备战 NOIP 不可不看的算法

一、数论算法

1．求两数的最大公约数

```
function gcd(a,b:integer):integer;
begin
  if b=0 then gcd:=a
    else gcd:=gcd (b,a mod b);
end ;
```

2．求两数的最小公倍数

```
function lcm(a,b:integer):integer;
begin
  if a<b then swap(a,b);
  lcm:=a;
  while lcm mod b>0 do inc(lcm,a);
end;
```

3．素数的求法

A.小范围内判断一个数是否为质数：

```
function prime (n: integer): Boolean;
  var I: integer;
  begin
    for I:=2 to trunc(sqrt(n)) do
    if n mod I=0 then begin
      prime:=false; exit;
    end;
      prime:=true;
  end;
```

B.判断 longint 范围内的数是否为素数（包含求 50000 以内的素数表）：

```
  procedure getprime;
    var
    i,j:longint;
    p:array[1..50000] of boolean;
    begin
      fillchar(p,sizeof(p),true);
p[1];=false;
i:=2;
while i<50000 do begin
  if p then begin
    j:=i*2;
    while j<50000 do begin
    p[j]:=false;
    inc(j,i);
    end;
  end;
```

```
    inc(i);
    end;
    l:=0;
    for i:=1 to 50000 do
    if p then begin
       inc(l);pr[l]:=i;
    end;
end;{getprime}

function prime(x:longint):integer;
       var i:integer;
       begin
          prime:=false;
for i:=1 to l do
    if pr>=x then break
       else if x mod pr=0 then exit;
prime:=true;
end;{prime}
```

二、图论算法

1．最小生成树

A.Prim 算法：

```
    procedure prim(v0:integer);
       var
          lowcost,closest:array[1..maxn] of integer;
i,j,k,min:integer;
       begin
          for i:=1 to n do begin
    lowcost:=cost[v0,i];
    closest:=v0;
    end;
for i:=1 to n-1 do begin
    {寻找离生成树最近的未加入顶点 k}
    min:=maxlongint;
    for j:=1 to n do
       if (lowcost[j]<min) and (lowcost[j]<>0) then begin
       min:=lowcost[j];
       k:=j;
       end;
    lowcost[k]:=0; {将顶点 k 加入生成树}
       {生成树中增加一条新的边 k 到 closest[k]}
    {修正各点的 lowcost 和 closest 值}
    for j:=1 to n do
       if cost[k,j]<lwocost[j] then begin
       lowcost[j]:=cost[k,j];
```

```
        closest[j]:=k;
        end;
    end;
end;{prim}
```

B.Kruskal 算法：(贪心)

按权值递增顺序删去图中的边，若不形成回路则将此边加入最小生成树。

```
function find(v:integer):integer; {返回顶点 v 所在的集合}
var i:integer;
begin
    i:=1;
    while (i<=n) and (not v in vset) do inc(i);
    if i<=n then find:=i else find:=0;
end;
procedure kruskal;
var
    tot,i,j:integer;
begin
    for i:=1 to n do vset:=;{初始化定义 n 个集合，第 I 个集合包含一个元素 I}
p:=n-1; q:=1; tot:=0; {p 为尚待加入的边数，q 为边集指针}
sort;
{对所有边按权值递增排序，存于 e[I]中，e[I].v1 与 e[I].v2 为边 I 所连接的两个顶点的序号，
e[I].len 为第 I 条边的长度}
    while p>0 do begin
        i:=find(e[q].v1);j:=find(e[q].v2);
        if i<>j then begin
        inc(tot,e[q].len);
        vset:=vset+vset[j];vset[j]:=[];
        dec(p);
        end;
        inc(q);
    end;
    writeln(tot);
end;
```

2.最短路径

A.标号法求解单源点最短路径：

```
    var
        a:array[1..maxn,1..maxn] of integer;
        b:array[1..maxn] of integer; {b 指顶点 i 到源点的最短路径}
        mark:array[1..maxn] of boolean;
    procedure bhf;
        var
        best,best_j:integer;
        begin
        fillchar(mark,sizeof(mark),false);
```

```
  mark[1]:=true; b[1]:=0;{1 为源点}
  repeat
    best:=0;
      for i:=1 to n do
      If mark then {对每一个已计算出最短路径的点}
      for j:=1 to n do
        if (not mark[j]) and (a[i,j]>0) then
        if (best=0) or (b+a[i,j]<best) then begin
        best:=b+a[i,j]; best_j:=j;
      end;
    if best>0 then begin
      b[best_j]:=best；mark[best_j]:=true;
    end;
    until best=0;
    end;{bhf}
  B.Floyed 算法求解所有顶点对之间的最短路径：
    procedure floyed;
    begin
for I:=1 to n do
for j:=1 to n do
if a[I,j]>0 then p[I,j]:=I else p[I,j]:=0; {p[I,j]表示 I 到 j 的最短路径上 j 的前驱结点}
  for k:=1 to n do {枚举中间结点}
  for i:=1 to n do
    for j:=1 to n do
      if a[i,k]+a[j,k]<a[i,j] then begin
    a[i,j]:=a[i,k]+a[k,j];
        p[I,j]:=p[k,j];
    end;
    end;
C. Dijkstra 算法：
var
    a:array[1..maxn,1..maxn] of integer;
    b,pre:array[1..maxn] of integer; {pre 指最短路径上 I 的前驱结点}
    mark:array[1..maxn] of boolean;
procedure dijkstra(v0:integer);
begin
  fillchar(mark,sizeof(mark),false);
  for i:=1 to n do begin
    d:=a[v0,i];
    if d<>0 then pre:=v0 else pre:=0;
  end;
  mark[v0]:=true;
  repeat    {每循环一次加入一个离 1 集合最近的结点并调整其他结点的参数}
    min:=maxint; u:=0; {u 记录离 1 集合最近的结点}
```

```
      for i:=1 to n do
      if (not mark) and (d<min) then begin
         u:=i; min:=d;
      end;
      if u<>0 then begin
      mark:=true;
      for i:=1 to n do
         if (not mark) and (a[u,i]+d<d) then begin
         d:=a[u,i]+d;
         pre:=u;
      end;
      end;
   until u=0;
end;
```

3.计算图的传递闭包

```
Procedure Longlink;
Var
T:array[1..maxn,1..maxn] of boolean;
Begin
Fillchar(t,sizeof(t),false);
For k:=1 to n do
For I:=1 to n do
   For j:=1 to n do T[I,j]:=t[I,j] or (t[I,k] and t[k,j]);
End;
```

4．无向图的连通分量

A.深度优先

```
procedure dfs ( now,color: integer);
   begin
      for i:=1 to n do
      if a[now,i] and c=0 then begin {对结点 I 染色}
         c:=color;
         dfs(I,color);
      end;
end;
```

B 宽度优先（种子染色法）

5．关键路径

几个定义： 顶点 1 为源点，n 为汇点。

a. 顶点事件最早发生时间 Ve[j], Ve [j] = max{ Ve [j] + w[I,j] },其中 Ve (1) = 0;

b. 顶点事件最晚发生时间 Vl[j], Vl [j] = min{ Vl[j] – w[I,j] },其中 Vl(n) = Ve(n);

c. 边活动最早开始时间 Ee[I], 若边 I 由<j,k>表示，则 Ee[I] = Ve[j];

d. 边活动最晚开始时间 El[I], 若边 I 由<j,k>表示，则 El[I] = Vl[k] – w[j,k];

若 Ee[j] = El[j] ，则活动 j 为关键活动，由关键活动组成的路径为关键路径。

求解方法：

a. 从源点起 topsort,判断是否有回路并计算 Ve;

b. 从汇点起 topsort,求 Vl;
c. 算 Ee 和 El;
6．拓扑排序
找入度为 0 的点，删去与其相连的所有边，不断重复这一过程。
例 寻找一数列，其中任意连续 p 项之和为正，任意 q 项之和为负，若不存在则输出 NO.
7.回路问题
Euler 回路(DFS)
定义：经过图的每条边仅一次的回路。（充要条件：图连同且无奇点）
Hamilton 回路
定义：经过图的每个顶点仅一次的回路。
一笔画
充要条件：图连通且奇点个数为 0 个或 2 个。
9．判断图中是否有负权回路 Bellman-ford 算法
x[I],y[I],t[I]分别表示第 I 条边的起点，终点和权。共 n 个结点和 m 条边。

```
  procedure bellman-ford
  begin
for I:=0 to n-1 do d[I]:=+infinitive;
d[0]:=0;
for I:=1 to n-1 do
for j:=1 to m do {枚举每一条边}
  if d[x[j]]+t[j]<d[y[j]] then d[y[j]]:=d[x[j]]+t[j];
for I:=1 to m do
if d[x[j]]+t[j]<d[y[j]] then return false else return true;
  end;
```

10．第 n 最短路径问题
*第二最短路径：每举最短路径上的每条边，每次删除一条，然后求新图的最短路径，取这些路径中最短的一条即为第二最短路径。
*同理，第 n 最短路径可在求解第 n-1 最短路径的基础上求解。
三、背包问题
*部分背包问题可有贪心法求解：计算 Pi/Wi
数据结构：
w:第 i 个背包的重量；
p:第 i 个背包的价值；
1．0-1 背包： 每个背包只能使用一次或有限次(可转化为一次)：
A.求最多可放入的重量。
NOIP2001 装箱问题
有一个箱子容量为 v(正整数，o≤v≤20000)，同时有 n 个物品(o≤n≤30)，每个物品有一个体积 (正整数)。要求从 n 个物品中，任取若干个装入箱内，使箱子的剩余空间为最小。
l 搜索方法

```
  procedure search(k,v:integer); {搜索第 k 个物品，剩余空间为 v}
  var i,j:integer;
  begin
    if v<best then best:=v;
    if v-(s[n]-s[k-1])>=best then exit; {s[n]为前 n 个物品的重量和}
```

```
    if k<=n then begin
    if v>w[k] then search(k+1,v-w[k]);
    search(k+1,v);
    end;
  end;
```

l DP

F[I,j]为前 i 个物品中选择若干个放入使其体积正好为 j 的标志，为布尔型。

实现:将最优化问题转化为判定性问题

f [I, j] = f [i-1, j-w] (w[I]<=j<=v)　　边界：f[0,0]:=true.

```
For I:=1 to n do
For j:=w[I] to v do F[I,j]:=f[I-1,j-w[I]];
```

优化：当前状态只与前一阶段状态有关，可降至一维。

```
F[0]:=true;
For I:=1 to n do begin
F1:=f;
For j:=w[I] to v do
If f[j-w[I]] then f1[j]:=true;
F:=f1;
End;
```

B.求可以放入的最大价值。

F[I,j] 为容量为 I 时取前 j 个背包所能获得的最大价值。

F [i,j] = max { f [i – w [j], j-1] + p [j], f[i,j-1] }

C.求恰好装满的情况数。

DP:

```
Procedure update;
var j,k:integer;
begin
  c:=a;
  for j:=0 to n do
    if a[j]>0 then
      if j+now<=n then inc(c[j+now],a[j]);
  a:=c;
end;
```

2．可重复背包

A 求最多可放入的重量。

F[I,j]为前 i 个物品中选择若干个放入使其体积正好为 j 的标志，为布尔型。

状态转移方程为

f[I,j] = f [I-1, j – w[I]*k] (k=1.. j div w[I])

B.求可以放入的最大价值。

USACO 1.2 Score Inflation

进行一次竞赛，总时间 T 固定，有若干种可选择的题目，每种题目可选入的数量不限，每种题目有一个 ti（解答此题所需的时间）和一个 si（解答此题所得的分数），现要选择若干题目，使解这些题的总时间在 T 以内的前提下，所得的总分最大，求最大的得分。

*易想到：

f[i,j] = max { f [i- k*w[j], j-1] + k*p[j] } (0<=k<= i div w[j])
其中 f[i,j]表示容量为 i 时取前 j 种背包所能达到的最大值。
*实现：

```
Begin
FillChar(f,SizeOf(f),0);
For i:=1 To M Do
For j:=1 To N Do
  If i-problem[j].time>=0 Then
  Begin
    t:=problem[j].point+f[i-problem[j].time];
    If t>f Then f:=t;
  End;
Writeln(f[M]);
End.
```

C.求恰好装满的情况数。
Ahoi2001 Problem2
求自然数 n 本质不同的质数和的表达式的数目。
思路一，生成每个质数的系数的排列，在一一测试，这是通法。

```
procedure try(dep:integer);
  var i,j:integer;
  begin
    cal; {此过程计算当前系数的计算结果，now 为结果}
    if now>n then exit; {剪枝}
    if dep=l+1 then begin {生成所有系数}
    cal;
    if now=n then inc(tot);
    exit;
    end;
    for i:=0 to n div pr[dep] do begin
    xs[dep]:=i;
    try(dep+1);
    xs[dep]:=0;
    end;
  end;
```

思路二，递归搜索效率较高

```
procedure try(dep,rest:integer);
  var i,j,x:integer;
  begin
    if (rest<=0) or (dep=l+1) then begin
    if rest=0 then inc(tot);
    exit;
    end;
    for i:=0 to rest div pr[dep] do
    try(dep+1,rest-pr[dep]*i);
```

```
  end;
{main: try(1,n); }
```

思路三：可使用动态规划求解

USACO1.2 money system

V 个物品，背包容量为 n，求放法总数。

转移方程：

```
Procedure update;
var j,k:integer;
begin
  c:=a;
  for j:=0 to n do
    if a[j]>0 then
    for k:=1 to n div now do
      if j+now*k<=n then inc(c[j+now*k],a[j]);
  a:=c;
end;
{main}
begin
read(now); {读入第一个物品的重量}
i:=0;    {a 为背包容量为 i 时的放法总数}
while i<=n do begin
a:=1; inc(i,now); end; {定义第一个物品重的整数倍的重量 a 值为 1，作为初值}
for i:=2 to v do
begin
  read(now);
  update; {动态更新}
end;
writeln(a[n]);
```

四、排序算法

1.快速排序：

```
procedure qsort(l,r:integer);
  var i,j,mid:integer;
  begin
    i:=l;j:=r; mid:=a[(l+r) div 2]; {将当前序列在中间位置的数定义为中间数}
  repeat
    while a<mid do inc(i); {在左半部分寻找比中间数大的数}
    while a[j]>mid do dec(j);{在右半部分寻找比中间数小的数}
    if i<=j then begin {若找到一组与排序目标不一致的数对则交换它们}
    swap(a,a[j]);
    inc(i);dec(j); {继续找}
    end;
  until i>j;
  if l<j then qsort(l,j); {若未到两个数的边界，则递归搜索左右区间}
  if i<r then qsort(i,r);
```

```
end;{sort}
```
B.插入排序：
思路：当前 a[1]..a[i-1]已排好序了，现要插入 a 使 a[1]..a 有序。
```
procedure insert_sort;
var i,j:integer;
begin
   for i:=2 to n do begin
      a[0]:=a;
      j:=i-1;
      while a[0]<a[j] do begin
      a[j+1]:=a[j];
j:=j-1;
      end;
      a[j+1]:=a[0];
   end;
end;{inset_sort}
```
C.选择排序：
```
procedure sort;
   var i,j,k:integer;
   begin
   for i:=1 to n-1 do
      for j:=i+1 to n do
         if a>a[j] then swap(a,a[j]);
   end;
```
D. 冒泡排序
```
procedure bubble_sort;
   var i,j,k:integer;
   begin
   for i:=1 to n-1 do
      for j:=n downto i+1 do
      if a[j]<a[j-1] then swap( a[j],a[j-1]); {每次比较相邻元素的关系}
end;
```
E.堆排序：
```
procedure sift(i,m:integer);{调整以 i 为根的子树成为堆,m 为结点总数}
var k:integer;
begin
   a[0]:=a; k:=2*i;{在完全二叉树中结点 i 的左孩子为 2*i,右孩子为 2*i+1}
   while k<=m do begin
      if (k<m) and (a[k]<a[k+1]) then inc(k);{找出 a[k]与 a[k+1]中较大值}
   if a[0]<a[k] then begin a:=a[k];i:=k;k:=2*i; end
   else k:=m+1;
   end;
   a:=a[0]; {将根放在合适的位置}
end;
```

```
procedure heapsort;
var
   j:integer;
begin
   for j:=n div 2 downto 1 do sift(j,n);
   for j:=n downto 2 do begin
      swap(a[1],a[j]);
      sift(1,j-1);
   end;
end;
```

F. 归并排序

```
{a 为序列表，tmp 为辅助数组}
procedure merge(var a:listtype; p,q,r:integer);
{将已排序好的子序列 a[p..q]与 a[q+1..r]合并为有序的 tmp[p..r]}
var I,j,t:integer;
   tmp:listtype;
begin
   t:=p;i:=p;j:=q+1;{t 为 tmp 指针，I,j 分别为左右子序列的指针}
   while (t<=r) do begin
      if (i<=q){左序列有剩余} and ((j>r) or (a<=a[j])) {满足取左边序列当前元素的要求}
      then begin
            tmp[t]:=a; inc(i);
      end
      else begin
            tmp[t]:=a[j];inc(j);
      end;
   inc(t);
end;
for i:=p to r do a:=tmp;
end;{merge}
procedure merge_sort(var a:listtype; p,r: integer); {合并排序 a[p..r]}
var q:integer;
begin
   if p<>r then begin
      q:=(p+r-1) div 2;
      merge_sort (a,p,q);
      merge_sort (a,q+1,r);
      merge (a,p,q,r);
   end;
end;
{main}
begin
merge_sort(a,1,n);
end.
```

G.基数排序

思想：对每个元素按从低位到高位对每一位进行一次排序

五、高精度计算

高精度数的定义：

```
type
   hp=array[1..maxlen] of integer;
```

1．高精度加法

```
procedure plus ( a,b:hp; var c:hp);
var i,len:integer;
begin
fillchar(c,sizeof(c),0);
if a[0]>b[0] then len:=a[0] else len:=b[0];
for i:=1 to len do begin
   inc(c,a+b);
if c>10 then begin dec(c,10); inc(c[i+1]); end; {进位}
end;
if c[len+1]>0 then inc(len);
c[0]:=len;
end;{plus}
```

2．高精度减法

```
procedure substract(a,b:hp;var c:hp);
var i,len:integer;
begin
   fillchar(c,sizeof(c),0);
   if a[0]>b[0] then len:=a[0] else len:=b[0];
   for i:=1 to len do begin
      inc(c,a-b);
      if c<0 then begin inc(c,10);dec(c[i+1]); end;
      while (len>1) and (c[len]=0) do dec(len);
   c[0]:=len;
end;
```

3．高精度乘以低精度

```
procedure multiply(a:hp;b:longint;var c:hp);
var i,len:integer;
begin
   fillchar(c,sizeof(c),0);
   len:=a[0];
   for i:=1 to len do begin
      inc(c,a*b);
      inc(c[i+1],(a*b) div 10);
      c:=c mod 10;
   end;
   inc(len);
   while (c[len]>=10) do begin {处理最高位的进位}
```

```
        c[len+1]:=c[len] div 10;
        c[len]:=c[len] mod 10;
        inc(len);
    end;
    while (len>1) and (c[len]=0) do dec(len); {若不需进位则调整 len}
    c[0]:=len;
end;{multiply}
```

4．高精度乘以高精度

```
procedure high_multiply(a,b:hp; var c:hp}
var i,j,len:integer;
begin
    fillchar(c,sizeof(c),0);
    for i:=1 to a[0] do
        for j:=1 to b[0] do begin
        inc(c[i+j-1],a*b[j]);
    inc(c[i+j],c[i+j-1] div 10);
    c[i+j-1]:=c[i+j-1] mod 10;
        end;
    len:=a[0]+b[0]+1;
    while (len>1) and (c[len]=0) do dec(len);
    c[0]:=len;
end;
```

5．高精度除以低精度

```
procedure devide(a:hp;b:longint; var c:hp; var d:longint);
{c:=a div b; d:= a mod b}
var i,len:integer;
begin
    fillchar(c,sizeof(c),0);
    len:=a[0]; d:=0;
    for i:=len downto 1 do begin
        d:=d*10+a;
        c:=d div b;
        d:=d mod b;
    end;
    while (len>1) and (c[len]=0) then dec(len);
    c[0]:=len;
end;
```

6．高精度除以高精度

```
procedure high_devide(a,b:hp; var c,d:hp);
var
    i,len:integer;
begin
    fillchar(c,sizeof(c),0);
    fillchar(d,sizeof(d),0);
```

```
  len:=a[0];d[0]:=1;
  for i:=len downto 1 do begin
    multiply(d,10,d);
    d[1]:=a;
    while(compare(d,b)>=0) do {即 d>=b}
    begin
    Subtract(d,b,d);
    inc(c);
    end;
  end;
  while(len>1)and(c.s[len]=0) do dec(len);
  c.len:=len;
end;
```

六、 树的遍历

1．已知前序中序求后序

```
procedure Solve(pre,mid:string);
var i:integer;
begin
  if (pre='') or (mid='') then exit;
  i:=pos(pre[1],mid);
  solve(copy(pre,2,i),copy(mid,1,i-1));
  solve(copy(pre,i+1,length(pre)-i),copy(mid,i+1,length(mid)-i));
  post:=post+pre[1]; {加上根，递归结束后 post 即为后序遍历}
end;
```

2．已知中序后序求前序

```
procedure Solve(mid,post:string);
var i:integer;
begin
  if (mid='') or (post='') then exit;
  i:=pos(post[length(post)],mid);
  pre:=pre+post[length(post)]; {加上根，递归结束后 pre 即为前序遍历}
  solve(copy(mid,1,I-1),copy(post,1,I-1));
  solve(copy(mid,I+1,length(mid)-I),copy(post,I,length(post)-i));
end;
```

3．已知前序后序求中序的一种

```
function ok(s1,s2:string):boolean;
var i,l:integer;     p:boolean;
begin
  ok:=true;
  l:=length(s1);
  for i:=1 to l do begin
    p:=false;
    for j:=1 to l do
    if s1=s2[j] then p:=true;
```

```
      if not p then begin ok:=false;exit;end;
   end;
end;
procedure solve(pre,post:string);
   var i:integer;
   begin
   if (pre='') or (post='') then exit;
   i:=0;
   repeat
      inc(i);
   until ok(copy(pre,2,i),copy(post,1,i));
   solve(copy(pre,2,i),copy(post,1,i));
   midstr:=midstr+pre[1];
   solve(copy(pre,i+2,length(pre)-i-1),copy(post,i+1,length(post)-i-1));
   end;
```

七 进制转换

1 任意正整数进制间的互化

除 n 取余

2 实数任意正整数进制间的互化

乘 n 取整

3 负数进制：

设计一个程序，读入一个十进制数的基数和一个负进制数的基数，并将此十进制数转换为此负 进制下的数：-R∈{-2，-3，-4,....-20}

八 全排列与组合的生成

1 排列的生成：（1..n）

```
procedure solve(dep:integer);
   var
      i:integer;
   begin
      if dep=n+1 then begin writeln(s);exit; end;
      for i:=1 to n do
      if not used then begin
         s:=s+chr(i+ord('0'));used:=true;
         solve(dep+1);
         s:=copy(s,1,length(s)-1); used:=false;
      end;
   end;
```

2 组合的生成(1..n 中选取 k 个数的所有方案)

```
procedure solve(dep,pre:integer);
   var
      i:integer;
   begin
      if dep=k+1 then begin writeln(s);exit; end;
      for i:=1 to n do
```

```
      if (not used) and (i>pre) then begin
         s:=s+chr(i+ord('0'));used:=true;
         solve(dep+1,i);
         s:=copy(s,1,length(s)-1); used:=false;
      end;
end;
```

九.查找算法

1 折半查找

```
function binsearch(k:keytype):integer;
var low,hig,mid:integer;
begin
   low:=1;hig:=n;
   mid:=(low+hig) div 2;
   while (a[mid].key<>k) and (low<=hig) do begin
      if a[mid].key>k then hig:=mid-1
      else low:=mid+1;
      mid:=(low+hig) div 2;
   end;
   if low>hig then mid:=0;
   binsearch:=mid;
end;
```

2 树形查找

二叉排序树：每个结点的值都大于其左子树任一结点的值而小于其右子树任一结点的值。

查找

```
function treesrh(k:keytype):pointer;
var q:pointer;
begin
   q:=root;
   while (q<>nil) and (q^.key<>k) do
      if k<q^.key then q:=q^.left
      else q:=q^.right;
   treesrh:=q;
end;
```

十、贪心

*会议问题

（1） n 个活动每个活动有一个开始时间和一个结束时间，任一时刻仅一项活动进行，求满足活动数最多的情况。

解：按每项活动的结束时间进行排序，排在前面的优先满足。

（2）会议室空闲时间最少。

（3）每个客户有一个愿付的租金，求最大利润。

（4）共 R 间会议室，第 i 个客户需使用 i 间会议室，费用相同，求最大利润。

十一、回溯法框架

1. n 皇后问题

```
procedure try(i:byte);
```

```
var j:byte;
begin
  if i=n+1 then begin print;exit;end;
    for j:=1 to n do
    if a and b[j+i] and c[j-i] then begin
      x:=j;
    a[j]:=false; b[j+i]:=false; c[j-i]:=false;
    try(i+1);
    a[j]:=true; b[i+j]:=true; c[j-i]:=true;
  end;
end;
```

2.Hanoi Tower h(n)=2*h(n-1)+1 h(1)=1

初始所有铜片都在 a 柱上

```
procedure hanoi(n,a,b,c:byte); {将第 n 块铜片从 a 柱通过 b 柱移到 c 柱上}
begin
if n=0 then exit;
hanoi(n-1,a,c,b); {将上面的 n-1 块从 a 柱通过 c 柱移到 b 柱上}
write(n,'moved from',a,'to',c);
hanoi(n-1,b,a,c);{ 将 b 上的 n-1 块从 b 柱通过 a 柱移到 c 柱上
end;
```

初始铜片分布在 3 个柱上，给定目标柱 goal

h[1..3,0..n]存放三个柱的状态，now 与 nowp 存最大的不到位的铜片的柱号和编号,h[I,0]存第 I 个柱上的个数。

```
Procedure move(k,goal:integer); {将最大不到位的 k 移到目标柱 goal 上}
Begin
If k=0 then exit;
For I:=1 to 3 do
  For j:=1 to han[I,0] do
  If h[I,j]=k then begin now:=I;nowp:=j; end; {找到 k 的位置}
If now<>goal then begin {若未移到目标}
  Move(k-1,6-now-goal); {剩下的先移到没用的柱上}
  Writeln(k moved from now to goal);
  H[goal,h[goal,0]+1]:=h[now,nowp]; h[now,nowp]:=0;
  Inc(h[goal,0]); dec(h[now,0]);
  Move(k-1,goal); {剩下的移到目标上}
End;
```

十二、DFS 框架

NOIP2001 数的划分

```
procedure work(dep,pre,s:longint); {入口为 work(1,1,n)}
{dep 为当前试放的第 dep 个数,pre 为前一次试放的数,s 为当前剩余可分的总数}
var j:longint;
begin
  if dep=n then begin
    if s>=pre then inc(r); exit;
```

```
    end;
    for j:=pre to s div 2 do work(dep+1,j,s-j);
end;
```

类似：

```
procedure try(dep:integer);
    var i:integer;
    begin
        if dep=k then begin
        if tot>=a[dep-1] then inc(sum);
            exit; end;
        for i:=a[dep-1] to tot div 2 do begin
        a[dep]:=i; dec(tot,i);
try(dep+1);
        inc(tot,i);
        end;
    end;{try}
```

十三、BFS 框架

IOI94 房间问题

```
head:=1; tail:=0;
while tail<head do begin
inc(tail);
for k:=1 to n do
    if k 方向可扩展 then begin
        inc(head);
        list[head].x:=list[tail].x+dx[k]; {扩展出新结点 list[head]}
        list[head].y:=list[tail].y+dy[k];
        处理新结点 list[head];
    end;
end;
```

十五、数据结构相关算法

1．链表的定位函数 loc(I:integer):pointer; {寻找链表中的第 I 个结点的指针}

```
procedure loc(L:linklist; I:integer):pointer;
var p:pointer;
j:integer;
begin
p:=L.head; j:=0;
if (I>=1) and (I<=L.len) then
while j<I do begin p:=p^.next; inc(j); end;
loc:=p;
end;
```

2．单链表的插入操作

```
procedure insert(L:linklist; I:integer; x:datatype);
var p,q:pointer;
begin
```

```
p:=loc(L,I);
new(q);
q^.data:=x;
q^.next:=p^.next;
p^.next:=q;
inc(L.len);
end;
```

3．单链表的删除操作

```
procedure delete(L:linklist; I:integer);
var p,q:pointer;
begin
p:=loc(L,I-1);
q:=p^.next;
p^.next:=q^.next;
dispose(q);
dec(L.len);
end;
```

4．双链表的插入操作（插入新结点 q）

```
p:=loc(L,I);
new(q);
q^.data:=x;
q^.pre:=p;
q^.next:=p^.next;
p^.next:=q;
q^.next^.pre:=q;
```

5．双链表的删除操作

```
p:=loc(L,I); {p 为要删除的结点}
p^.pre^.next:=p^.next;
p^.next^.pre:=p^.pre;
dispose(p);
```

关键路径（最长路经）：

```
var a,b:array [1..10,1..10] of integer;
   n,last,out:integer;
   q,c:array [1..10] of integer;
   o:set of 1..10;
procedure init;
var i,j:integer;
begin
readln(n);
for i:=1 to n do
   for j:=1 to n do
      read(a[i,j]);
last:=0;
o:=[]; out:=0;
```

```
b:=a;
end;
procedure sort;
var i,j:integer;
      p:boolean;
begin
while out<>n do begin
   for i:=1 to n do
   if not (i in o) then begin
      p:=true;
      for j:=1 to n do
      if a[j,i]=1 then begin
         p:=false;
         break;
      end;
      if p then begin
      inc(last);
      q[last]:=i;
      inc(out);
o:=o+;
      fillchar(a,sizeof(a),0);
      end;
   end;
end;
end;
procedure work_1;
var i,j,t,k:integer;
begin
a:=b; c[1]:=0;
for i:=1 to n do begin
   k:=0;
   for j:=1 to i-1 do
      if (a[q[j],q]>0) and (a[q[j],q]+c[q[j]]>k)
      then k:=a[q[j],q]+c[q[j]];
   c[q]:=k;
end;
end;
procedure work_2;
var i,j,k:integer;
begin
writeln(q[n]);
for i:=n-1 downto 1 do begin
   k:=maxint;
   for j:=i+1 to n do
```

```
      if (a[q,q[j]]>0) and (c[q[j]]-a[q,q[j]]<k) then k:=c[q[j]]-a[q,q[j]];
   if c[q]=k then writeln(q,' ');
   c[q]:=k;
end;
end;
begin
init;
sort;
work_1;
work_2;
end.
```

拓扑排序：

```
var a:array [1..100,1..100] of 0..1;
   n:integer;
   p:set of 1..100;
procedure init;
var i,j,k:integer;
begin
fillchar(a,sizeof(a),0);
readln(n);
for i:=1 to n do begin
   read(k);
   while k<>0 do begin
      a[i,k]:=1;
      read(k);
   end;
end;
p:=[];
end;
procedure search;
var i,j,t,sum,printed:integer;
begin
printed:=0;
while printed<n do
   for i:=1 to n do begin
      sum:=0;
      for j:=1 to n do sum:=sum+a[j,i];
      if (sum=0) and not(i in p) then begin
      write(i,' ');
      p:=p+;
      inc(printed);
      for t:=1 to n do a[i,t]:=0;
      end;
   end;
```

```
end;
begin
init;
search;
end.
```