }  int main()  {  freopen("message.in","r",stdin);  freopen("message.out","w",stdout);  int m, n, i, j, k, l;  scanf("%d%d", &m, &n);  for (i = 1; i <= m; i++)  for (j = 1; j <= n; j++)  scanf("%d", &G[i][j]);  for (i = 1; i <= m; i++)  for (j = 1; j <= n; j++)  for (k = 1; k <= m; k++)  {  if (i+j-k > 0)  l = i+j-k;  else  continue;  f[i][j][k][l]=max(f[i-1][j][k-1][l],f[i-1][j][k][l-1],f[i][j-1][k-1][l],f[i][j-1][k][l-1])+G[i][j]+G[k][l];  if (i == k && j == l)  f[i][j][k][l] -= G[i][j];  }  printf("%d\n", f[m][n][m][n]);  } |

### 三维算法2

用状态 f [k][x1][x2] 来表示走第k步时第一条路所在位置的横坐标和第二条路所在位置的横坐标，由于步数和横坐标都确定，所以纵坐标也是确定的，如第一条路的纵坐标即为k-x1+1，第二条路的纵坐标为k-x2+1,方便起见我们用y1,y2来表示，这样每一个状态的前一个状态就有四种可能，分别是：

（1)f[k-1][x1][x2](两条路都向下走)；

（2）f[k-1][x1-1][x2](第一条路向右，第二条路向下)；

（3）f[k-1][x1][x2-1](第一条路向下，第二条路向右)；

（4）f[k-1][x1-1][x2-1](两条路都向右)。

取最大值并加上两条路的所在位置的值即可，动态转移方程为：

f(k,x1,x2)=max{f(k-1,x1,x2), f(k-1,x1-1,x2) ,f(k-1,x1,x2-1), f(k-1,x1-1,x2-1) }+v(x1,y1)+v(x2,y2) (注意两条路不能有重复点)

参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | //传纸条－三维算法2  #include <stdio.h>  #include <stdlib.h>  int data[101][52][52];  int main()  {  freopen("message.in","r",stdin);  freopen("message.out","w",stdout);  int m,n,i,j,a[100][100],k,x1,x2,t=0,mm;  scanf("%d%d",&m,&n);  for(i=1;i<=m;i++)  for(j=1;j<=n;j++)  scanf("%d",&a[i][j]);  mm=m<n?n:m;  data[2][2][1]=a[1][2]+a[2][1];  data[2][1][2]=data[2][2][1];  for(k=3;k<=m+n-1;k++)  for(x1=1;x1<=k;x1++)  for(x2=1;x2<=k;x2++)  {  if(x1==x2)  break;  if(k==n+m-1)  data[m+n-1][n][n]=data[m+n-1][n-1][n]<data[m+n-1][n][n-1]?data[m+n-1][n][n-1]:data[m+n-1][n-1][n];  t=data[k-1][x1][x2]+a[k-x1+1][x1]+a[k-x2+1][x2];  if(data[k-1][x1-1][x2]+a[k-x1+1][x1]+a[k-x2+1][x2]>t)  t=data[k-1][x1-1][x2]+a[k-x1+1][x1]+a[k-x2+1][x2];  if(data[k-1][x1][x2-1]+a[k-x1+1][x1]+a[k-x2+1][x2]>t)  t=data[k-1][x1][x2-1]+a[k-x1+1][x1]+a[k-x2+1][x2];  if(data[k-1][x1-1][x2-1]+a[k-x1+1][x1]+a[k-x2+1][x2]>t)  t=data[k-1][x1-1][x2-1]+a[k-x1+1][x1]+a[k-x2+1][x2];  data[k][x1][x2]=t;  }  printf("%d",data[m+n-1][n][n]);  return 0;  } |

### 费用流算法

该算法比动态规划效率高，内存空间占用少，但难度已远超NOIP，无网络流基础的读者请略过。

把问题抽象成图论问题，数学模型是求从S到T的两条不相交的路径，使得路径上点的权值之和最大。

费用流建模，首先拆点，把顶点i拆成i.a和i.b，i.a 与i.b之间连接一条费用为“好心值”，容量为1的有向边，特殊地，左上角和右下角两个节点拆分后点内边容量设为2，因为我们要找两条不相交路径。i右边和下边的节点j，连接一条(i.b,j.a)费用为0，容量为1的有向边。

求最大费用最大流即可，费用流值就是要求的结果。参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154 | //传纸条 －费用流，已超过NOIP难度  #include <iostream>  #include <cstdio>  #include <cstdlib>  #include <cmath>  #include <cstring>  using namespace std;  const int MAXL=51,MAXN=MAXL\*MAXL\*2,MAXM=MAXN\*3,INF=~0U>>1;  struct Queue  {  int Q[MAXN],head,tail,size;  bool inq[MAXN];  Queue()  {  memset(inq,0,sizeof(inq));  head=size =0;  tail=-1;  }  void ins(int p)  {  inq[p]=true;  if (++tail >= MAXN)  tail = 0;  Q[tail] = p;  size ++;  }  int pop()  {  int p=Q[head];  if (++head >= MAXN)  head = 0;  inq[p]=false;  size --;  return p;  }  }Q;  struct edge  {  edge \*next,\*op;  int t,c,v;  }ES[MAXM],\*V[MAXN],\*fe[MAXN];  int N,M,EC,S,T,CostFlow;  int dist[MAXN],ft[MAXN];  inline void addedge(int a,int b,int v)  {  ES[++EC].next = V[a];  V[a]=ES+EC;  V[a]->t = b;  V[a]->c=1;  V[a]->v = v;  ES[++EC].next = V[b];  V[b]=ES+EC;  V[b]->t = a;  V[b]->c=0;  V[b]->v = -v;  V[a]->op = V[b];  V[b]->op = V[a];  }  void init()  {  int i,j,a,b,c;  freopen("message.in","r",stdin);  freopen("message.out","w",stdout);  scanf("%d%d",&N,&M);  for (i=1;i<=N;i++)  {  for (j=1;j<=M;j++)  {  a=(i-1)\*M+j;b=a+a;a=b-1;  scanf("%d",&c);  addedge(a,b,c);  }  }  ES[EC-1].c=ES[1].c=2;  for (i=1;i<=N;i++)  {  for (j=1;j<=M;j++)  {  a=((i-1)\*M+j)\*2;  if (j+1<=M)  {  b = ((i-1)\*M+j+1)\*2 - 1;  addedge(a,b,0);  }  if (i+1<=N)  {  b = (i\*M+j)\*2 - 1;  addedge(a,b,0);  }  }  }  S=1; T = N \* M \* 2;  }  bool spfa()  {  int i,j;  for (i=S;i<=T;i++)  dist[i]=-INF;  dist[S]=0;  Q.ins(S);  while (Q.size)  {  i= Q.pop();  for (edge \*e=V[i];e;e=e->next)  {  j=e->t;  if (e->c && dist[i] + e->v > dist[j])  {  dist[j] = dist[i] + e->v;  ft[j] = i;  fe[j] = e;  if (!Q.inq[j])  Q.ins(j);  }  }  }  return dist[T]!=-INF;  }  void aug()  {  int i,delta=INF;  for (i=T;i!=S;i=ft[i])  if (fe[i]->c < delta)  delta = fe[i]->c;  for (i=T;i!=S;i=ft[i])  {  fe[i]->c -= delta;  fe[i]->op->c +=delta;  CostFlow += fe[i]->v \* delta;  }  }  void solve()  {  while(spfa())  aug();  }  int main()  {  freopen("message.in","r",stdin);  freopen("message.out","w",stdout);  init();  solve();  printf("%d\n",CostFlow);  return 0;  } |

## 猛兽动物园

【问题描述】猛兽动物园(zoo.cpp/c/pas)　APIO2007

张琪曼惊喜道：“哇，抓到这么多猛兽，把它们都关到动物园吧。不过这个牙齿锋利的狮子好可怕啊。”

楚继光：“我倒不觉得，看那头鬃毛多可爱啊，不过这大猩猩太难看了。”

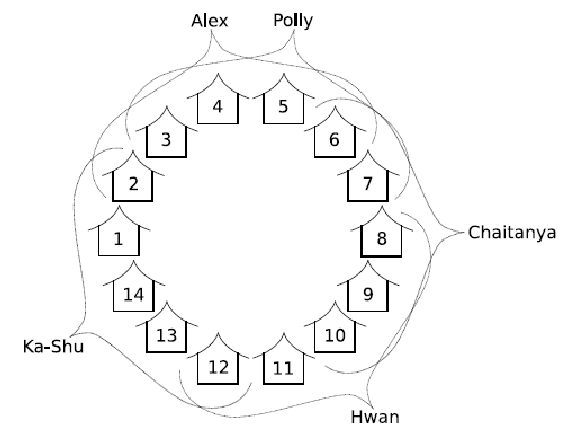
　　现在这些猛兽都被关在魔法世界的圆形动物园，圆形动物园包含一大圈围栏，每个围栏里有一种动物，有K个小朋友站在大围栏圈的外面，可以看到连续的5个围栏。每个小朋友有喜欢和害怕的动物。当下面两处情况之一发生时，小朋友就会高兴：

至少有一个他害怕的动物被移走；

至少有一个他喜欢的动物没被移走。

你可以选择将一些动物从围栏中移走以使得小朋友不会害怕。但你不能移走所有的动物，否则小朋友们就没有动物可看了。

例如，考虑图中的小伙伴和动物：





图

假如你将围栏4和12的动物移走。Alex和Ka-Shu将很高兴，因为至少有一个他们害怕的动物被移走了。这也会使Chaitanya高兴，因为他喜欢的围栏6和8中的动物都保留了。但是，Polly和Hwan将不高兴，因为他们看不到任何他们喜欢的动物，而他们害怕的动物都还在。这种安排方式使得三个小朋友高兴。

现在，换一种方法，如果你将围栏4和6中的动物移走，Alex和Polly将很高兴，因为他们害怕的动物被移走了。Chaitanya也会高兴，虽然他喜欢的动物6被移走了，他仍可以看到围栏8里面他喜欢的动物。同样的Hwan也会因可以看到自己喜欢的动物12而高兴。唯一不高兴的只有Ka-Shu。

如果你只移走围栏13中的动物，Ka-Shu将高兴，因为有一个他害怕的动物被移走了，Alex, Polly, Chaitanya和Hwan也会高兴，因为他们都可以看到至少一个他们喜欢的动物。所以有5个小朋友会高兴。这种方法使得了最多的小朋友高兴。

【输入格式】

输入的第一行包含两个整数N，C，用空格分隔。N是围栏数(1≤N≤10 000)，C是小朋友的个数(1≤C≤50 000)。围栏按照顺时针的方向编号为1，2，3，...，N。

接下来的C行，每行描述一个小朋友，描述下面的形式给出： E F L X1X2 ... XF Y1 Y2 ... YL其中：

E表示小朋友可以看到的第一个围栏的编号(1≤E≤N)，也就是说，小朋友可以看到的围栏为E，E+1，E+2，E+3，E+4。注意，如果编号超过N将继续从1开始算。如：当N=14，E=13时，小朋友可以看到的围栏为13，14，1，2和3。

F表示小朋友害怕的动物数。L表示小朋友喜欢的动物数。 围栏X1，X2，...， XF中包含小朋友害怕的动物。 围栏Y1，Y2，...， YL中包含该小朋友喜欢的动物。

X1， X2，...，XF， Y1， Y2，... YL是两两不同的数，而且所表示的围栏都是小朋友可以看到的。

小朋友已经按照他们可以看到的第一个围栏的编号从小到大的顺序排好了(这样最小的E对应的小朋友排在第一个，最大的E对应的小朋友排在最后一个)。注意可能有多于一个小朋友对应的E是相同的。

【输出格式】

仅输出一个数，表示最多可以让多少个小朋友高兴。

【输入样例1】

14 5

2 1 2 4 2 6

3 1 1 6 4

6 1 2 9 6 8

8 1 1 9 12

12 3 0 12 13 2

【输出样例1】

6

【输入样例2】

12 7

1 1 1 1 5

5 1 1 5 7

5 0 3 5 7 9

7 1 1 7 9

9 1 1 9 11

9 3 0 9 11 1

11 1 1 11 1

【输出样例2】

6

【样例说明】

第一个样例是题目描述中的例子，所有的C=5个小朋友都能高兴。第二个样例是一个不能使得所有C=7个小朋友都高兴的例子。

【算法分析】

动态规划，用二进制表示动物的取舍。

F[i][j]表示从1到i+4的动物，其中i到i+4的动物的取舍状态为j时，最多的高兴的小朋友的数目。

状态转移方程为:

F[i][j]=max{F[i-1][(j<<1)&31]，F[i-1][((j<<1)&31)|1]}+num[i][j];

j的二进制串中，最后一位表示i，倒数第二为表示i+1,以此类推。

num[i][j]为看到i到i+4的动物的小朋友在j的状态下高兴的数目；

边界条件：F[0][j]=0;

目标函数：max{F[N][j]}(0<=j<32);

考虑到动物园为环形，因此要枚举固定开头的4个动物。

参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106 | //猛兽动物园  #include<iostream>  #include<cstdio>  #include<cstring>  using namespace std;    const int MaxN=100010;  int Hate,Love,N,C,ans=0,E;  int F[MaxN][32]={0},num[MaxN][32];  void DFS(int x,int dep,int status)  {  if (dep>4)  {  if((Love&status)||(Hate&status)!=Hate)  num[E][status]++;  return;  }  DFS(x,dep+1,status);  DFS(x,dep+1,status|(1<<dep));  }  void Init()  {  int i,H,L,x;  scanf("%d%d",&N,&C);  for (i=1;i<=C;i++)  {  scanf("%d%d%d",&E,&H,&L);  Hate=Love=0;  while (H--)  {  scanf("%d",&x);  Hate=Hate|(1<<(x-E+N)%N);  }  while (L--)  {  scanf("%d",&x);  Love=Love|(1<<(x-E+N)%N);  }  DFS(i,0,0);  }  }  int Max(int a,int b)  {  return a>b?a:b;  }  void DP(int st)  {  int i,j,k,r,p,q,w1,w2;  for (j=0;j<32;j++)  F[0][j]=0;  for (i=1;i<=N;i++)  if (i<5)  for (j=0;j<32;j++)  {  q=st>>(i-1);  r=(1<<(5-i))-1;  if ((j&r)==q)  {  w1=(j<<1)&31;w2=w1+1;  F[i][j]=Max(F[i-1][w1],F[i-1][w2])+num[i][j];  }  else  F[i][j]=0;  }  else if(i+3<N)  for (j=0;j<32;j++)  {  w1=(j<<1)&31;w2=w1+1;  F[i][j]=Max(F[i-1][w1],F[i-1][w2])+num[i][j];  }  else  {  q=st&((1<<(i+4-N))-1);  for (j=0;j<32;j++)  if ((j>>(N-i+1))==q)  {  w1=(j<<1)&31;w2=w1+1;  F[i][j]=Max(F[i-1][w1],F[i-1][w2])+num[i][j];  }  else  F[i][j]=0;  }  for (j=0;j<32;j++)  if(ans<F[N][j])  ans=F[N][j];  }  void Solve()  {  for (int st=0;st<16;st++)  DP(st);  printf("%d\n",ans);  }  int main()  {  freopen("zoo.in","r",stdin);  freopen("zoo.out","w",stdout);  Init();  Solve();  return 0;  } |

## 技能树

【问题描述】技能树（skill）Tju1053浙江省2004组队赛第二试

修罗王挖到“神话时代”的宝藏后，发现宝藏里的科技源可以看成是一棵技能树，一颗技能树的每个结点都是一项技能，要学会这项技能则需要耗费一定的技能点数。

只有学会了某一项技能以后，才能继续学习它的后继技能。每项技能又有着不同的级别，级别越高效果越好，而技能的升级也是需要耗费技能点数的。

现在修罗王已有一定的技能点数，他想尽可能地利用这些技能点数来达到最好的效果。因此他给所有的级别都打上了分，他认为效果越好的分数也越高。现在他要你帮忙寻找一个分配技能点数的方案，使得分数总和最高。

【输入格式】

该题有多组测试数据。

每组测试数据第一行是一个整数n（1<=n<=20）,表示所有不同技能的总数。

接下来依次给出n个不同技能的详细情况。

每个技能描述包括5行。

第一行是该技能的名称。

第2行是该技能在技能树中父技能的名称，名称为None则表示该技能不需要任何的先修技能便能学习。

第3行是一个整数L（1<=L<=20），表示这项技能所能拥有的最高级别。

第4行共有L个整数，其中第I个整数表示从第I-1级升到第I级所需要的技能点数（0级表示没有学习过）。

第5行包括L个整数，其中第I个整数表示从第I-1级升级到第I级的效果评分，分数不超过20。

在技能描述之后，共有两行，第1行是一个整数P，表示目前所拥有的技能点数。

接下来1行是N个整数，依次表示角色当前习得的技能级别，0表示还未学习。这里不会出现非法情况。

【输出格式】

每组测试数据只需输出最佳分配方案所得的分数总和。

【输入样例】

3

Freezing Arrow

Ice Arrow

3

3 3 3

15 4 6

Ice Arrow

Cold Arrow

2

4 3

10 17

Cold Arrow

None

3

3 3 2

15 5 2

10

0 0 1

【输出样例】

42

【算法分析】

由题目可知，技能分为自身技能、兄弟技能和子技能，子技能要在学习了父技能的条件下学习，自身等级技能可以一步一步学习。故可开设三个数组brother[50]，child[50]，son[50]保存。

设dp(i，j)表示在第i个技能下，有j个技能点，最多能得到的价值。则有动态转移方程：

dp(i，j)=max{dp(child[i]，cost)+dp(brother[i]，j-cost)} (j>=cost，cost是花费给child的技能点)，其中dp(child[i]，cost)表示升级子技能得到的价值，dp(brother[i]，j-cost)表示兄弟技能得到的价值。

参考代码如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105 | //技能树  #include <iostream>  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  using namespace std;  int n,p,i,l[50];  int brother[50],child[50],son[50];//兄弟技能，子技能，自身  long f[50][1000];  struct Node  {  char n[100], f[100];  int l, cost[50], value[50], learn;  }stree[50];  int find(char ch[100])//找到父技能，只有学了父技能，才能学子技能    {  for (int j = 1; j <= n; j++)  if (strcmp(stree[j].n, ch) == 0)  return j;  return 0;  }  void clear()  {  memset(son, 0, sizeof(son));  memset(brother, 0, sizeof(brother));  memset(child, 0, sizeof(child));  memset(f, -1, sizeof(f));  memset(stree, 0, sizeof(stree));  }  void set\_tree()  {  int father;  clear();  for (i = 1; i <= n; i++)  {  cin.getline(stree[i].n, 100);//extra '\n'  cin.getline(stree[i].n, 100);  cin.getline(stree[i].f, 100);  scanf("%d", &stree[i].l);  for (int j = 1; j <= stree[i].l; j++)  scanf("%d", &stree[i].cost[j]);  for (int j = 1; j <= stree[i].l; j++)  scanf("%d", &stree[i].value[j]);  }  scanf("%d", &p);  for (i = 1; i <= n; i++)  scanf("%d", &stree[i].learn);  for (i = 0; i <= p; i++)  f[0][i] = 0;  for (i = 1; i <= n; i++)//按要求把技能之间的关系建好树  {  father = find(stree[i].f);  if (son[father] == 0)  child[father] = i;  else  brother[son[father]] = i;  son[father] = i;  }  }  long dp(int root, int j)//动规  {  if (root == 0)  return f[root][j] = 0; // 此时直接继续  if (f[root][j] >= 0)  return f[root][j];  long max, co = 0, va = 0, s;  max = dp(brother[root], j);  if (stree[root].learn) //如果该技能已经学习过一些  for (int k = 1; k <= j; k++)  {  s=dp(child[root],k)+dp(brother[root],j-k);//子技能  if(s>max)  max = s;  }  for (int g=stree[root].learn+1;g<=stree[root].l;g++)//学习本身  {  co += stree[root].cost[g];  va += stree[root].value[g];  for (int k = 0; k <= j - co; k++)  {  s=dp(child[root],k)+dp(brother[root],j-co-k)+va;  if(s>max)  max = s;  }  }  return f[root][j] = max;  }  int main()  {  freopen("skill.out", "w", stdout);  freopen("skill.in", "r", stdin);  while (scanf("%d", &n) != EOF)  {  set\_tree();  printf("%ld\n", dp(child[0], p));  }  } |

## 逆转未来

【题目描述】逆转未来（reverse.cpp/c/pas）HDU 1561

后世的历史学家始终不明白为什么修罗王在人类文明存亡的生死攸关之际给了天顶星人最后的一击，他们从心理学、阴谋论、人性说等方面提出了种种猜想并为此争论不休。不过我们不必理会他们那些夸夸其谈的酸腐理论，因为全球著名无厘头导演张某某拍摄的搞笑纪录片《宇宙文明终极战－我和天顶星人不得不说的故事》里说的很靠谱：修罗王一定是看上天顶星人的宝物了。他是这样描述的：愚蠢而贪婪的修罗王率领他的铁血军团攻击由万恶的天顶星人占据的N座城堡，每座城堡都有一定的宝物，但由于地理位置原因，有些城堡不能直接攻克，要攻克这些城堡必须先攻克其它某一个特定的城堡。你能帮修罗王算出要获得尽量多的宝物应该攻克哪M个城堡吗？

【输入格式】

每个测试实例首先包括2个整数，N，M。(1 <= M <= N <= 200)；在接下来的N行里，每行包括2个整数a，b。在第 i 行，a 代表要攻克第 i 个城堡必须先攻克第 a 个城堡，如果 a = 0 则代表可以直接攻克第 i 个城堡。b 代表第 i 个城堡的宝物数量，b >= 0。当N = 0， M = 0输入结束。

【输出格式】

对于每个测试实例，输出一个整数，代表攻克M个城堡所获得的最多宝物的数量。

【输入样例】

3 2

0 1

0 2

0 3

7 4

2 2

0 1

0 4

2 1

7 1

7 6

2 2

0 0

【输出样例】

5

13

【算法分析】

首先将多叉树转化为二叉树，然后定义dp[N][M]为节点号为N的子树上取M个节点的最大值，则动态转移方程为：

dp[N][M]=MAX{dp[N.right][M]，dp[N.left][K-1]+N.value+dp[N.right][N-M]}，即取MAX{直接去N的兄弟上M个点的最大值，去N的K-1个孩子+N的值（取它的孩子，则必须取它）+N的有兄弟M-K个点的最大值}。

参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94 | //逆转未来  #include<iostream>  #include <cstdlib>  using namespace std;  const int MAX\_N = 208;  struct Tree  { int lastSon;//最后一个孩子  int left;//儿子(左）  int right;//兄弟（右）  int value;//权重  }tree[MAX\_N];  int idx,N,M;  int dp[MAX\_N][MAX\_N];  inline int MAX(int a,int b)  {  return a>b?a:b;  }  void Init()  {  int i,j;  for(i=0;i<=N;i++) //初始化  {  tree[i].lastSon=-1;  tree[i].right=-1;  tree[i].left=-1;  tree[i].value=0;  for(j=0;j<=N;j++)  dp[i][j]=-1;  }  }  void BuildTree(int son,int parent)//建树  {  if(tree[parent].lastSon==-1)//左孩子  {  tree[parent].left=son;  }  else //右兄弟  {  tree[tree[parent].lastSon].right=son;  }  tree[parent].lastSon=son;  }  bool Read()//读入数据  {  int p,i;  if(scanf("%d%d",&N,&M)&&!N&&!M)  return false;    Init();  for(M++,i=1;i<=N;i++)  {  scanf("%d%d",&p,&tree[i].value);  BuildTree(i,p);  }  return true;  }  int DP(int node,int k)  {  if(node==-1||k==0) //如果node或者k==0  {  return 0;  }  if(dp[node][k]!=-1)//如果已经求出值了  {  return dp[node][k];//则直接返回  }  int i,tmp;  dp[node][k]=DP(tree[node].right,k);  for(i=1;i<=k;i++)  {  tmp=DP(tree[node].left,i-1)+tree[node].value+DP(tree[node].right,k-i);  dp[node][k]=MAX(dp[node][k],tmp);  }  return dp[node][k];  }  int main()  {  freopen("reverse.in","r",stdin);  freopen("reverse.out","w",stdout);  while(Read())  {  DP(0,M);  printf("%d\n",dp[0][M]);  }  return 0;  } |

## 最后一战

【题目描述】最后一战（lastwar.cpp/c/pas） tyvj 1513

受伤的天顶星人愤怒道：“你不是要做这世界之主吗？那为什么之前你不仅故意通风报信毁灭了我交给你的那些强大武装，现在还来攻击我？”

修罗王冷笑：“没错，我是不喜欢这个世界的许多规则，所以我想重新按我的规则来改正这个世界，但这是我和这个世界的事，与你异星人无关。再说了，你觉得我修罗王，什么时候会受别人控制？”

“好吧，我承认我低估了你们人类，但你要想抓到我也是不可能的。” 天顶星人在狂笑声中消失。

此时修罗王正在一个城堡中，城堡中的路形成一棵树，每条路的结点处因地形的险要程度不同，所需要的士兵数量也不相同，只有达到需要的士兵数量，该结点才可以构成强大的火力网以对抗天顶星人，注意，若有士兵在一个结点上时，与该结点相连的所有边都可以被防守。

请你帮修罗王计算出他至少需要放置多少士兵才可以防守住所有道路。

【输入格式】

输入文件中数据表示一棵树，描述如下：

第1行 n，表示树中结点的数目。

第2行至第n+1行，每行描述每个结点信息，依次为：该结点标号i（0<i<=n），在该结点安置士兵所需的数量k，该边的儿子数m，接下来m个数，分别是这个节点的m个儿子的标号r1，r2，...，rm。

对于一个n（0 < n <= 1500）个结点的树，结点标号在1到n之间，且标号不重复。

【输出格式】

输出文件仅包含一个数，为至少需要的士兵数。

【样例输入】

6

1 30 3 2 3 4

2 16 2 5 6

3 5 0

4 4 0

5 11 0

6 5 0

【样例输出】

25

【算法分析】

设F[i，0]表示i点不放，且以i为根节点的子树（包括i节点）全部被防守到；

F[i，1]表示i点不放，且以i为根节点的子树（可以不包括i节点）全部被防守到；

F[i，2]表示i点放，且以i为根节点的子树全部被防守到。

则可以推出：

1、由F[i,0]定义可知，设j为i的儿子节点，至少要有一个i的儿子节点是放置士兵的，其余的儿子节点可放可不放，但由于根节点i不放，所以其余的儿子节点如果不放的话，必须保证能被防守到，即F[j][0]；所以我们需要枚举必须放置的儿子节点，转移方程如下:

F[i，0] = min{ ∑(min(F[j][0]，F[j，2]))+F[k，2]}，其中k为枚举的必放的儿子节点，j为除了k之外的儿子节点。

2、由F[i,1]定义可知，i可以被防守到也可以不被防守到,但儿子节点必须都要被防守到，转移方程如下：

F[i，1] = ∑(min(F[j，0]，F[j，2]))，其中 j是i的儿子节点。

3、由F[i，2]定义可知，i点放置了士兵，所以对于每个儿子节点都能被防守到，取F[j，0]，F[j，1]，F[j，2]最小值即可，转移方程如下：

F[i，2] = min(F[j，0]，F[j，1]，F[j，2])，其中 j是i的儿子节点。

对于叶节点i，F[i，0] = F[i，2] = data[i]，F[i，1] = 0。

参考程序如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69 | //最后一战  #include <iostream>  #include <algorithm>  #include <cstdio>  #include <cstring>  #include <string>  using namespace std;    const int MAX = 1500+10;  int f[MAX][3],data[MAX],son[MAX][MAX],len[MAX],in[MAX];  int n,x,root;  void dp(int x)  {  if (len[x] == 0)  {  f[x][0] = f[x][2] = data[x];  f[x][1] = 0;  return;  }  for (int i = 1;i <= len[x];i++)  dp(son[x][i]);  f[x][0] = 1299999999;  for (int i = 1;i <= len[x];i++)  {  int tmp = 0;  for (int j = 1;j <= len[x];j++)  if (i!=j)  tmp += min(f[son[x][j]][0],f[son[x][j]][2]);  f[x][0] = min(f[x][0],tmp+f[son[x][i]][2]);  }  f[x][1] = 0;  for (int i = 1;i <= len[x];i++)  f[x][1] += min(f[son[x][i]][0],f[son[x][i]][2]);  f[x][2] = data[x];  for (int i = 1;i <= len[x];i++)  f[x][2]+=min(f[son[x][i]][0],min(f[son[x][i]][1],f[son[x][i]][2]));  }    int main()  {  freopen("lastwar.in","r",stdin);  freopen("lastwar.out","w",stdout);  scanf("%d",&n);  memset(data,0,sizeof(data)); //存每个点放置士兵数  memset(f,0,sizeof(f)); //dp数组，F[i,j]含义如上述分析  memset(in,0,sizeof(in));//存储每个点的入度，用于找出根节点  memset(son,0,sizeof(son));//存储每个点的儿子节点  memset(len,0,sizeof(len));//存储每个点儿子节点个数  for (int i = 1;i <= n;i++)  {  scanf("%d",&x);  scanf("%d%d",&data[x],&len[x]);  for (int j = 1;j <= len[x];j++)  {  scanf("%d",&son[x][j]);  in[son[x][j]]++;  }  }  for (int i = 1;i <= n;i++)//找到根结点  if (in[i] == 0)  {  root =i;  break;  }  dp(root);  cout<<min(f[root][0],f[root][2])<<endl;  return 0;  } |