Pan-algorithm Contest Code Spells 4.1 [Code Recipe Printable Version]

2012-10-09

Content

Sort(排序方法)

Heap Sort

code:

```
/*
 a heap sort sample
 堆排序示例
 write by gestapolur
 2010-9-2
*/
#include<iostream>
#define MAXN 100
using std::cin;
using std::cout;
int a[MAXN],n,heapsize;
```

```
/*维护最大堆*/
void maxheapify(int i)
{
 int l,r,largest,t;
 l=i<<1;
 r=(i<<1)+1;
 if(l<=heapsize and a[i]<a[l])
  largest=l;
 else
  largest=i;
 if(r<=heapsize and a[r]>a[largest])
  largest=r;
 if(largest != i)
  {
   t=a[i];a[i]=a[largest];a[largest]=t;
   maxheapify(largest);
  }
 return ;
}

/*建立最大堆*/
void BuildMaxHeap()
{
 int i;
 heapsize=n;
 for(i=n/2;i>=1;i--)
  maxheapify(i);
 return ;
}

/*堆排序*/
void heapsort()
{
 int i,t;
 BuildMaxHeap();/*建立最大堆*/
 for(i=n;i>=2;i--)
  {
   t=a[1];a[1]=a[i];a[i]=t;/*将 a[i]与 a[1]交换*/
   --heapsize;/*剔除交换后的元素 a[i]*/
   maxheapify(1);/*维护最大堆性质*/
  }
 return ;
}

int main()
{
 int i;

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

```
 heapsort();

 for(i=1;i<=n;i++)
  cout<<a[i]<<" ";
 cout<<"\n";

 return 0;
}
```

Pority Queue

```
/*
 Name: A model of heap priority queue
 Copyright: 1.0
 Author: Gestapolur
 Date: 04-11-08 16:31
*/
#include<iostream>
#define MAXH 100
#define INF 0xffffff
using std::cin;
using std::cout;

int n,heapsize;
int a[MAXH];
```

```
void maxheapify(int i)
 {
 int l,r,largest,t;
 l=i<<1;
 r=(i<<1)+1;
 if(l<=heapsize and a[i]<a[l])
   largest=l;
 else
   largest=i;
 if(r<=heapsize and a[r]>a[largest])
   largest=r;
 if(largest not_eq i)
     {
   t=a[i];a[i]=a[largest];a[largest]=t;
   maxheapify(largest);
     }
 return ;
 }

void BuildMaxHeap()
 {
 int i;
 heapsize=n;
 for(i=n/2;i>=1;i--)
   maxheapify(i);
```

```
 return ;
 }

int ExtractMax()
 {
 int max;
 max=a[1];
 a[1]=a[heapsize];
 heapsize--;
 maxheapify(1);
 return max;
 }

void HeapIncreaseKey(int i,int key)
 {
 int t;
 if(a[i]>key)
   {cout<<"CURRENT VERX BIGGER THAN
KEY\n";return;}
 a[i]=key;
 while(i>1 and a[i>>1]<a[i])
   {t=a[i];a[i]=a[i>>1];a[i>>1]=t;i=i>>1;}
 return ;
 }

void MaxHeapInsert(int key)
 {
 heapsize++;
 a[heapsize]=-INF;
 HeapIncreaseKey(heapsize,key);
 return;
 }

int main()
 {
 int i,key,v;
 cin>>n;
 for(i=1;i<=n;i++)
   cin>>a[i];
 BuildMaxHeap();
 cout<<"Please Insert A Instruction:\n";
 cout<<"1.RETURN MAX VERX\n";
 cout<<"2.EXTRACT MAX VERX\n";
 cout<<"3.INCREASE VERX\n";
 cout<<"4.INSERT NEW VERX\n";
 cout<<"OTHER.EXIT\n";
 while(1)
  {
  cin>>i;
  if(i==1)
   cout<<a[1]<<"\n";
```

```
  else if(i==2)
   cout<<ExtractMax()<<"\n";
  else if(i==3)
   {cin>>v>>key;HeapIncreaseKey(i,key);}
  else if(i==4)
   {cin>>key;MaxHeapInsert(key);}
  else
   break;
  }

 return 0;
}
```

```
public:
 bool operator () ( const node a , const node b ) const
 { return ( a.dist > b.dist ); }
};

priority_queue< node , vector<node> , cmp > min_heap;

//insert
min_heap.push( <argument_to_insert> );
//top
min_heap.top();
//pop top
min_heap.pop()
//check if heap were empty
min_heap.empty();
```

STL 里 heap 和 priority queue 的使用

```
//A example of minimum heap use STL priority queue
class node
{
public:
 <node_data>
//construct function
 node ( <> ) : first_parameter( argument_1 ) , ... {}
};

class cmp//compare function
{
```

Merge sort

code:

```
#include<stdio.h>
#include<stdlib.h>
int a[100000];
int b[100000];
void Mergesort(int L,int R)
{
  int i,j,k;
  if(L>=R)
```

```
  return;
 int mid=(L+R)/2;
 Mergesort(L,mid);
 Mergesort(mid+1,R);
 for(i=L;i<=R;i++)
  b[i]=a[i];
 i=L,j=mid+1;
 for(k=L;k<=R;k++)
 {
  if(i<=mid&&((j>R)||b[i]<b[j]))
    a[k]=b[i++];
  else
    a[k]=b[j++];
 }
}
int main()
{
 freopen("sort.in","r",stdin);
 freopen("sort.out","w",stdout);
 int i,j,n;
 scanf("%d",&n);
 for(i=0;i<n;i++)
  scanf("%d",&a[i]);
 Mergesort(0,n-1);
 for(i=0;i<n;i++)
   printf("%d ",a[i]);
 //system("pause");
}
```

基数排序(Bucket Sort)

```
#include<iostream>
#define MAXN 10001
using namespace std;

int a[ MAXN ] , b[ MAXN ] , r[ MAXN ];
int n,m;

int main()
{
 int i;
 cin>>n;
 for( i = 1 ; i <= n ; ++ i )
  {
   cin>>a[ i ];
   m = m > a[ i ] ? m : a[ i ];
```

```
  }
 //bucket sort
 for( i = 1 ; i <= n ; ++ i )
  ++ b[ a[ i ] ];//统计每个关键字的数量
 for( i = 1 ; i <= m ; ++ i )
  b[ i ] += b[ i - 1 ];/*将关键字按照大小累积，使得越靠后的关键字序数越大
换言之就是使桶中的关键字按照序号递增*/
 for( i = n ; i >= 1 ; -- i )
  r[ b[ a[ i ] ]-- ] = i;
 for( i = 1 ; i <= n ; ++ i )
  cout<<a[ r[ i ] ]<<" ";
 cout<<"\n";
 return 0;
}
```

```
int a[ MAXN ] , b[ MAXN ] , r[ MAXN ];
int n,m;
int main()
{
 int i;
 cin>>n;
 for( i = 1 ; i <= n ; ++ i )
  {
   cin>>a[ i ];
   m = m > a[ i ] ? m : a[ i ];
  }
 //bucket sort
 for( i = 1 ; i <= n ; ++ i )
  ++ b[ a[ i ] ];//统计每个关键字的数量
 for( i = 1 ; i <= m ; ++ i )
  b[ i ] += b[ i - 1 ];/*将关键字按照大小累积，使得越靠后的关键字序数越大
换言之就是使桶中的关键字按照序号递增*/
 for( i = n ; i >= 1 ; -- i )
  r[ b[ a[ i ] ]-- ] = i;
 for( i = 1 ; i <= n ; ++ i )
  cout<<a[ r[ i ] ]<<" ";
 cout<<"\n";
```

```
 return 0;
}
```

bucket sort 关键就在于用累加的方法使得同一关键字的序号和其数量成线序的关系从而完成排序过程。

Data Structure(數據結構)

Hash Functions

“拉鍊哈希”

(TIMUS 1806)

```
/*
use a link list to avoid conflict in hash
*/
int head[ MAXH ] , Next[ MAXH ];

inline int Hash( char *str )
{
int v = 0 , seed = 131;
while( *str )
v = v * seed + *(str ++ );//str[ x ] ++
return ( v & 0x7fffffff ) & MAXH;
}

inline void insert_Hash( int idx )
{
int i , h;
h = Hash( s[ idx ] );
Next[ idx ] = head[ h ];
head[ h ] = idx;
}
```

Graph Theory(图论)

Forward Star(前向星表示法)

一种存储图的方式，维护一个链表 list[i]记录从点 i 出发的所有边的编号。

一个伪代码例子：

```
//添加一条边
add( i , u , v , w )//i 是边的编号
{
   u[ i ] := u , v[ i ] := v , w[ i ] := w
   next[ i ] := head[ u ];
   head[ u ] := i;
}
```

另外对于双向边,可以记录 e 和 e+1 为正向边和反向边，如果知道其中一条边 e，那么另外一条边就是(e - 1 ^ 1) + 1。

SPFA

```
/*
 SPFA
 2011-01-11
 write by Gestapolur
 most case tested
*/
#include<iostream>
#define MAXN 1002
#define INF 2141483647
using namespace std;

int n,m;//verx,edge
int rear,front;
int w[MAXN][MAXN],que[MAXN *
100],mark[MAXN]; // record the number of verx in
queue ;
int pre[MAXN];
bool in[MAXN];
//array w is adj matrix

void SPFA(int s)
{
 int i;
 for(i=1;i<=n;i++) mark[i] = INF;
 front = 0,rear = 1;
 que[rear] = s,mark[s] = 0;
 do{
  ++front;
  in[ que[ front ] ] = false;//notice que[front] here

  for(i=1;i<=n;i++)
   {
    if( w[que[front]][i] > 0 and mark[i] >
mark[que[front]] + w[que[front]][i] )
      {
       mark[i] = mark[que[front]] + w[que[front]][i];
       pre[ i ] = que[ front ];

       if(not in [ i ])
         {
         que[ ++rear ] = i;
         in [ i ] = true;
         }
      }
   }
 }while(front<=rear);

 for(i=1;i<=n;i++)
  cout<<mark[i]<<" ";
 cout<<"\n";
 return ;
}
```

```
int main()
{
 int i,u,v,wi;
 cin>>n>>m;
 for(i = 1 ; i <= m ; ++ i)
  {
   cin>>u>>v>>wi;
   if(w[u][v] not_eq 0)
    w[u][v] = w[u][v] < wi ? w[u][v] : wi ;
   else
    w[u][v] = wi;
  }

 SPFA(1);

 return 0;
}
```

Bellman-Ford

```
/*
 bellman_ford algorithm
 2011-01-11
 write by gestapolur
 mostly cases tested
*/
```

```
#include<iostream>
#define MAXN 1002
#define INF 2141483647
using namespace std;

//adj list
int n,e;//e is the num of edge
int mark[MAXN],u[MAXN * MAXN],v[MAXN *
MAXN],w[MAXN * MAXN];

bool bellman_ford(int s)
{
 int i,j;
 bool sign ;
 //init
 for(i=1;i<=n;i++) mark[i] = INF;
 mark[s] = 0;

 for(i=1;i<n;i++)
  {
   sign = false;
   for(j=1;j<=e;j++)
    if(mark[u[j]] not_eq INF and mark[v[j]] > mark[u[j]]
+ w[j])
      {mark[v[j]] = mark[u[j]] + w[j];sign = true;}
   if(not sign)
    break;
  }
```

```
 //验证是否存在负权回路
 for(i=1;i<=e;i++)
  if(mark[v[i]] > mark[u[i]] + w[i])
    return false;

 return true;
}

int main()
{
 int i;
 cin>>n>>e;
 for(i=1;i<=e;i++)
  cin>>u[i]>>v[i]>>w[i];

 if(bellman_ford(1))
  for(i=1;i<=n;i++)
    cout<<mark[i]<<" ";
 cout<<"\n";

 cin.close();
 cout.close();

 return 0;
}
```

Dijkstra

```
/*
 dijkstra algorithm
 2011-01-11
 write by gestapolur
*/
#include<iostream>
#define INF 2141483647
#define MAXN 1002
using namespace std;

int n,m;
//n are verxs , m are edges
int w[MAXN][MAXN],mark[MAXN];
int pre[MAXN];
bool in[MAXN];

void dijkstra(int s)
{
 int i,j,k,min;
 for(i=1;i<=n;i++) {mark[i] = w[s][i];pre[i] = s;}
 mark[s] = 0;
 pre[s] = 0;
 in[s] = true;

 for(i=1;i<=n;i++)
  {
```

```
    min = INF ;k = 0;
    for(j=1;j<=n;j++)
     if(not in[j] and min > mark[j])
       {k = j;min = mark[j];}

    in[k] = true;

    for(j=1;j<=n;j++)
     if(w[k][j] not_eq INF and mark[j] > mark[k] + w[k]
[j])
       {
         mark[j] = mark[k] + w[k][j];
         pre[ j ] = k;
       }
   }

 //print stage to every point's shortest path ;
 for(i=1;i<=n;i++)
  cout<<mark[i]<<" ";
 cout<<"\n";
 /*
 for(i = 1 ; i <= n ; ++ i)
  cout<<pre[ i ]<<" ";
 cout<<"\n";
 */
 return ;
}
```

```
int main()
{
 int i,u,v,wi;
 cin>>n>>m;
 //init
 for(u=0;u<=n;u++)
  for(v=0;v<=n;v++)
   w[u][v] = INF;

 for(i=1;i<=m;i++)
  {
   cin>>u>>v>>wi;
   w[u][v] = w[u][v] < wi ? w[u][v] : wi;
  }

 dijkstra(1);

 return 0;
}
```

Floyd)

```
{
TIMUS 1004
Hint: use floyd to find minimum circle
write by gestapolur
```

```
2012-07-08
ACCEPTED
}
program timus1004;
const
  MAXN    = 100;
  INF   = 1 shl 28;
var
  n , m , cnt : longint;
  ans         : longint;
  f , w , pre : array[ 1..MAXN , 1..MAXN ] of longint;
  sv          : array[ 1..MAXN ] of longint;

function init() : boolean;
var
  i , sx , sy , sw : longint;
begin
  read( n );
  if n = -1 then
  begin
    readln;
    exit( false );
  end;
  readln( m );

  for sx := 1 to n do
    for sy := 1 to n do
    begin
      w[ sx , sy ] := INF;
      f[ sx , sy ] := INF;
    end;


  for i := 1 to m do begin
    readln( sx , sy , sw );
    if f[ sx , sy ] > sw then begin
      f[ sx , sy ] := sw;
      pre[ sx , sy ] := sx;
    end;
    if f[ sy , sx ] > sw then begin
      f[ sy , sx ] := sw;
      pre[ sy , sx ] := sy;
    end;
    w[ sx , sy ] := f[ sx , sy ];
    w[ sy , sx ] := f[ sy , sx ];
  end;


  exit( true );
end; { init }

procedure floyd();
var
  i , j , k , cur : longint;
begin
  ans := INF;
```

```
  cnt := 0;
  for k := 1 to n do begin
    for i := 1 to k - 1 do
      for j := 1 to k - 1 do
      begin
        if ( i <> j ) and ( w[ k , j ] <> INF ) and ( w[ i , k ]
<> INF ) and ( ans > f[ j , i ] + w[ i , k ] + w[ k , j ] ) then
        begin
          ans := f[ j , i  ] + w[ i , k ] + w[ k , j ];
          cnt := 0;
          cur := i;
          repeat
           inc( cnt );
           sv[ cnt ] := cur;
           cur := pre[ j , cur ];
          until ( cur = i ) or ( cur = j );

          inc( cnt );
          sv[ cnt ] := j;
          inc( cnt );
          sv[ cnt ] := k;
        end;
      end;


    for i := 1 to k do
      for j := 1 to k do
      begin
        if ( i <> j ) and ( f[ i , k ] <> INF ) and ( f[ k , j ] <>
INF ) and ( f[ i , j ] > f[ i , k ] + f[ k , j ] ) then
        begin
          f[ i , j ] := f[ i , k ] + f[ k , j ];
          pre[ i , j ] := pre[ k , j ];
        end;
        if ( f[ j , i ] <> INF ) and ( f[ k , j ] <> INF ) and ( f[ k
, i ] > f[ k , j ] + f[ j , i ] ) then
        begin
          f[ k , i ] := f[ k , j ] + f[ j , i ];
          pre[ k , i ] := pre[ j , i ];
        end;
        if ( f[ i , j ] <> INF ) and ( f[ j , k ] <> INF ) and ( f[ i
, k ] > f[ i , j ] + f[ j , k ] ) then
        begin
          f[ i , k ] := f[ i , j ] + f[ j , k ];
          pre[ i , k ] := pre[ j , k ];
        end;
      end;
  end;
end; { floyd }

procedure out();
var i : longint ;
begin
  if cnt = 0 then begin
    writeln('No solution.');
    exit();
  end;
  for i := 1 to cnt - 1 do
    write( sv[ i ] , ' ' );
  writeln( sv[ cnt ] );
```

```
end; { out }

begin
  while not eof do begin
    if init = false then
      break;
    floyd;
    out;
  end;
end.
```

E.g : (TIMUS 1004)

prim

```
/*
 prim algorthim sample programme
 prim 算法的演示程序
 这里使用无向图
 write by gestapolur
 2010-9-2
*/
#include<iostream>
#define MAXN 2001
#define INF 999999999
using std::cin;
using std::cout;
int n,e;
int w[MAXN][MAXN];/*两点之间边的权值*/
int mincount[MAXN];/*从初始顶点到该顶点的最小权值*/

void prim(int s)
{
 int i,j,count=0,min;/*count 为生成树所有边的权值和*/
 int k;

 for(i=1;i<=n;i++)
   mincount[i]=w[s][i];/*初始化，设 w(1,i)为初始点 k 到 i 的最小权值，如果没有则为+∞*/
 mincount[s]=0;

 for(i=1;i<n;i++)
  {
   min=INF;
   for(j=1;j<=n;j++)
    if(mincount[j]!=0 and mincount[j]<min)
      /*如果该节点没有被加入到最小生成树中并且该点权值为当前最小*/
      {
       min=mincount[j];
       k=j;
       /*记录该点*/
      }
```

```
 mincount[k]=0;/*把这个点加入到最小生成树中*/
 count+=min; /*将这条边加入到最小生成树中*/

 for(j=1;j<n;j++) /*修正初始点到每个点的最小权值*/
  if(w[k][j]<mincount[j])
   mincount[j]=w[k][j];
  }

 cout<<count<<"\n";
 return ;
}

/*演示过程,从标号为 1 的顶点开始构造生成树*/
int main()
 {
 int i,j,tx,ty;
 for(i=0;i<=MAXN;i++)
  for(j=0;j<=MAXN;j++)
    w[i][j]=INF;

 cin>>n>>e;
 for(i=1;i<=e;i++)
      {
    cin>>tx>>ty>>w[tx][ty];
    w[ty][tx]=w[tx][ty];
      }
 prim(1);
 return 0;
}
```

Kruskal

```
/*
 Kruskal algorthim to MST
 求最小生成树的 kurskal 算法
 这个图不同于样例中的无向图，这是一个有向图
 write by gestapolur
 2010-9-1
*/
#include<iostream>
#define MAXN 2000
#define INF 99999999
using std::cin;
using std::cout;
int n,e;/*点数和边数*/
int x[MAXN],y[MAXN],w[MAXN];/*x 为边的起点，y 为终点，
w 为边的权值*/
int father[MAXN];/*每个节点属于的父亲集合*/

/*给边权排序*/
void sort(int i,int j)
{
 if(i>=j)
```

```
  return ;
 int m,n,t,k;
 m=i,n=j,k=w[ (i+j) >> 1];
 while(m<=n)
  {
   while(w[m]<k)
    m++;
   while(w[n]>k)
    n--;
   if(m<=n)
    {
      t=x[m];
      x[m]=x[n];
      x[n]=t;
      t=y[m];
      y[m]=y[n];
      y[n]=t;
      t=w[m];
      w[m]=w[n];
      w[n]=t;
      m++;
      n--;
    }
  }
 sort(i,n);
 sort(m,j);
}
```

```
int getfather(int x)/*查找点 x 属于的集合*/
{
 if(x==father[x])
  return x;
 father[x]=getfather(father[x]);/*查找过程中更新 x 的父亲集合，
相当于并查集归并过程*/
 return (father[x]);
}

void kruskal()
 {
  int i,p,ans;/*p:已经加入的边数,ans:加入边的边权和*/
  for(i=1;i<=n;i++)
   father[i]=i;/*初始化点的集合*/

  p=1,ans=0;

  for(i=1;i<=e;i++)
   {
    if(getfather(x[i]) not_eq getfather(y[i]))
     /*如果边的两点不在同一个集合内，则归并两个集合*/
     {
       ans+=w[i];/*加入这条边并统计其权值*/
       father[getfather(x[i])]=y[i];/*将两个集合并为同一父亲集
合*/
```

```
        p++;
        if(p == n)/*如果最小生成树中的顶点数等于全部顶点数 - 1，
这里为加快计算速度 p 初始化为 1 所以 p=n 结束计算*/
          {cout<<ans<<"\n";return ;}
       }
     }
   return ;
 }

int main()
 {
 int i,j;

 cin>>n>>e;

 for(i=1;i<=e;i++)
   cin>>x[i]>>y[i]>>w[i];

 sort(1,e);/*将边按权值大小排序*/

 kruskal();

 return 0;
 }
```

度限制生成樹*

这个代码不是咱写的，不知道对否。

```
#include<stdio.h>
#include<string.h>
#define NN 30
#define INF 0x3fffffff

int idx, S;// S 为需要限制度的那一点

int k, mst;// k 表示 k 度限制，mst 为最后的结果
int pre[NN];
int mark[NN];
int dis[NN];
int vis[NN];
char str[NN][NN];
int map[NN][NN];

int best[NN]; // 存的是最大权值边的终点

int edg[NN][NN];// edg[i][j] = 1 表示边[i,j]已在生成树中

int father[NN];// 生成树中的父节点

int find(char s[]){
    int i;
    for (i = 0; i < idx; i++){
       if (strcmp(str[i], s) == 0) return i;
    }
     return -1;
```

```
}

void dfs(int cur){// 将树拉成有根树
    int i;
    for (i = 0; i < idx; i++){
        if (edg[i][cur] && mark[i]){
            father[i] = cur;
            mark[i] = 0;
            dfs(i);
        }
    }
}

 int prim(int s){// 求最小生成树
    int i, key, Min;
    int sum = 0;
    memset(pre, 0, sizeof(pre));
    for (i = 0; i < idx; i++){
        dis[i] = map[s][i];
        pre[i] = s;
    }
    memset(mark, 0, sizeof(mark));
    mark[s] = 1;
    vis[s] = 1;

    while(1){
      Min = INF;
      key = -1;
      for (i = 0; i < idx; i++){
        if (!vis[i] && !mark[i] && dis[i] < Min){
          key = i;
           Min = dis[i];
        }
      }
      if (key == -1) break;
      mark[key] = 1;
      vis[key] = 1;
      edg[pre[key]][key] = edg[key][pre[key]] = 1;
      sum += dis[key];
      for (i = 0; i < idx; i++){
        if (!vis[i] && !mark[i] && dis[i] > map[key][i]){
          dis[i] = map[key][i];
```

```
            pre[i] = key;
          }
        }
      }
    Min = INF;
    int root = -1; // 树根
    for (i = 0; i < idx; i++){
      if (mark[i] && map[i][S] < Min){
        Min = map[i][S];
        root = i;
      }
    }
    // 拉成有根树
    mark[root] = 0;
    dfs(root);
    father[root] = S;
    return sum + Min;
}

int Best(int x){// 求得 x 到 S 路径上的最大权值边
   int tmp;
    if (father[x] == S) return -1;
    if (best[x] != -1){
      return best[x];
    }
    tmp = Best(father[x]);
    if (tmp != -1 && map[tmp][father[tmp]] >
map[father[x]][x]){
      best[x] = tmp;
    }else best[x] = x;
    return best[x];
}

void Solve()
{
    int i, j;
    memset(vis, 0, sizeof(vis));
    vis[S] = 1;
    int m = 0;// 分支个数
    mst = 0;// 最小生成树和
    memset(father, -1, sizeof(father));
    memset(edg, 0, sizeof(edg));
    for (i = 0; i < idx; i++){// 先求得 m 限制生成树
```

```
      if (!vis[i]){
        m++;
        mst += prim(i);
      }
    }
/*    for (i = 0; i < idx; i++){
      printf("%d----%d %d\n", father[i], i, map[i][father[i]]);
    }可以用于调试错误
*/
    int minadd, ax, bx,tmp;
    int change; // 回路上权值最大的边，用于交换
    for (i = m + 1; i <= k && i < idx; i++){
      // 再由 m 度生成树得到 m+1 度生成树，最后求得 k 限制生成树
      memset(best, -1, sizeof(best));
      for (j = 0; j < idx; j++){
        if (best[j] == -1 && father[j] != S){
          Best(j);
        }
      }
      minadd = INF; // 交换边的最小差值
      for (j = 0; j < idx; j++){// 遍历所有临边
        if (map[S][j] != INF && father[j] != S){
          ax = best[j];
          bx = father[ax];
          tmp = map[S][j] - map[ax][bx];
          if (tmp < minadd){
            minadd = tmp;
            change = j;
          }
        }
      }
      if (minadd >= 0) break;//用于度数不大于 k 的限制，如果 k 限制，就不用 break 了
      mst += minadd;
      ax = best[change];
      bx = father[ax];
      map[ax][bx] = map[bx][ax] = INF;
      father[ax] = bx = S;// 改变生成树，将点 ax 直接指向源点 S
      map[ax][bx] = map[bx][ax] = map[change][S];
      map[S][change] = map[change][S] = INF;
    }
```

```
}
int main()
{
   int i, j, n, x, y, d;
   char s1[NN], s2[NN];
   scanf("%d", &n);
   for (i = 0; i <= NN - 2; i++){
      for (j = 0; j <= NN - 2; j++){
         map[i][j] = INF;
      }
   }
   idx = 1;
   strcpy(str[0], "Park");
   while(n--){
     scanf("%s%s%d", s1, s2, &d);
     x = find(s1);
     if (x == -1){
       strcpy(str[idx++], s1);
       x = idx - 1;
     }
     y = find(s2);
     if (y == -1){
       strcpy(str[idx++], s2);
       y = idx - 1;
     }
     if (d < map[x][y]){
       map[x][y] = d;
       map[y][x] = d;
     }
   }
   scanf("%d", &k);
   S = 0;
   Solve();
   printf("Total miles driven: %d\n", mst);
   return 0;
}
```

次小生成树 prim

算法分析

比较常见的是用 kruskal 算法计算最小生成树。然后再分别去掉最小生成树上的每一条边，再加入一条除了去掉了的那一条边之外的使得分开的两部分 相

连的最小边。分别去掉这 n-1 条边之后取最小值即可。判断两点是否属于同一部分重新算一遍 kruskal 的并查集即可。这种方法对于稀疏图来说很有效，时间复杂度大概 O(M * (N – 1) + N – 1 * logN + M*logM)去边的时间复杂度＋排序的时间复杂度＋合并的时间复杂度)

不过这道题我为了试 prim 的方法用了计算一个环中的第二大的边的方法，结果费了一个下午的时间。我没有看到有怎么介绍具体实现方法的所以这里写一下。

首先设 w[i , j]是从 i 到 j 的一条边，mat[i , j]是生成树上从 i 到 j 路径的最大边,pre[i]是点 i 在加入生成树之前生成树内连接的边权最短的点。

因为结点加入的先后次序不同，所以我们最后会得到一棵有根树。

Prim 算法在计算的时候每次会往生成树中增加一个点，我们记录这个点为 k,对于 k 的所有祖先 j，如果有 mat[j，k] < w[pre[k]，k]则将 mat[j，k]更新为 w[pre[k]，k]。否则则将其变成 mat[j，pre[k]]。这个过程类似于一个简单的区间类型的动态规划就是说我们对于 j 到 k 这段路径，选取其[j，pre[k]]这一段的最大值或者是新加入这段路径的 w[pre[k]，k]这条边。由于这个递推过程在 prim 过程中，所以递推的顺序和节点加入的顺序是一致的。最终我们会得到 1 到任意结点 k 以及 k 的任意祖先到 k 的路径上的最 大边。 比如说我们完成这个步骤之后会得到这样一棵生成树: 1->2->3->4

|

V

5->6->7

我们知道了[1,7]和[1,4]这两个区间内所有点之间的最大边，但是对于[4,7]这样祖先不同的路径这里还没有做处理，所以完成这个步骤之后，还需要对前驱(祖先)结点不同(互不包含)的节点进行处理，假如 i,j 是两个祖先不同的节点，那么就有：

mat[i , j] = max(mat[LCA(i,j) , i] , mat[LCA(i,j) , j]);LCA(i,j)是 i,j 的最低公共祖先，这个式子实际上就很显然了。

最后对于每个不在生成树中的边进行添加和删除操作，计算其最小值就可以了，为了方便起见在代码当中我将树中加入的边在每次循环结束后都赋为 0 了。设 MST 的权值和为 cnt,那么次小生成树的权值和则是 cnt + min(w[i , j] – mat[i , j]),其中 w[i , j]是不在 MST 中的边。

实际运行过程中没觉得有多快，可能稀疏图和稠密图的数据相抵消了吧。不知道把计算 LCA 的过程转化为 RMQ 是不是要快一些。

解题过程

数据会有权值为 0 的情况，+1 即可解决，最后结果减 n-1 即可，我觉得这是这题最有意思的地方了。

另外下面的代码可以把邻接矩阵换成邻接表可能会快一些。

```
#include<iostream>
#include<cstring>
#define INF 2141483647
#define MAXN 502
using namespace std;

int n , m ;
int cnt , scnt ;
```

```
int w[ MAXN ][ MAXN ] , mark[ MAXN ];
int mat[ MAXN ][ MAXN ];
int pre[ MAXN ];

void prim()
{
 int i , j , k , v ;
 int  mint ;
 cnt = 0 ;
 for( i = 1 ; i <= n ; ++ i )
   { mark[ i ] = w[ 1 ][ i ] > 0 ? w[ 1 ][ i ] : INF ; pre[ i ] =
1 ;}

 mark[ 1 ] = -1 ; pre[ 1 ] = 0;

 for( i = 1 ; i < n ; ++ i )
   {
    mint = INF; k = 0 ;
    for( j = 1 ; j <= n ; ++ j )
      if( mark[ j ] > 0  and mint > mark[ j ])
        { k = j , mint = mark[ j ] ; }

    cnt += mint;
    mark[ k ] = -1;

    //max range
    j = k ;
    while( pre[ j ] )
      {
        mat[ pre[ j ] ][ k ] = mat[ pre[ j ] ][ k ] > w[ pre[ k ] ]
[ k ] ? mat[ pre[ j ]][ k ] : w[ pre[ k ] ][ k ];
        if( mat[ pre[ j ] ][ k ] < mat[ pre[ j ] ][ pre[ k ] ])
          mat[ pre[ j ] ][ k ] = mat[ pre[ j ] ][ pre[ k ] ];
        mat[ k ][ pre[ j ] ] = mat[ pre[ j ] ][ k ];

        j = pre[ j ];
      }

    w[ pre[ k ] ][ k ] = w[ k ][ pre[ k ] ] = 0 ;


    for( j = 1 ; j <= n ; ++ j )
      if(mark[ j ] not_eq -1 and w[ k ][ j ] > 0  and mark[ j ]
> w[ k ][ j ] )
        mark[ j ] = w[ k ][ j ] , pre[ j ] = k;
   }

 for( i = 2 ; i <= n ; ++ i )
   for( j = i + 1  ; j <= n ; ++ j )
     {
      if( mat[ i ][ j ] == 0 )
        {
          for( k = i  ; k > 0 ; k = pre[ k ] )
            for( v = j  ; v > 0 ; v = pre[ v ] )
              if( k == v )
                {
                  mat[ i ][ j ] = mat[ i ][ k ] > mat[ j ][ v ] ? mat[
i ][ k ] : mat[ j ][ v ];
                  goto end;
                }
        }
    end:;
    }
```

```
 scnt = INF ;
 for( i = 1 ; i <= n ; ++ i )
  for( j = i + 1 ; j <= n ; ++ j )
   if( w[ i ][ j ]  and scnt > cnt - mat[ i ][ j ] + w[ i ][ j ] )
    scnt = cnt - mat[ i ][ j ] + w[ i ][ j ];

 return ;
}

int main()
{
 int u , v ;
 int  wi ;
 cin>>n>>m;
    for(int i = 1 ; i <= m ; ++ i )
      {
       cin>>u>>v>>wi;
       w[ v ][ u ] = w[ u ][ v ] = wi + 1 ;
      }

    prim();
    if( n - 1 == m ) scnt = -1;
    else scnt = scnt - n + 1 ;
    cout<<"Cost: "<<cnt - n + 1<<"\nCost:
"<<scnt<<"\n";
 return 0 ;
}
```

Euler Route/Tour

實際在小數據中遞歸和迭代差距幾乎可以忽略掉。

recursion:

```
#include<iostream>
#define MAXN 502
using namespace std;

int w[ MAXN ][ MAXN ];
int ans[ MAXN * 3 ] , ind[ MAXN ];
int n , m , st;

inline int max( int a , int b )
{ return a > b ? a : b; }

void init()
{
 int s , e;

 m = 0;
 for( int i = 1 ; i <= n ; ++ i )
  {
   cin>>s>>e;
   m = m > max( s , e ) ? m : max( s , e );
   ++ ind[ s ];
   ++ ind[ e ];
   ++ w[ s ][ e ];
```

```
    ++ w[ e ][ s ];
   }

 st = 0;
 for( int i = 1 ; i <= m ; ++ i )
  if( ind[ i ] & 1 )
    {
     st = i;
     break;
    }
  else if( ind[ i ] and not st )
    st = i;

 return ;
}

void dfs( int pos )
{
 for( int i = 1 ; i <= m ; ++ i )
  if( w[ pos ][ i ] )
    {
     -- w[ pos ][ i ];
     -- w[ i ][ pos ];
     dfs( i );
    }
 ans[ ++ ans[ 0 ] ] = pos;

 return ;
}

void out()
{
 for( int i = ans[ 0 ] ; i > 0 ; -- i )
  cout<<ans[ i ]<<"\n";
 return ;
}

void clr()
{
 for( int i = 1 ; i <= m ; ++ i )
  for( int j = 1 ; j <= m ; ++ j )
    w[ i ][ j ] = 0;
 for( int i = 1 ; i <= m ; ++ i )
  ind[ i ] = 0;
 ans[ 0 ] = 0;
 return ;
}

int main()
{
 while( cin>>n )
  {
```

```
    init();

    dfs( st );

    out();

    clr();
   }
 return 0;
}
```

iteration:

```
/*
 Euler Route calculate ( iteration )
 gestapolur
 2012-03-12
 2012-04-26
 description : find a euler-route/tour in a graph which exist a euler route/tour.
 hint: Use DFS as the recursion strategy. In case that a node may viste over once, it used a two dimension array pre[] to store last visted nodes.
*/
#include<cstdlib>
#include<cstdio>
#include<cstring>
#include<stack>
#define MAXN 502
using std::stack;

short int n , m , start;
short int ans[ MAXN * 2 ];
short int ind[ MAXN ] , st[ MAXN ];
short int w[ MAXN ][ MAXN ];
stack <short int> pre[ MAXN ];

inline short int max( short int a , short int b ){ return a > b ? a : b ; }

void init()
{
 short int i , s , e;

 m = 0;
 for( i = 1 ; i <= n ; ++ i )
  {
   scanf( "%hd%hd" , &s , &e );
   m = m > max( s , e ) ? m : max( s , e );
   ++ ind[ s ];
   ++ ind[ e ];
   ++ w[ s ][ e ];
   ++ w[ e ][ s ];
  }

 start = 0;
```

```
 for( i = 1 ; i <= m ; ++ i )
  if( ind[ i ] & 1 )
    {
     start = i;
     break;
    }
  else if( ind[ i ] and not start )
    start = i;

 for( i = 1 ; i <= m ; ++ i )
  st[ i ] = 1;
 return ;
}


void dfs()
{
 short int i , pos;
 bool turn;

 pos = start;
 do{
  turn = false;
  for( i = st[ pos ] ; i <= m ; ++ i )
   if( w[ pos ][ i ] )
     {
      -- w[ pos ][ i ];
      -- w[ i ][ pos ];
      st[ pos ] = i + 1;
      pre[ i ].push( pos );
      pos = i ;
      st[ i ] = 1;
      turn = true;
      break;
     }

 if( not turn )
  {
   ans[ ++ ans[ 0 ] ] = pos;
   st[ pos ] = 1;
   if( pre[ pos ].size() > 0 )
     {
      i = pre[ pos ].top();
      pre[ pos ].pop();
     }
   else
    i = 0;
   pos = i;
  }
}while( pos );
```

```
 for( i = ans[ 0 ] ; i > 0 ; -- i )
  printf( "%hd\n" , ans[ i ] );

 return ;
}

void clr()
{
 short int i , j;
 for( i = 1 ; i <= m ; ++ i )
  for( j = 1 ; j <= m ; ++ j )
   w[ i ][ j ] = 0;


 ans[ 0 ] = 0;
 for( i = 1 ; i <= m ; ++ i )
  {
   while( pre[ i ].size() )
    pre[ i ].pop();
   //st[ i ] = 1;
   //pre[ i ][ 0 ] = 0;
   ind[ i ] = 0;
  }
 return ;
}

int main()
{
```

```
 while( scanf( "%hd" , &n ) not_eq EOF )
  {
   init();

   dfs();

   clr();
  }

 return 0;
}
```

并查集 递归/迭代 *

迭代方法，设置一个临时变量，将找节点 a 的最终父亲节点和回溯更改路径上节点的父节点分成两步进行。

```
int gf(int a)
{
 if(f[ a ] == a )
  return a;
 int t,b;
 t = a;
 while(t not_eq f[ t ] )
  t = f[ t ];
```

```
 while(a not_eq t )
  {
   b = a;
   a = f[ a ];
   f[ b ] = t;
  }
 return t;
}
```

递归方法，一个经典例题，mafia

```
#include<iostream>
#define MAXN 100001
using namespace std;

int father[MAXN],n,m,q;

/*获取当前节点的父亲集合并更新递归路径上节点的父亲集合*/

int getfather(int v)
{
 if (father[v]==v)
   return v;
 father[v] = getfather(father[v]);
 return father[v];
}
```

```
/*判断两个节点是否属于同一集合*/
bool same(int x,int y)
{
 if( getfather(x) == getfather(y) )
  return true;
 return false;
}

/*合并两个不相交的集合,注意 cpp 中 union 是关键字不能定义为函数名*/
void Union(int x,int y)
{
  int fx,fy;
  fx = getfather(x);
  fy = getfather(y);
  if ( fx == fy ) return ;
  father[fx]=fy;
}

void init_work()
{
 cin>>n;//n 代表子树数
 cin>>m;//m 代表关系数
 cin>>q;//q 代表询问数
 int i ;
 for ( i=1; i<=n; i++)
```

```
  father[i]=i;
 for ( i=1; i<=m; i++)
  {
   int a,b;cin>>a>>b;
   Union(a,b);
  }
 int a,b;
 for ( i=1;i<=q;i++)
  {
   cin>>a>>b;
   cout<<(same(a,b)?"friend":"enime")<<endl;
  }
 return ;
}

int main()
{
 init_work();
 return 0;
}
```

强连通分量算法

Tarjan

利用 DFS 生成的搜索树和记录最早能够回溯到的搜索树上的节点 low 的方法来寻找强连通分量。

这个之前没写总结现在看得有点忘，回来再看一下。下面这个代码不保证正确。

con 里面最后存贮的是每个节点属于哪个强连通分量。

```
//tarjan test
#include<iostream>
#define MAXN 1001
#define INF 32767
using namespace std;

int lnk[ MAXN ][ MAXN ];
int mark[ MAXN ] , st[ MAXN ] , con[ MAXN ] ,
low[ MAXN ];
int n , m , idx , scc ;

bool in[ MAXN ];

void tarjan( int u )
{
 int i , v ;
 mark[ u ] = low[ u ] = ++ idx;
 st[ ++st[ 0 ] ] = u;
 in[ u ] = true;


 for( i = 1 ; i <= lnk[ u ][ 0 ] ; ++ i )
  if( mark[ lnk[ u ][ i ] ] == 0 )
   {
    tarjan( lnk[ u ][ i ] );
    low[ u ] = low[ u ] < low[ lnk[ u ][ i ] ] ? low[ u ] :
low[ lnk[ u ][ i ] ];
```

```
    }
   else if( in[ lnk[ u ][ i ] ] )
    low[ u ] = low[ u ] < mark[ lnk[ u ][ i ] ] ? low[ u ] :
mark[ lnk[ u ][ i ] ];

 if( low[ u ] == mark[ u ])
  {
   ++scc;
   do{
    v = st[ st[ 0 ] -- ];
    con[ v ] = scc;
    in[ v ] = false;
   }while( u not_eq v );
  }
 return ;
}


int main()
{
 int i , u , v ;
 cin>>n>>m;

 for( i = 1 ; i <= m ; ++ i )
  {
   cin>>u>>v;
   lnk[ u ][ ++ lnk[ u ][ 0 ] ] = v;
  }
 idx = 0 ;

 for( i = 1 ; i <= n ; ++ i )
  if( mark[ i ] == 0 )
   {
    st[ 0 ] = 0 ;
    tarjan( i );
   }

 for( i = 1 ; i <= n ; ++ i )
  cout<<low[ i ]<<" ";
 cout<<"\n";

 return 0 ;
}
```

計算 bridge: 桥 : dfs[i] < low[i] 割点 dfs[i] <= low[i]

Network flow(網絡流問題)

最大流问题

SAP Algorithm

```
/*
 SAP 非递归代码
 过程解释
 gestapolur
 2010.10.26 v1.0
 2010.12.11 v1.1
 */
#include<iostream>
#define MAXN 1006
#define INF 2141483647
using std::cin;
using std::cout;
/*
ifstream cin("ditch.in");
ofstream cout("ditch.out");
*/
bool flag;/*标记是否有路径被增广*/
int h[MAXN];/*当前 DFS 流到该点的流量*/
int pre[MAXN];/*某节点在 DFS 过程中标记的前驱节点*/
int d[MAXN];/*DFS 遍历的标号*/
int vh[MAXN];/*记录标号为 i 的节点的个数*/
int di[MAXN];/*某节点当前的流量*/
int w[MAXN][MAXN];/*网络当前的容量，用邻接矩阵表示*/
int mint,flow,aug,m,n;

/*
 aug 是当前增广路的流
 flow 为整个流网络的最大流
 m 图上的总边数
 n 图上的总节点数
*/
void dfs(int source,int sink)
{
 int i,j,tmp,rec=0;

 vh[0] = n;/* */

 for(i=1;i<=n;i++)/*di is the mark of DFS ,DFS 当前节点遍历到的节点的标记*/
  di[i] = 1;

 i = source;/*从源点开始*/
 aug=INF;

 while(d[source] < n) //DFS 过程，遍历结束
  {
   h[i] = aug;/*从当前节点标记的流开始*/
   flag = false;

   for(j = di[i] ; j <= n ; j++)
    if( w[i][j] > 0 and d[ j ] + 1 == d[ i ] )
      {/*DFS 标记必须是连续的，为什么是连续的?DFS 原理，关键！！！*/
        flag = true;
```

```
        di[ i ] = j;
        /*当前路径能够增广的流量取决于流量最小的那条边*/
        if( map[i][j] < aug )
          aug = map[i][j];

        pre[j] = i;
        i=j;

        if(i == sink)/*发现了一条增广路*/
          {
         flow += aug;/*增加总的流量*/
         while(i not_eq source)/*回溯，调整网络中的剩余流量*/
           {
             tmp = i;
             i = pre[i];
             map[i][tmp]-=aug;
             map[tmp][i]+=aug;
           }
         aug = INF;
          }
        break;
      }

    if(flag)  continue;
    /*以下是没有增广的情况，回溯到上一步增加节点的标号*/
    mint = n-1;
    /*关于下面的*/
    for(j=1;j<=n;j++)/*搜寻和当前顶点 i 相连的最短的没有流满的
```

```
顶点，标记在当前顶点的 di 上*/
     if(map[i][j] > 0 and d[j] < mint)/*标记最小的节点的原因是
因为前面增广的过程是 di[i] - n 的逐个扫描*/
        {rec = j; mint = d[j]; }/*以此保持 DFS 遍历的性质*/

    /*下面这几步的意思是在没有增广路可选择的情况下，给当前节点的
标号+1 然后回溯至上一顶点继续寻找*/

    di[i]=rec;/*标记顶点 i*/

    vh[d[i]]--;

    if(vh[d[i]] == 0)//没有通路了。
     break;

    d[i] = mint + 1;/*给节点 i 的标号+1*/

    vh[d[i]]++;//

    if(i not_eq source)/*回溯至前一个标记的顶点*/
     {i = pre[i];aug = h[i];}

   }

 return ;
}
```

```
int main()
{
 int i,x,y,w;
 cin>>n>>m;
 for(i=1;i<=m;i++)
  {
   cin>>x>>y>>w;
   map[x][y]+=w;
  }

 dfs(n,1);

 cout<<flow<<"\n";
 return 0;
}
```

forward star version

```
/*
 A SAP ALGORITHM IMPLEMENT
 write by gestapolur
 2012-08-13
 2012-08-23
 hint:SAP 算法的一个前向星实现
*/
#include<cstdio>
#include<cstring>
#define MAXN 502
#define MAXE MAXN * MAXN
#define INF 2141483647

/*
 流網絡的點數、邊數、源、匯，以及前向星鏈表表頭 head 和 next 指針,存儲從 i 出發的所有邊的序號。
 */
int n , m;
int tn , tm , vs , vt;
short u[ MAXE << 1 ] , v[ MAXE << 1 ];
int head[ MAXN ] , next[ MAXE << 1 ];

int res[ MAXE << 1 ];// cap[ MAXE << 1 ];
int h[ MAXN ];
int stk[ MAXN ];
short d[ MAXN ] , vh[ MAXN ] , di[ MAXN ];
int flow;
/*
 res - 邊上的剩餘流量
 h - 當前流到該點的流量
 stk - 記錄增廣過程中走過的邊的序號
 d - DFS 的標號
 vh - 標號爲 i 的節點數量
 di - 標記當前 DFS 過程搜索到的邊的序號
*/

void dfs( int source , int sink )
{
```

```
bool flag;//標記是否有路徑被增廣
int i , j , edg , cnt , tmp , rec = 0 , aug , mint;
vh[ 0 ] = n;
flow = 0;//初始化流量
aug = INF;
cnt = 0;
i = source;
memcpy( di , head, sizeof( di ) );
while( d[ source ] < n )
 {
  h[ i ] = aug;
  flag = false;
  for( edg = di[ i ] ; edg ; edg = next[ edg ] )
   {
    j = v[ edg ];
    if( res[ edg ] and d[ j ] + 1 == d[ i ] )
      {
       flag = true;
       di[ i ] = edg;
       aug = res[ edg ] < aug ? res[ edg ] : aug;
       stk[ ++ cnt ] = edg;
       i = j;
       if( i == sink )
       {
        flow += aug;
        while( cnt )
          {
           edg = stk[ cnt -- ];
           res[ edg ] -= aug;
           res[ ( edg - 1 ^ 1 ) + 1 ] += aug;
          }
        aug = INF;
        i = source;
       }
       break;
      }
   }
  if( flag ) continue;
  //在沒有通路的情況下，找有剩餘流量的標號最小的點，記錄到這個點的邊的標號，然後從這個
  mint = n - 1;
  for( edg = head[ i ] ; edg ; edg = next[ edg ] )
   if( res[ edg ] and d[ v[ edg ] ] < mint )
     { rec = edg ; mint = d[ v[ edg ] ]; }
  di[ i ] = rec;
  -- vh[ d[ i ] ];
  //如果調整過後某一個標號的數量爲 0，那麼就沒有增廣路了。
  if( not vh[ d[ i ] ] )
   break;
  d[ i ] = mint + 1;
  ++ vh[ d[ i ] ];
  if( i not_eq source )
   aug = h[ i = u[ stk[ cnt -- ] ] ];
 }
return ;
}
```

```
inline void add_edge( int i , short u1 , short v1 , int c )
{
 u[ i ] = u1; v[ i ] = v1;
 res[ i ] = c;
 next[ i ] = head[ u1 ];
 head[ u1 ] = i;
 return ;
}

void init()
{
 int i , u , v , c;
 scanf( "%d%d%d%d" , &n , &m , &vs , &vt );
 m *= 2;
 for( i = 1 ; i <= m ; i += 2 )
   {
    scanf( "%d%d%d" , &u , &v , &c );
    add_edge( i , u , v , c );
    add_edge( ( i - 1 ^ 1 ) + 1 , v , u , 0 );
   }
 return ;
}

int main()
{
 init();
 dfs( vs , vt );
 printf( "%d\n" , flow );
 for( int i = 1 ; i <= m ; i += 2 )
  printf( "%d %d (%d/%d)\n" , u[ i ] , v[ i ] , res[ i ] + res[ i + 1 ] , res[ i ] );
 return 0;
}
```

ford-fulkerson 与压入与重标记的对象不同；

网络流算法的实现方式很多样，BFS 和 DFS 貌似不同情况下有很大的区别；

h[u] == h[v] + 1 的解释：

h[i]为顶点的高度函数，在每次增量的过程中，h[i]最多增加 1，所以每次扫描恰好多出 1 的高度。is that so ?

SAP 和 Dinic 最大流都基于压入与重标记思想；

//push - relabel

DFS，BFS 每个顶点可能多于一次的遍历

//sap introduction

最短增广路算法

SAP 是用 DFS 的方法，直接增广一条从源到汇的路径，将这条路径流满。

首先来回顾一下用迭代的 DFS 方式遍历一个图的过程：

SAP 的 DFS 过程：

while 当原点 d[source]没有被抬高至 n 时(可能存在有增广路)

{

if 没有找到增广路

{

抬高标号最小且和

}

}

通过 SAP 的标记过程可以知道，标号 di 实际上是预先标记好的，只有在没有路径的情况下才需要提升某个节点的标号。

SAP 是基于上述 DFS 的过程进行的增广，因此节点需要标记之前节点 pre[i]和到此节点的流量 h[i]。

关于 gap 优化

为什么标号 d 必须要保持连续才能进行增广？

因为在抬高标号的过程中，标号每次只+1，

我们之前标记的 h[i],是以节点 i 为汇点的，之前所有顶点的最大可以流经的流量，i 之前的顶点都是由相差为 1 的标号的节点流过来的，

只有这样，才能够保持流守恒的特性。（增广流的时候）

简言之，就是到 i 的流已经是当前路径能够增广的最大流。这个标号的设置就是为了保持这一流守恒的特性。

Dinic Algorithm

Dinic 是基于 DFS 的分层图算法，算法每次增广过程只在当前构造出的分层图内进行。

forward star

```
/*
 a dinic implement test
 DITCH ORIGINAL DATA PASSED
 2012-08-07
 gestapolur
*/
#include<cstdio>
#include<cstring>
int const MAXV = 217;
int const MAXE = 217;

struct edge{//边
 short int u，v ;//e( u，v )
 int cap，flow，next; //残量网络的容量、流量、指向下一条从 u 出发的边的序号的指针(forward star 表示法)
```

```
};

short int n , m;
int cnt;
int head[ MAXV ];
edge ap[MAXE * 2];

short int h[ MAXV ], Q[ MAXV ];//Q 是用于构造分层图的一个队列
int G[ MAXV ];

//构造分层图
inline bool build(int s, int t){
 int p , x , y , front , rear;
 memset(h,255,sizeof(h));
 memset(G,0,sizeof(G));
 front = rear = h[ s ] = 0;
 Q[ rear ++ ] = s;
 G[ s ] = head[ s ];
 while( front < rear )
  {
   x = Q[ front ++ ];
   p = head[ x ];
   while( p )
    {
     y = ap[ p ].v;
     if( ap[ p ].cap and h[ y ] == -1 )
       {
         h[ y ] = h[ x ] + 1;
         G[ y ] = head[ y ];
         if( y == t ) return true;
         Q[ rear ++ ] = y;
       }
      p = ap[ p ].next;
    }
  }
 return false;
}

inline int min( int a , int b ){ return ( a < b ? a : b );}

//augumenting，dfs 增广过程 x 是当前节点，t 是汇点,low 是当前增广路上的流量。
int find(int x, int t, int low=0xffffffff)
{
 if(x==t)return low;
 int ret=0,y;
 //遍历所有与 x 相连的点 y，如果 y 在当前的分层图当中且仍然有剩余流量，则增广这条边并调整正向边和反向边的剩余流量
 for(int &p=G[x];p;p=ap[p].next)
  {
   y = ap[ p ].v;
   if(ap[p].cap && h[y] == h[x]+1 && ( ret = find( y , t , min( low , ap[p].cap ) ) ) )
    {
     ap[ p ].cap-=ret;
     ap[ (p - 1 ^ 1 ) + 1 ].cap+=ret;//p == odd ? p + 1 : p
```

```
- 1;
        return ret;
      }
    }
  return 0;
}

void maxflow( int s, int t )
{
 int flow;
 cnt = 0;
 while( build(s, t) )
   while ( flow = find( s , t ) )
     cnt+=flow;
 return ;
}

inline void add_edge( int i , int u , int v , int c )
{
 ap[ i ].u = u;
 ap[ i ].v = v;
 ap[ i ].cap = c;
 ap[ i ].flow = c;
 ap[ i ].next = head[ u ];
 head[ u ] = i;
 return ;
}
```

```
bool init()
{
 if( scanf( "%hd%hd" , &m , &n ) not_eq EOF )
   {
    int u , v , cap;
    memset( ap , 0 , sizeof( ap ) );
    memset( head , 0 , sizeof( head ) );
    for( int i = 0 ; i < m ; ++ i )
     {
       scanf( "%d%d%d" , &u , &v , &cap );
       add_edge( i * 2 + 1 , u , v , cap );
       add_edge( ( i + 1 ) * 2 , v , u , 0 );
     }
    return true;
   }
 return false;
}

void out()
{
 printf( "%d\n" , cnt );
 return ;
}

int main()
{
 while( init() )
```

```
  {
   maxflow( 1，n );
   out();
  }
 return 0;
}
```

最大流問題常見模型

拆點-將點權轉化爲邊權

計算最小權值和的割點集合

給出一個 N(N<=200)個點和 M(M<=20000)條邊的無向無權圖計算從 s 到 t 點權最小的割點集合的權值。(2012 ICPC Regional Chengdu Oline B)

對於原圖上的每個點拆成兩個點 i,i'，從 i 到 i'連一條流量爲該點點權的邊對於原圖的每條邊(u,v),在流網絡上加入(u+N,v)和(v+N,u)兩條邊，流量爲正無窮，原點和匯點分別爲 s 和 t',計算最大流。取最大流值和 s,t 點權值的最小者即爲答案。

拆點-將點轉化爲邊控制流量

有 F 種食物，D 種饮料,给出每種食物和飲料的數量，給出 N 個人對於每種食物和飲料是否能够接受的列表。一份食物和一份飲料可以给一個人配餐，每个人只能接受他们能接受的食物和飲料，问最多可以给多少人配餐。
N，F，D 不超过 200.

可以用最大流建模。從源點向每種食物連一條流量爲食物數量的邊，從每種飲料出發向匯點連一條流量爲飲料數量的邊，再將每個人轉化成一條邊和兩個點(將第 i 個人轉化成兩個點 i 和 i'，他們之間連一條流量爲 1 的邊)，然後從第 i 個人可以接受的食物到點 i 連一條流量爲 1 的邊，從 i'出發向他能夠接受的飲料連一條流量爲 1 的邊，計算最大流即可。

费用流問題(Cost Flow Algorithm)

消圈算法(Circle-Canceling Algorithm)

首先在一个流网络上，我们记录边(u，v)的费用为 w[u，v]，记录其反向边(v，u)的权值为-w[u，v](这是为了在调整路径时可以改变流量)。先随便走出来一个最大的可行流，然后在残量网络上去找所有的负的增广圈，找出所有这样的增广圈，然后对其增广一次，同时调整流量和权值。这种算法实现起来比较困难，而且如果真是这样消圈的话时间复杂度也会比较高，所以用的比较多的是后面良两种算法。
，

连续最短路径算法(Successive Shortest Path Algorithm)

这个算法实际上是消圈算法的优化，按照消圈算法构图之后每次选择一条费用最少的道路进行增广，每次增广的流量就是这条增广路上流量最小的边。这个算法的正确性可以保证，实际上这个算法是将消圈算法中所需要取消的圈用 Bellman-ford 或者 SPFA 在寻找增广路的过程中直接找出并增广了这些圈。

在没有负流量的增广圈(就是一个流量圈，走完以后费用会减少但是流量不会变，因为是一个圈，所以流量不会改变，所以可以走 n 次，总的费用就可以

是无穷小)

沿着存在剩余流量的边扩充一条源到汇的最短路。并对当前的残量网络进行增广。

实际上是用 SAP 的 DFS 的方法，但是如果增广路当中出现了负的费用圈的话最短路算法设置的前驱可能出现死循环。所以一般的費用流問題都可以保證建立的模型的正向邊是有向無環圖。

最小/大費用流的 SAP 實現(implement of cost flow)

計算在最大流的前提下費用最小 / 最大的方案。

最小費用流：

hint:輸入 n,m,vs,vt,分別表示點數、邊數和源、匯點。接下來 m 行 u,v,c,ct 分別表示邊的起始點和容量、費用。

```
/*
 Minimum cost maximum flow test
 2012-08-20
 2012-08-21
 Gestapolur
*/
#include<cstdio>
#include<cstring>
#define MAXN 502
#define MAXE MAXN * MAXN
#define INF 2141483647

int n , m;
int tn , tm , vs , vt;
short u[ MAXE << 1 ] , v[ MAXE << 1 ];
int head[ MAXN ] , next[ MAXE << 1 ];

int res[ MAXE << 1 ];
int h[ MAXN ];
int stk[ MAXN ];
short d[ MAXN ] , vh[ MAXN ] , di[ MAXN ];
int flow;

int cost[ MAXE << 1 ] , mark[ MAXN ] , pre[ MAXN ] ,
prem[ MAXN ] , mp[ MAXE << 1 ];
int mcost;

void dfs( int source , int sink )
{
 bool flag;
 int i , j , edg , cnt , tmp , rec = 0 , aug , mint;
 memset( h , 0 , sizeof( h ) );
 memset( d , 0 , sizeof( d ) );
 vh[ 0 ] = n;
 aug = INF;
 cnt = 0;
 i = source;
 memcpy( di , head , sizeof( di ) );
 while( d[ source ] < n )
  {
   h[ i ] = aug;
```

```
    flag = false;
    for( edg = di[ i ] ; edg ; edg = next[ edg ] )
     {
      j = v[ edg ];
      if( res[ edg ] and mp[ edg ] and d[ j ] + 1 == d[ i ] )
        {
         flag = true;
         di[ i ] = edg;
         aug = res[ edg ] < aug ? res[ edg ] : aug;
         stk[ ++ cnt ] = edg;
         i = j;
         if( i == sink )
         {
          flow += aug;
          while( cnt )
            {
             edg = stk[ cnt -- ];
             res[ edg ] -= aug;
             res[ ( edg - 1 ^ 1 ) + 1 ] += aug;
             mcost += cost[ edg ] * aug;
             //printf( "%d %d %d\n" , u[ edg ] , v[ edg ] ,
cost[ edg ] * aug );
            }
          aug = INF;
          i = source;
         }
         break;
        }
     }
    if( flag ) continue;
    mint = n - 1;
    for( edg = head[ i ] ; edg ; edg = next[ edg ] )
     if( res[ edg ] and mp[ edg ] and d[ v[ edg ] ] < mint )
       { rec = edg; mint = d[ v[ edg ] ];}
    di[ i ] = rec;
    -- vh[ d[ i ] ];
    if( not vh[ d[ i ] ] )
     break;
    d[ i ] = mint + 1;
    ++ vh[ d[ i ] ];
    if( i not_eq source )
     aug = h[ i = u[ stk[ cnt -- ] ] ];
   }
  return ;
}

bool bellman_ford()
{
 bool sign;
 int i , j , k;
 for( i = 1 ; i <= n ; ++ i )
  mark[ i ] = INF;
 mark[ vs ] = 0;
 for( i = 1 ; i <= n ; ++ i )
  {
```

```
    sign = false;
    for( j = 1 ; j <= n ; ++ j )
     if( mark[ j ] not_eq INF )
       for( k = head[ j ] ; k ; k = next[ k ] )
           if( mark[ v[ k ] ] > mark[ j ] + cost[ k ] and
res[ k ] )
           {
            mark[ v[ k ] ] = mark[ j ] + cost[ k ];
            pre[ v[ k ] ] = j;
            prem[ v[ k ] ] = k;
            sign = true;
           }
    if( not sign )
     break;
   }
 for( i = 1 ; i <= n ; ++ i )
  for( j = head[ i ] ; j ; j = next[ j ] )
   if ( mark[ v[ j ] ] > mark[ i ] + cost[ j ] and mark[ i ]
not_eq INF and res[ j ] )
      return false;
 return ( mark[ vt ] < INF ? true : false );
}

void out()
{
 printf("%d %d\n" , flow , mcost );
 return ;
}
```

```
inline void add_edge( int i , int u0 , int v0 , int c , int ct )
{
 u[ i ] = u0;
 v[ i ] = v0;
 res[ i ] = c;
 cost[ i ] = ct;
 next[ i ] = head[ u0 ];
 head[ u0 ] = i;
 return ;
}

bool init()
{
 if( scanf( "%d%d%d%d" , &n , &m , &vs , &vt ) not_eq
EOF )
  {
   int u , v , c , ct;
   flow = 0;
   mcost = 0;
   m <<= 1;
   for( int i = 1 ; i <= m ; i += 2)
    {
     scanf( "%d%d%d%d" , &u , &v , &c , &ct );
     add_edge( i , u , v , c , ct );
     add_edge( i + 1 , v , u , 0 , -ct );
    }
   return true;
```

```
  }
 return false;
}

inline void rec()
{
 int x = vt;
 memset( mp , false , sizeof( mp ) );
 while( x not_eq vs )
  {
   mp[ prem[ x ] ] = true;
   mp[ ( prem[ x ] - 1 ^ 1 ) + 1 ] = true;
   x = pre[ x ];
  }
 return ;
}

int main()
{
 while( init() )
  {
   while( bellman_ford() )
    {
     rec();
     dfs( vs , vt );
    }
   out();
  }
 return 0;
}
```

最大費用流：

hint:輸入同上，用 DFS 查找最長路徑

```
/*
 Maximum cost flow test
 TESTED
 2012-08-20
*/
#include<cstdio>
#include<cstring>
#define MAXN 502
#define MAXE MAXN * MAXN
#define INF 2141483647

int n , m;
int tn , tm , vs , vt;
short u[ MAXE << 1 ] , v[ MAXE << 1 ];
int head[ MAXN ] , next[ MAXE << 1 ];

int res[ MAXE << 1 ];
int h[ MAXN ];
int stk[ MAXN ];
short d[ MAXN ] , vh[ MAXN ] , di[ MAXN ];
int flow;
```

```
int cost[ MAXE << 1 ] , mp[ MAXE << 1 ];//minimum /
maximum path
int mcost;

void dfs( int source , int sink )
{
 bool flag;
 int i , j , edg , cnt , tmp , rec = 0 , aug , mint;
 memset( h , 0 , sizeof( h ) );
 memset( d , 0 , sizeof( d ) );
 vh[ 0 ] = n;
 aug = INF;
 cnt = 0;
 i = source;
 memcpy( di , head , sizeof( di ) );
 while( d[ source ] < n )
  {
   h[ i ] = aug;
   flag = false;
   for( edg = di[ i ] ; edg ; edg = next[ edg ] )
    {
     j = v[ edg ];
     if( res[ edg ] and d[ j ] + 1 == d[ i ] and mp[ edg ] )
       {
         flag = true;
         di[ i ] = edg;
         aug = res[ edg ] < aug ? res[ edg ] : aug;
         stk[ ++ cnt ] = edg;
         i = j;
         if( i == sink )
         {
          flow += aug;
          while( cnt )
            {
              edg = stk[ cnt -- ];
              res[ edg ] -= aug;
              res[ ( edg - 1 ^ 1 ) + 1 ] += aug;
              mcost += cost[ edg ] * aug;
            }
          aug = INF;
          i = source;
         }
         break;
       }
    }
 if( flag ) continue;


 mint = n - 1;
 for( edg = head[ i ] ; edg ; edg = next[ edg ] )
  if( res[ edg ] and mp[ edg ] and d[ v[ edg ] ] < mint )
    { rec = edg; mint = d[ v[ edg ] ];}
 di[ i ] = rec;
 -- vh[ d[ i ] ];
 if( not vh[ d[ i ] ] )
  break;
 d[ i ] = mint + 1;
 ++ vh[ d[ i ] ];
```

```
    if( i not_eq source )
     aug = h[ i = u[ stk[ cnt -- ] ] ];
   }
 return ;
}

bool sign[ MAXN ];
int mark[ MAXN ] , pre[ MAXN ] , prem[ MAXN ];
bool dfs( int x )
{
 for( int i = head[ x ] ; i ; i = next[ i ] )
  if( not sign[ v[ i ] ] and mark[ v[ i ] ] <= mark[ x ] + cost[
i ] and res[ i ] )
    {
     sign[ v[ i ] ] = true;
     mark[ v[ i ] ] = mark[ x ] + cost[ i ];
     pre[ v[ i ] ] = x;
     prem[ v[ i ] ] = i;
     dfs( v[ i ] );
     sign[ v[ i ] ] = false;
    }
 return ( pre[ vt ] ? true : false );
}

void out()
{
 printf("%d %d\n" , flow , mcost );
 return ;
}
```

```
void add_edge( int i , int u0 , int v0 , int c , int ct )
{
 u[ i ] = u0;
 v[ i ] = v0;
 res[ i ] = c;
 cost[ i ] = ct;
 next[ i ] = head[ u0 ];
 head[ u0 ] = i;
 return ;
}

bool init()
{
 if( scanf( "%d%d%d%d" , &n , &m , &vs , &vt ) not_eq
EOF )
  {
   int u , v , c , ct;
   flow = 0;
   mcost = 0;
   m <<= 1;
   for( int i = 1 ; i <= m ; i += 2)
    {
     scanf( "%d%d%d%d" , &u , &v , &c , &ct );
     add_edge( i , u , v , c , ct );
     add_edge( i + 1 , v , u , 0 , -ct );
    }
```

```
    memset( sign , false , sizeof( sign ) );
    sign[ vs ] = true;
    for( int i = 1 ; i <= n ; ++ i ) mark[ i ] = -INF;
    mark[ vs ] = 0;
    return true;
   }
 return false;
}

inline void rec()
{
 int x = vt;
 memset( mp , false , sizeof( mp ));
 while( x not_eq vs )
  {
   mp[ prem[ x ] ] = true;
   mp[ ( prem[ x ] - 1 ^ 1 ) + 1 ] = true;
   x = pre[ x ];
  }
 for( x = 1 ; x <= n ; ++ x  ) mark[ x ] = -INF;
 mark[ vs ] = 0 ;
 memset( pre , 0 , sizeof( pre ) );
 return ;
}

int main()
{
 while( init() )
  {
   while( dfs( vs ) )
    {
     rec();
     dfs( vs , vt );
    }
   out();
  }
 return 0;
}
```

EK 算法*

最小割問題(Min-cut Algorithm)

Stoer-Wagner[1]

計算無向帶權圖 G (V , E) 的最小割。

1 A Simple Min-Cut Algorithm,Journal of the ACM, Vol. 44, No. 4, July 1997, pp. 585-591.

算法大致過程：

1.初始化一個點集 A={a}，每次選取與 A 有邊相連的 V-A 中的最大邊，並將在 V-A 中的點加入 A 中直至所有的點都加入 A。設最後加入的點 s 和倒數第二加入的點 t 然後計算割 G/{s-t}。合併 s 與 t.

2.重複過程 1.直至 | V | =1

參考題目： 2010 FUZHOU REGIONAL ONSITE B(SPOJ FZ10B)

```
/*
 Stoer-Wagner Algorithm test
 ACCEPTED
 2012-09-20
 gestapolur
*/
#include<cstdio>
#include<cstring>
#define MAXN 313
#define INF 2141483647
int n , m , s , cnt;
int w[ MAXN ][ MAXN ] , mark[ MAXN ];
bool in[ MAXN ];

bool init()
{
 scanf("%d%d%d" , &n , &m , &s );
 if( n == 0 && m == 0 && s == 0 )
  return false;
 int i , j , u , v , c;
 for( i = 1 ; i <= n ; ++ i )
  for( j = 1 ; j <= n ; ++ j )
   w[ i ][ j ] = 0;
 for( i = 1 ; i <= m ; ++ i )
  {
   scanf("%d%d%d" , &u , &v , &c );
   if( u != v )
    w[ u ][ v ] += c , w[ v ][ u ] += c;
 }
 return true;
}

void mincut()
{
 int i , j , k , l1 , l2 , minc = INF , tminc , tmax , tot = n;
 for( i = 1 ; i <= n ; ++ i )
  {
   for( tminc = 0 , j = 1 ; j <= n ; ++ j )
    tminc += w[ i ][ j ];
   minc = tminc < minc ? tminc : minc;
  }
 if( n > 2 )//mincutphase
  for( l1 = 0 , l2 = 1 , i = 1 ; i < tot ; ++ i , -- n )
   {
    memset( in , false , sizeof( bool ) * ( n + 1 ));
```

```
    memcpy( mark , w[ 1 ] , sizeof( int ) * ( n + 1 ));
    for( in[ 1 ] = true , mark[ 1 ] = 0 , j = 1 ; j < n ; ++ j )
      {
       for( l1 = l2 , l2 = 0 , tmax = 0 , k = 1 ; k <= n ; ++
k )
         if( !in[ k ] && tmax < mark[ k ] )
         { tmax = mark[ k ]; l2 = k; }
       if( !l2 ) { printf( "0\n" ); return ; }//graph were not
connected
       in[ l2 ] = true;
       for( k = 1 ; k <= n ; ++ k )
         if( mark[ k ] < w[ l2 ][ k ] )
         mark[ k ] = w[ l2 ][ k ];
      }
    for( j = 1 ; j <= n ; ++ j )//merge
      {
       if( j != l1 ) { w[ l1 ][ j ] += w[ l2 ][ j ]; w[ j ][ l1 ] +=
w[ j ][ l2 ];}
       w[ l2 ][ j ] = w[ n ][ j ]; w[ j ][ l2 ] = w[ j ][ n ];
      }
    for( tminc = 0 , j = 1 ; j < n ; ++ j )//count the number
      tminc += w[ l1 ][ j ];
    minc = tminc < minc ? tminc : minc;
   }
 printf( "%d\n" , m < tot - 1 ? 0 : minc );
 return ;
}

int main()
{
 while( init() )
  mincut();
 return 0;
}
```

Bipartite Graph (二分图问题)

Hungary 算法

简言之就是用 DFS 的方法找可行的路径，如果不行的话试着调整已经走过的路径上的点来增加匹配。

```
#include<iostream>
#include<cstring>
#define MAXN 302
using namespace std;

int n , m;
int w[ MAXN ][ MAXN ];
int linky[ MAXN ];
bool visx[ MAXN ] , visy[ MAXN ];

void init()
{
 int u , v;
 cin>>n>>m;
 for( int i = 1 ; i <= m ; ++ i )
```

```
  {
   cin>>u>>v;
   w[ u ][ v ] = 1;
  }
 return ;
}

bool find( int x )
{
 for( int i = 1 ; i <= n ; ++ i )
  if( w[ x ][ i ] and not visy[ i ] )
   {
    visy[ i ] = true;
    if( not linky[ i ] or find( linky[ i ] ) )
      {
       linky[ i ] = x;
       return true;
      }
   }
 return false;
}

void hungry()
{
 int cnt = 0;
 for( int i = 1 ; i <= n ; ++ i )
  {
   memset( visy , false , sizeof( visy ) );
   if( find( i ) )
    ++ cnt;
  }
 cout<<cnt<<"\n";
 for( int i = 1 ; i <= n ; ++ i )
  cout<<linky[ i ]<<" ";
 cout<<"\n";
 return ;
}

int main()
{
 init();
 hungry();
 return 0;
}
```

KM

```
/*
这里求的是除了最优匹配之外的图的权值
*/
#include <cstdio>
#include <cstring>
using namespace std;
```

```
const int maxn=160,OO=2147483647;
int w[maxn][maxn];
int lx[maxn],ly[maxn];
int linky[maxn];
int visx[maxn],visy[maxn];
int N;
int slack[maxn];

void input(){
    scanf("%d",&N);
    for(int i=0;i<N;++i)
        for(int j=0;j<N;++j)
            scanf("%d",&w[i][j]);
}
bool find(int x){
    visx[x]=true;
    for(int y=0;y<N;++y){
        if(visy[y])continue;
        int t=lx[x]+ly[y]-w[x][y];
        if(t==0){
            visy[y]=true;
            if(linky[y]==-1||find(linky[y])){
                linky[y]=x;
                return true;
            }
        }
        else{
            if(slack[y]>t)
                slack[y]=t;
        }
    }
    return false;
}
void KM(){
    memset(linky,-1,sizeof(linky));
    memset(lx,0,sizeof(lx));
    memset(ly,0,sizeof(ly));
    for(int i=0;i<N;++i)
        for(int j=0;j<N;++j)
            if(w[i][j]>lx[i])
                lx[i]=w[i][j];
    for(int x=0;x<N;++x){
        for(int i=0;i<N;++i)
            slack[i]=OO;
        for(;;){
            memset(visx,0,sizeof(visx));
            memset(visy,0,sizeof(visy));
            if(find(x))break;
            int d=OO;
            for(int i=0;i<N;++i){
                if(!visy[i])
                    if(d>slack[i])
                        d=slack[i];
            }
            for(int i=0;i<N;++i){
                if(visx[i])
                    lx[i]-=d;
            }
            for(int i=0;i<N;++i){
                if(visy[i])
                    ly[i]+=d;
                else
                    slack[i]-=d;
            }
```

```
            }
        }
    }
    void output(){
        int res=0;
        /*
        for(int i = 0 ; i < N ; ++ i)
          {
            for(int j = 0 ; j < N ; ++ j)
              printf("%d ",w[ i ][ j ]);
            printf("\n%d %d\n" ,linky[ i ] , w[ linky[ i ] ][ i ]);
          }
        */
        for(int j=0;j<N;++j){
            for(int i=0;i<N;++i)
                res+=w[i][j];
            res-=w[linky[j]][j];
        }
        printf("%d\n",res);
    }
    int main(){
        input();
        KM();
        output();
    }
```

Minimum Path Cover(最小路徑覆蓋問題)

DAG ： 有向无环图

路徑覆蓋：
http://en.wikipedia.org/wiki/Path_cover#CITEREFDiestel20

05

在 DAG 中，圖 G 的最小路径覆盖 = G 顶点数 V - 轉化成二分圖 G‘後的最大匹配数

证明：

1.首先这样建立一个二分图：（算法导论“最小路径覆盖”词条对应的练习）

2.假设在这个二分图中，已经有一个匹配。那么对于点 x'，如果它和某个 y'匹配，那么在原图 G 上 x'一定不是某个**覆盖路径**的结束的点。所以这个在匹配中的点 x'就都不是覆盖路径结束的点，显然每条匹配路径中只有一个结束的点，而且因为所有
的点都在覆盖路径中，那么 V 减去在匹配中的 x'的数量就是覆盖路径的数量了。

3.所以 x'的數量越多，覆蓋路徑的數量就會越少。根據 König 定理，最大覆蓋頂點數量等於最大匹配數，所以對這個二分圖計算最大匹配就可以算出最少覆蓋路徑了。

證明到此結束了。

上面實際上是 Dilworth 定理轉換成等價的二分圖表示後的證明（下面的鏈接）

這裏有一個歸納法的證明：http://en.wikipedia.org/wiki/Dilworth%27s_theorem

4.如何计算每一條覆盖路径

利用之前的匹配可以計算出一個拓撲序從而就知道每條路徑了。

ps:所以这个定理一定要在有向无环图上才能够成立也是顯然的了。

Balanced Tree(平衡树)

Treap

可能左旋和右旋时反的，使用时可能需要用注释掉的那段代码。

```
/*
 treap code sample
 by gestapolur
2009.4.14
 */
#include<iostream>
#define MAXN 1001
using std::cin;
using std::cout;

int n,root;

class Node
{
public:
 int key,rank,lc,rc,pa;
};

Node node[MAXN];

int find(int k)
{
 int x=root;
 while( x and node[x].key not_eq k )
  {
   if( k < node[x].key)
    x=node[x].lc;
   else
    x=node[x].rc;
  }
 return x;
}

int getmin(int x)
{
 while(node[x].lc)
  x=node[x].lc;
 return x;
}

int getmax(int x)
{
 while(node[x].rc)
  x=node[x].rc;
}

int successor(int x)
{
 if(node[x].rc)
  return getmin(node[x].lc);
```

```
 int y=node[x].pa;
 while( y and x==node[y].rc)
  {
   x=y;
   y=node[y].pa;
  }
 return y;
}
int pred(int x)
{
 if(node[x].lc)
  return getmax(node[x].lc);
 int y=node[x].pa;
 while( y and x==node[y].lc)
  {
   x=y;
   y=node[y].pa;
  }
 return y;
}
/*
void leftratote(int x)
{
 int y = node[x].pa;
 if(y==root) root=x;
 node[x].pa=node[y].pa;
 node[node[x].rc].pa=y;
 node[y].lc=node[x].rc;
 node[x].rc=y;
 if(node[node[y].pa].lc==y)
  node[node[y].pa].lc=x;
 else
  node[node[y].pa].rc=x;
 node[y].pa=x;
 return;
}

void rightratote(int x)
{
 int y=node[x].pa;
 if(y==root) root = x;
 node[x].pa=node[y].pa;
 node[node[x].lc].pa=y;
 node[y].rc=node[x].lc;
 node[x].lc=y;
 if(node[node[y].pa].lc==y)
  node[node[y].pa].lc=x;
 else
  node[node[y].pa].rc=x;
 node[y].pa=x;
 return;
}
```

```
*/

void rightratote(int x)
{
 int y;
 y=node[x].pa;
 node[y].lc=node[x].rc;
 if(y) root = x;
 node[x].pa=node[y].pa;
 if(node[node[y].pa].lc == y)
  node[y].lc = x;
 if(node[node[y].pa].rc == y)
  node[y].rc = x;
 node[x].rc = y;
 node[y].pa = x;
 return ;
}

void leftratote(int x)
{
 int y;
 y=node[x].rc;
 node[x].rc=node[y].lc;
 if(node[y].lc)
  node[node[y].lc].pa=x;
 node[y].pa=node[x].pa;
 if(node[x].pa)//x is the root
  root = y;
 if(node[node[x].pa].lc == x)//where is x original stayed
  node[node[x].pa].lc=y;//and change the x and y's position
 else
  node[node[x].pa].rc=y;
 node[y].lc=x;
 node[x].pa=y;
 return ;
}


void insert(int val)
{
 node[++n].key=val;
 node[n].lc=node[n].rc=node[n].pa=0;
 node[n].rank=rand();
 int x = root,y=0;
 while(x)
 {
  y=x;
  if(node[n].key < node[x].key)
   x=node[x].lc;
  else
```

```
   x=node[x].rc;
 }

 if(y)
  {
   node[n].pa=y;
   if(node[n].key < node[y].key)
    node[y].lc=n;
   else
    node[y].rc=n;
  }
 else
  root=n;
 //keep the heap
 while(node[n].pa and node[node[n].pa].rank >
node[n].rank )
  if( node[node[n].pa].lc==n)
   leftratote(n);
  else
   rightratote(n);
 return;
}

void del(int pos)
{
 while(node[pos].lc or node[pos].rc)
  if(node[node[pos].lc].rank < node[node[pos].rc].rank)
   leftratote(node[pos].lc);
  else
   rightratote(node[pos].rc);
 if(node[node[pos].pa].lc==pos)
  node[node[pos].pa].lc=0;
 else
  node[node[pos].pa].rc=0;
 node[pos].key=0;
 node[pos].rank=0;
 node[pos].pa=0;
 return;
}

void dis(int pos)
{
 if(node[pos].lc)
  dis(node[pos].lc);
 cout<<"num:"<<pos<<" key: "<<node[pos].key<<"
rank: "<<node[pos].rank<<" lc:"<<node[pos].lc<<"
rc:"<<node[pos].rc<<" p: "<<node[pos].pa<<"\n";
 if(node[pos].rc)
  dis(node[pos].rc);
 return ;
}
```

```
int main()
{
 int ins,k;
 while(1)
  {
   cin>>ins;
   if(ins == 1){cin>>ins;insert(ins);}
   else if(ins == 2){cin>>ins;del(ins);}
   else  break;
   dis(root);
  }
 return 0;
}
```

String Problems(串相关问题)

Suffix Array(后缀数组)

2

简要说明

后缀数组 SA[i]：排第 i 的后缀的起始位置

rank[i] - 从 i 开始的后缀的排名

字符串最后添加一个比其他字符都小的字符用来方便比较大小

如果知道了 rank 或者 SA 之一就可以用 O(1)的时间复杂度算出另外一个

如何排列：字符为第一关键字，长度（升序，长度短的优先）为第二关键字。

LCP 定理：suffix(sa[i])...suffix(sa[j]) (i < j) 的最长公共前缀等于 min{LCP(sa[k] , sa[k + 1]|i<=k<j}

这个定理的证明：

设 i<j，则 LCP(i,j)=min{LCP(k-1,k)|i+1≤k≤j}（LCP Theorem）

要证明LCP Theorem，首先证明LCP Lemma:

对任意1≤i<j<k≤n，LCP(i,k)=min{LCP(i,j),LCP(j,k)}

证明：设p=min{LCP(i,j),LCP(j,k)}，则有LCP(i,j)≥p,LCP(j,k)≥p。

设 Suffix(SA[i])=u,Suffix(SA[j])=v,Suffix(SA[k])=w。

由$u=_{LCP(i,j)}v$得$u=_p v$；同理$v=_p w$。

于是$Suffix(SA[i])=_p Suffix(SA[k])$，即LCP(i,k)≥p。 (1)

又设LCP(i,k)=q>p，则

u[1]=w[1],u[2]=w[2],...u[q]=w[q]。

而min{LCP(i,j),LCP(j,k)}=p 说明u[p+1]≠v[p+1]或v[p+1]≠w[q+1]，

设u[p+1]=x,v[p+1]=y,w[p+1]=z，显然有x≤y≤z，又由p<q 得p+1≤q，

应该有x=z，也就是x=y=z，这与u[p+1]≠v[p+1]或v[p+1]≠w[q+1]矛盾。

于是，q>p 不成立，即LCP(i,k)≤p。 (2)

2 这里题目列表和一些题目的解答参考了 NOI2009 国家集训队罗穗骞同学的论文《后缀数组——处理字符串的有力工具》及其附件里的代码

综合(1),(2)知 LCP(i,k)=p=min{LCP(i,j),LCP(j,k)}，LCP Lemma 得证。

于是LCP Theorem可以证明如下：

当j-i=1 和j-i=2 时，显然成立。

设j-i=m时LCP Theorem成立，当j-i=m+1 时，

由LCP Lemma 知LCP(i,j)=min{LCP(i,i+1),LCP(i+1,j)}，

因j-(i+1)≤m，LCP(i+1,j)=min{LCP(k-1,k)|i+2≤k≤j}，故当j-i=m+1 时，仍有

LCP(i,j)=min{LCP(i,i+1),min{LCP(k-1,k)|i+2≤k≤j}}=min{LCP(k-1,k}|i+1≤k≤j)

根据数学归纳法，LCP Theorem成立[3]

Doubling Augumenting 2 倍增算法

DA 的核心就是通过倍增子串并比较各子串的当前长度(关键字 x)和之前的 rank 值(关键字 y).

由于所有子串的长度都不相同，所以排序结束后必然所有子串的后缀数组值都不同，所以算法扩展至所有 rank 值都不同(最大 rank 值等于 n)后结束。

二倍增过程:

```
//x - 当前位置起始的后缀的排名
//y - 当前位置加上 2^k 以后位置的后缀的排名
S[ n + 1 ] = $//$是比{S}中的所有的字符都要小的值
x -> sort( {S} )//x 初始值为对{S}中单个字符排序之后的结果。
y -> 0
k -> 0
while 2^k < N do
    rank -> sort( x , y )//以第一关键字 x 和第二关键字 y 排序
    ++k
    x -> rank
    for i := 1 to n
        y[ i ] -> (i + 2 ^ k) <= n ? sa[ x[ 2 ^ k ] ]: $
    for i := 1 to n
        sa[ rank[ i ] ] = i;
```

算法的可行性:

当 2^k>=n 时必然所有[i , i + 2 ^ k](1<=i<=n)的字符串都会不相同，所以算法一定会结束并且所有的字符串的 rank 值都必然不同。

算法的正确性:(简要的证明)

x[i],y[i]代表从一个位置 i 开始[i , 2^k-1]和[2^k-1 , 2^k]的两段字符串在上一次的 rank

用 x[i],y[i]作为两个关键字排序，实际上是利用了上一次对这两段字符串排序结果的信息

时间复杂度

和字符串的最长公共前缀相关,最坏 O(N*logN)

```
#include<cstdlib>
#include<cstdio>
```

3 2004 年中国国家集训队，许智磊

```
#define maxn 1000001

int wa[maxn],wb[maxn],wv[maxn],ws[maxn];

int cmp(int *r,int a,int b,int l)
{return r[a]==r[b]&&r[a+l]==r[b+l];}
void da(int *r,int *sa,int n,int m)
{
    int i,j,p,*x=wa,*y=wb,*t;
    for(i=0;i<m;i++) ws[i]=0;
    for(i=0;i<n;i++) ws[x[i]=r[i]]++;
    for(i=1;i<m;i++) ws[i]+=ws[i-1];
    for(i=n-1;i>=0;i--) sa[--ws[x[i]]]=i;
    for(j=1,p=1;p<n;j*=2,m=p)
    {
      for(p=0,i=n-j;i<n;i++) y[p++]=i;
      for(i=0;i<n;i++) if(sa[i]>=j) y[p++]=sa[i]-j;
      for(i=0;i<n;i++) wv[i]=x[y[i]];
      for(i=0;i<m;i++) ws[i]=0;
      for(i=0;i<n;i++) ws[wv[i]]++;
      for(i=1;i<m;i++) ws[i]+=ws[i-1];
      for(i=n-1;i>=0;i--) sa[--ws[wv[i]]]=y[i];
      for(t=x,x=y,y=t,p=1,x[sa[0]]=0,i=1;i<n;i++)
      x[sa[i]]=cmp(y,sa[i-1],sa[i],j)?p-1:p++;
    }
    return;
}
int rank[maxn],height[maxn];
void calheight(int *r,int *sa,int n)
{
    int i,j,k=0;
    for(i=1;i<=n;i++) rank[sa[i]]=i;
    for(i=0;i<n;height[rank[i++]]=k)
    for(k?k--:0,j=sa[rank[i]-1];r[i+k]==r[j+k];k++);
    return;
}
int RMQ[maxn];
int mm[maxn];
int best[20][maxn];
void initRMQ(int n)
{
    int i,j,a,b;
    for(mm[0]=-1,i=1;i<=n;i++)
    mm[i]=((i&(i-1))==0)?mm[i-1]+1:mm[i-1];
```

```
    for(i=1;i<=n;i++)  best[0][i]=i;
    for(i=1;i<=mm[n];i++)
    for(j=1;j<=n+1-(1<<i);j++)
    {
      a=best[i-1][j];
      b=best[i-1][j+(1<<(i-1))];
      if(RMQ[a]<RMQ[b])  best[i][j]=a;
      else  best[i][j]=b;
    }
    return;
}
int  askRMQ(int  a,int  b)
{
   int  t;
   t=mm[b-a+1];b-=(1<<t)-1;
   a=best[t][a];b=best[t][b];
   return  RMQ[a]<RMQ[b]?a:b;
}
int  lcp(int  a,int  b)
{
   int  t;
   a=rank[a];b=rank[b];
   if(a>b)  {t=a;a=b;b=t;}
   return(height[askRMQ(a+1,b)]);
}

int  r[  maxn  ],sa[  maxn  ];

int  main()
{
  int  i  ,  j  ,  n  ,  m  =  0  ;
  char  ch;
  scanf("%d",  &n  );
  getchar();
  for(  i  =  0  ;  i  <=  n  ;  ++  i  )
    {
      scanf("%c",  &ch  );
      r[  i  ]  =  ch  -  48;
      m  =  m  >  r[  i  ]  ?  m  :  r[  i  ]  +  1  ;
    }
  for(  i  =  0  ;  i  <  n  ;  ++  i  )  printf("%d  ",r[  i  ]);printf("\n");
  //r[  n  ++  ]  =  0  ;
  da(  r  ,  sa  ,  n  ,  m  );
  /*
```

```
  for( i = 0 ; i < n ; ++ i )
    printf("%d ",sa[ i ]);
  printf("\n");
  */
  calheight( r , sa , n );
  initRMQ( n );
  while( 1 )
    {
      int a,b;
      scanf("%d%d",&a,&b);
      printf("%d %d\n", askRMQ( a , b ) , lcp( a , b ));
    }
  return 0;
}
```

DC3

下面的代码计算 height 数组之后计算最长公共前缀(LCP)。

```
#include<cstdlib>
#include<cstdio>
#define maxn 1000003
#define F(x) ((x)/3+((x)%3==1?0:tb))
#define G(x) ((x)<tb?(x)*3+1:((x)-tb)*3+2)
int wa[maxn],wb[maxn],wv[maxn],ws[maxn];
int c0(int *r,int a,int b)
{return
r[a]==r[b]&&r[a+1]==r[b+1]&&r[a+2]==r[b+2];}
int c12(int k,int *r,int a,int b)
{if(k==2) return r[a]<r[b]||
r[a]==r[b]&&c12(1,r,a+1,b+1);
 else return r[a]<r[b]||r[a]==r[b]&&wv[a+1]<wv[b+1];}
void sort(int *r,int *a,int *b,int n,int m)
{
   int i;
   for(i=0;i<n;i++) wv[i]=r[a[i]];
   for(i=0;i<m;i++) ws[i]=0;
   for(i=0;i<n;i++) ws[wv[i]]++;
   for(i=1;i<m;i++) ws[i]+=ws[i-1];
   for(i=n-1;i>=0;i--) b[--ws[wv[i]]]=a[i];
   return;
}
void dc3(int *r,int *sa,int n,int m)
{
   int i,j,*rn=r+n,*san=sa+n,ta=0,tb=(n+1)/3,tbc=0,p;
   r[n]=r[n+1]=0;
   for(i=0;i<n;i++) if(i%3!=0) wa[tbc++]=i;
   sort(r+2,wa,wb,tbc,m);
   sort(r+1,wb,wa,tbc,m);
   sort(r,wa,wb,tbc,m);
```

```
   for(p=1,rn[F(wb[0])]=0,i=1;i<tbc;i++)
   rn[F(wb[i])]=c0(r,wb[i-1],wb[i])?p-1:p++;
   if(p<tbc) dc3(rn,san,tbc,p);
   else for(i=0;i<tbc;i++) san[rn[i]]=i;
   for(i=0;i<tbc;i++) if(san[i]<tb) wb[ta++]=san[i]*3;
   if(n%3==1) wb[ta++]=n-1;
   sort(r,wb,wa,ta,m);
   for(i=0;i<tbc;i++) wv[wb[i]=G(san[i])]=i;
   for(i=0,j=0,p=0;i<ta && j<tbc;p++)
   sa[p]=c12(wb[j]%3,r,wa[i],wb[j])?wa[i++]:wb[j++];
   for(;i<ta;p++) sa[p]=wa[i++];
   for(;j<tbc;p++) sa[p]=wb[j++];
   return;
}
int rank[maxn],height[maxn];
void calheight(int *r,int *sa,int n)
{
   int i,j,k=0;
   for(i=1;i<=n;i++) rank[sa[i]]=i;
   for(i=0;i<n;height[rank[i++]]=k)
   for(k?k--:0,j=sa[rank[i]-1];r[i+k]==r[j+k];k++);
   return;
}
int RMQ[maxn];
int mm[maxn];
int best[20][maxn];
void initRMQ(int n)
{
   int i,j,a,b;
   for(mm[0]=-1,i=1;i<=n;i++)
   mm[i]=((i&(i-1))==0)?mm[i-1]+1:mm[i-1];
   for(i=1;i<=n;i++) best[0][i]=i;
   for(i=1;i<=mm[n];i++)
   for(j=1;j<=n+1-(1<<i);j++)
   {
    a=best[i-1][j];
    b=best[i-1][j+(1<<(i-1))];
    if(RMQ[a]<RMQ[b]) best[i][j]=a;
    else best[i][j]=b;
   }
   return;
}
int askRMQ(int a,int b)
{
  int t;
  t=mm[b-a+1];b-=(1<<t)-1;
  a=best[t][a];b=best[t][b];
  return RMQ[a]<RMQ[b]?a:b;
}
int lcp(int a,int b)
{
  int t;
```

```
  a=rank[a];b=rank[b];
  if(a>b) {t=a;a=b;b=t;}
  return(height[askRMQ(a+1,b)]);
}

int r[ maxn ] , sa[ maxn ];

int main()
{
 int i , j , n , m = 0 ;
 char ch;
 scanf("%d", &n );
 getchar();
 for( i = 0 ; i <= n ; ++ i )
  {
   scanf("%c", &ch );
   r[ i ] = ch - 48;
   m = m > r[ i ] ? m : r[ i ] + 1 ;
  }
 for( i = 0 ; i < n ; ++ i ) printf("%d ",r[ i ]);printf("\n");
 //r[ n ++ ] = 0 ;
 dc3( r , sa , n , m );



 for( i = 0 ; i < n ; ++ i )
  printf("%d ",sa[ i ]);
 printf("\n");

 /*
 calheight( r , sa , n );
 for( i = 0 ; i < n ; ++ i ) printf("%d ",rank[ i ]);printf("\n");
 initRMQ( n );
 printf("!!!!\n");
 while( 1 )
  {
   int a,b;
   scanf("%d%d",&a,&b);
   printf("%d %d\n", askRMQ( a , b ) , lcp( a , b ));
  }
 */
 return 0;
}
```

KMP 及其应用*

KMP 算法注记

所谓 KMP 简言之可以理解为先用 pattern 匹配自己得出来如果自己与自己的**前一个(这个可以理解为向前错一位以后的匹配结果)**不匹配的话前面的最长匹配的长度是多少。这个就是所谓的 pie 函数，做好这个函数之后再和文本去用同样的方法匹配就可以了，因为如果不匹配的话就会产生一个类似

于自动机的过程去跳转到上一个最长匹配的串的长度，所以可以一直往下走。所谓 pie 函数也就是一个记录如果不匹配的话之前的最长匹配到哪里的一个数组。

利用 pie 数组的这个性质可以解决这样的问题：已知字符串 S 由一个字符串重复多次构成，问这个重复的字符串的最短长度（所谓的字符串的循环节，POJ 2406）[4]。

KMP 参考代码

hint:KMP 读入一直用的是 char 型的字符串从 1 开始读，因为如果从 0 开始读的话脚标有点不好处理不过最后还是解决了可以用 string 从 0 开始读入，参见 type2.

type1:

```
//The KMP algorthim
#include<iostream>
using std::cin;
using std::cout;

int* compute_prefix_fun(char P[])
 {
 int n,k,q;
 n=strlen(P+1);
 int *pie=new int[n+1];
 pie[1]=0;
 k=0;
 for(q=2;q<=n;q++)
  {
  while(k>0 and P[k+1] not_eq P[q])
    k=pie[k];
  if(P[k+1]==P[q])
    k++;
  pie[q]=k;
  }
 return pie;
}

void KMP(char T[],char P[])
 {
 int n,m,q,i,j;
 int *pie;
 n=strlen(T+1);
 m=strlen(P+1);
 pie=compute_prefix_fun(P);
 q=0;
 for(i=1;i<=n;i++)
  {
  while(q>0 and P[q+1] not_eq T[i])
    q=pie[q];
  if(P[q+1]==T[i])
    q++;
```

4 http://poj.org/problem?id=2406

```
   if(q==m)
      {
      cout<<"Parttern Shift Occured\n";
      q=pie[q];
      }
   }
 return;
}

int main()
{
char T[100],P[100];
cin>>T+1>>P+1;
KMP(T,P);
return 0;
}
```

type2:

```
/*
 KMP algorithm
 gestapolur
*/
#include<iostream>
#include<string>
#define MAXN 100005
using namespace std;

int pi[ MAXN ];
string text , pattern;
int n , m;

void prefix()
{
 int k;

 k = -1;
 pi[ 0 ] = -1;

 for( int i = 1 ; i < m ; ++ i )
  {
   while( k >= 0 and pattern[ k + 1 ] not_eq pattern[ i ]
)
     k = pi[ k ];
   if( pattern[ k + 1 ] == pattern[ i ] )
     ++ k;
   pi[ i ] = k;

  }

 return ;
}

void kmp()
{
 int q;
```

```
 q = -1;

 for( int i = 0 ; i < n ; ++ i )
  {
   while( q >= 0 and pattern[ q + 1 ] not_eq text[ i ] )
    q = pi[ q ];
   if( pattern[ q + 1 ] == text[ i ] )
    ++ q;

   if( q == m - 1 )
    {
     cout<< i + 1 <<"\n";
     //cout<<"pattern shift occured\n";
     q = pi[ q ];
    }
  }

 return ;
}

void init()
{
 cin>>text>>pattern;

 n = text.size();
 m = pattern.size();
 return ;
}

int main()
{
 init();

 prefix();

 kmp();

 return 0;
}
```

KMP-Extend(扩展 KMP)*

RMQ(區間極值問題)

RMQ 问题的 sparse table 算法

ST 算法的解释：

dp 过程，选取[i,i+2^j-1],[i+2^j,i-1]的极值。边界条件是 i + 2^j -1 <= n

对于查询的解释：f[i,j] = max(f[i,k]，f[j - 2^k + 1，k])比较的是区间[i,i+2^k-1]和[j-2^k + 1，j]的大小，这两段区间必然会覆盖区间

[i,j],因为 2^k >= (j - i + 1) / 2.所以能够取到该段区间的最大值。

Binary Index Tree(树状数组)

树状数组用来计算一段序列一段连续区间的和，可以快速更改序列某个元素的值

2^k=i&(i^(i-1)) 标号为最右边的 1 的从左往右数的位置。
tree[lowbit(x)]记录 data[x]压缩后的值。计算 1..x 的和的过程就是在沿树上行。

-x 是 x 的反码+1.

x&-x 是取最右边的 1，这个操作即是树状数组的 lowbit 操作

计算逆序对的方法

TIMUS1090

```
//TIMUS 1090
/*
the follow is a kid of merge sort that count the
invesertion part.

#include<iostream>
#define MAXN 100006
using std::cin;
using std::cout;
int a[MAXN];
int tot=0,n,mt=0,ans;

void merge(int l,int r)
{
 if(l>=r)
  return ;
 int i,j,k;
 int m=l+r>>1;
 int n1=m-l+1;
 int n2=r-m;
 int *L=new int [n1+1];
 int *R=new int [n2+1];
 merge(l,m);
 merge(m+1,r);

 for(i=l;i<=m;i++)
  L[i-l]=a[i];
 for(j=m+1;j<=r;j++)
  R[j-m-1]=a[j];

 i=0,j=0;

 for(k=l;k<=r;k++)
  if(i<n1 and j<n2)
```

```
    {
     if(L[i] < R[j])
         {
           a[k]=L[i];
           i++;
         }
     else
       {
         tot+=i+1;
         a[k]=R[j];
         j++;
       }
    }
  else
    break;

 if(i== n1 and j < n2)
  while(j<n2)
    a[k++]=R[j++];
 if(i<n1 and j==n2)
  while(i<n1)
    a[k++]=L[i++];

 free(L);
 free(R);

}
```

```
int main()
{
 int i,j,k;


 cin>>n>>k;
 for(j=1;j<=k;j++)
  {
   for(i=1;i<=n;i++)
    cin>>a[i];
   merge(1,n);
   if(tot>mt)
    mt=tot,ans=j;
   tot=0;
  }
 cout<<ans<<"\n";



 cin>>n;
 for(i=1;i<=n;i++)
  cin>>a[i];
 merge(1,n);
 for(i=1;i<=n;i++)
  cout<<a[i]<<" ";
 cout<<"\n";
 cout<<tot<<"\n";
```

```
 return 0;
}
*/

#include<iostream>
#define MAXN 10002
using std::cin;
using std::cout;

int n,k,max=-1,ans;
int idt[MAXN];

int L(int k)
{return (k bitand -k);}

int getsum(int idx)//srch
{
 int s=0;
 while(idx>0)
   {
    s+=idt[idx];
    idx-=L(idx);
    //idx &= idx - 1;
   }
 return s;
}

void bit_update(int idx)//add
{
 while(idx<=n)
   {
    ++idt[idx];//the count value is 1
    idx+= l(idx);
    //idx+= idx & -idx;
   }
 return ;
}

int main()
{
 int i,j,sum,a;
 cin>>n>>k;
 for(i=1;i<=k;i++)
   {
    sum=0;
    for(j=0;j<=n;j++)
     idt[j]=0;

    for(j=0;j<n;j++)
     {
       cin>>a;
       a=n+1-a;//why inversion here?
       sum+=getsum(a);
       bit_update(a);
     }
```

```
   if(sum > max)
    {max = sum;ans=i;}
  }

 cout<<ans<<"\n";
 return 0;
}
```

树状数组的应用

Lowest Common Ancestor（最低公共祖先)*

用 DFS 的方式转化为 RMQ 问题或者±1RMQ[5]，但是后者感觉好复杂一般 RMQ 的 ST 算法就够了吧。

Segmental Tree(線段樹)

線段樹的離散化

如果要維護一棵至多有 10^5 個元素的線段樹，然而每個區間至多到 10^9。顯然不能建立一個[1,10^9]的線段樹，如果預先讀入所有節點的操作，那麼就可以知道有多少個不同的節點，所以最多只用建立[1,10^5]的線段樹就可以了。

首先讀入所有的操作，可以預處理出所有的 add 操作一共有多少種不同的元素以及每個元素的大小，設這個值爲 T，那麼線段樹中就至多會有 T 個元素。先建立一個[1,T]的線段樹，然後對於進行操作的節點 x，可以用二分或者 std::lower_bound()找出它在線段樹當中的位置對其進行操作。

參考題目：
http://www.codeforces.com/problemset/problem/85/D

不遍历到根结点就可以维护线段树的方法

不遍历到根节点就能够维护的线段树的方法可以这么搞，在线段树结点的标记上弄一个标记，标记当前这个结点的两个孩子是否都有值，如果都有值就标记 1，否则是 2，没有结点有值为 0.

在加入一段区间的时候，如果恰好有一个结点的 L,R 符合加入结点的区间，则将它和它的孩子都标记为 1，否则如果这个结点只是在插入路径上的经过的结点，那么就说明他的某一个孩子上有值，所以将这个结点标记为 2.对标记为 2 的结点更新它的孩子之后，可能原先没有被线段占满的结点在更新的过程中被某条加入的线段占满了，所以加入完成之后如果发现这个结点的左右孩子的标记都为 1 的话，也将这个结点的标记更新为 1。

查找的时候如果发现标记为 1 的结点就返回这个要查找的值而不必遍历下面要找的那个结点，因为这个结点和它下面的子树都有值了，否则如果是 2 的话就沿树继续查找。

```
program segtree;
type treenode = record
            sum,l,r : longint;
```

5 國家集訓隊論文

```
          end;
var
  node     : array[0..2000000] of treenode;
  n,m      : longint;
  a,b      : longint;
  i      : integer;
  cin,cout : text;
procedure create(l,r,s: longint );
var
  mid : longint;
begin
  node[ s ].l := l;
  node[ s ].r := r;
  node[ s ].sum := 0;

  if l <> r then
  begin
    mid := (l + r) shr 1 ;
    create(l,mid,s shl 1 );
    create(mid + 1 , r , s shl 1 + 1 );
  end;
end; { create }

procedure insert(l,r,s : longint );
var
  mid : longint;
begin
  if (node[ s ].l = l) and ( node[ s ].r = r ) and
(node[ s ].sum <> 2) then
  begin
    node[ s ].sum := 1 ;
    node[ s shl 1 ].sum := 1 ;
    node[ s shl 1 or 1 ].sum := 1 ;
  end
  else
    node[ s ].sum := 2;

  if node[ s ].l = node[ s ].r then
    exit;

  mid := (node[ s ].l + node[ s ].r) shr 1;

  if mid >= r then
    insert( l , r , s shl 1 )
  else if mid < l then
    insert( l , r , s shl 1 or 1 )
  else
  begin
    insert( l , mid , s shl 1 );
    insert( mid + 1 , r , s shl 1 or 1 );
  end;
```

```
  if node[ s shl 1 ].sum = node[ s shl 1 or 1 ].sum then
    node[ s ].sum := node[ s shl 1 ].sum;

end; { insert }

function find(l,r,s : longint ):longint;
var
  mid : longint;
begin
  {writeln(l,' ',r,' ',s);}
  if ( node[ s ].l = l ) and ( node[ s ].r = r ) and
( node[ s ].sum <> 2 )then
  begin
    if node[ s ].sum = 1 then
      exit( r - l + 1 )
    else
      exit( 0 );
  end;
  if node[ s ].l = node[ s ].r then
    exit( 0 )
  else
  begin
    mid := (node[ s ].l + node[ s ].r) shr 1 ;
    if mid >= r then
      exit(find( l , r , s shl 1 ))
    else if mid < l then
      exit(find( l , r , s shl 1 + 1 ))
    else
      exit(find( l , mid , s shl 1 ) + find( mid + 1 , r , s shl 1
+ 1 ))
  end;
end;

begin
  assign(cin,'test.in');
  assign(cout,'out2.ans');
  reset(cin);
  rewrite(cout);
  while not eof do
  begin
    readln(n,m);
    create( 1 , n , 1 );
    for i := 1 to m do
    begin
      readln(cin,a,b);
      writeln(cout,find(a,b,1));
      insert(a,b,1);
    end;
    writeln(cout);
  end;
end.
```

Number Theory (数论问题)*

快速幂算法

```
/*
 quick power resolve
 2012-03-28
 gestapolur
*/
#include<cstdlib>
#include<cstdio>

int a , b , c;
/*
inline long long pow2( long long x )
{ return x * x; }
*/
long long resolve( int b )
{
 if( b == 2 )
  return a * a % c;
 else if( b == 1 )
  return a;
 else if( b & 1 )
  return resolve( b - 1 ) * a % c;
 else
  {
   long long t = resolve( b >> 1 );
   return t * t % c;
  }
}

int main()
{
 while( scanf("%d%d%d" , &a , &b , &c ) not_eq EOF )
  printf( "%Ld\n" , resolve( b ) );
 return 0;
}
```

矩陣快速冪

```
/*
 power A ^ P mod M
*/
#include<cstdio>
#include<cstring>
#define MAXN 32

int p , n , m;
int a[ MAXN ][ MAXN ] , b[ MAXN ][ MAXN ];

void mul( int a[ ][ MAXN ] , int b[ ][ MAXN ] )
{
 int i , j , k;
```

```
 int c[ MAXN ][ MAXN ];
 memset( c , 0 , sizeof( c ) );
 for( i = 1 ; i <= n ; ++ i )
  for( j = 1 ; j <= n ; ++ j )
   for( k = 1 ; k <= n ; ++ k )
    c[ i ][ j ] = ( c[ i ][ j ] + a[ i ][ k ] * b[ k ][ j ] % m ) % m;
 memcpy( a , c , sizeof( c ) );
 return ;
}

void q_power( long long q )
{
 if( q == 1 )
  {}
 else if( q & 1 )
  {
   q_power( q ^ 1 );
   mul( b , a );
  }
 else
  {
   q_power( q >> 1 );
   mul( b , b );
  }
 return ;
}
```

```
int main()
{
 int q;
 scanf("%d%d%d" , &n , &q , &m );
 for( int i = 1 ; i <= n ; ++ i )
  for( int j = 1 ; j <= n ; ++ j )
   {
    scanf("%d" , &a[ i ][ j ] );
    a[ i ][ j ] %= m;
   }
 memcpy( b , a , sizeof( b ) );
 q_power( q );
 for( int i = 1 ; i <= n ; ++ i )
  {
   for( int j =1 ; j <= n ; ++ j )
    printf("%d " , b[ i ][ j ] );
   printf("\n");
  }
 return 0;
}
```

example:（POJ 3233）

给出一个 n * n 矩阵 A 和一个正整数 k,求 $\mathbf{S} = \mathbf{A} + \mathbf{A}^2 + \mathbf{A}^3 + \ldots + \mathbf{A}^k$

擴展 Euclid 計算乘法逆元

計算組合數並取模

```
#include<cstdio>
#define MAXN 1000013
#define TOT 1000000
#define MOD 20000003

typedef long long LL ;

void ex_gcd(LL a , LL b , LL& x , LL& y){
  if( b==0 ){
    x = 1 ; y = 0 ; return  ;
  }
  ex_gcd( b, a%b , x , y)  ;
  LL t = x ; x = y ; y = t - a / b * y ;
}

long long ny(long long a)
{
  LL x, y ;
  ex_gcd( a, MOD , x, y)  ;
  x %= MOD ;
  x = ( x + MOD ) % MOD ;
  return x;
}

inline long long C( int m , int n )//combinator C( m , n )
{
  //printf( "%lld C\n" , ( g[ n ] / ( g[ m ] * g[ n - m ] ) ) );
  long long ff=( g[ m ] * g[ n - m ] )%MOD;
  ff=ny(ff);
  return ( g[ n ] * ff )%MOD;
  //return res;
}
```

Combinatoric Problems(組合數學問題)

位运算计算 N 皇后问题解的数目

[6]

```
/*
ID: gestapo1
PROG: checker
LANG: C++
*/

#include<fstream>
//#include<iostream>
//#include<windows.h>
#define N 15
/*
using std::cin;
```

6 方法来自 matrix67's blog

```
using std::cout;
*/
using std::ifstream;
using std::ofstream;
ifstream cin("checker.in");
ofstream cout("checker.out");


int a[N],n,pt=0,ul;

void compute(int row, int ld, int rd)
   {
  int pos,p;
  if (row not_eq ul)
    {
  pos=ul bitand ~(row bitor ld bitor rd);
  while (pos not_eq 0)
     {
    p=pos bitand -pos;
    pos-=p;
    compute(row+p,ld+p<<1,rd+p>>1);
     }
    }
  else
    ++pt;
   }
```

```
bool trys(int i,int k)
{
 int j=1;
 while(j<k)
  {
  if((a[j]==i)||(abs(a[j]-i)==abs(j-k)))
      {return false;} //CAN NOT PLACE.
  ++j;
  }
 return true;//COULD PLACE.
}

void place(int k)
{
 int i;
 if(k>n)
  {
  if(pt<3)
     {
    for(i=1;i<=n;i++)
      if(a[i]!=0)
         {
        cout<<a[i];
        if(i not_eq n)
         cout<<" ";
         }
```

```
    cout<<"\n";
      }
  else
   return ;
  pt++;
  }
 else
  {
  if(pt>=3)
   return ;
  for(i=1;i<=n;i++)// A line number.
   if (trys(i,k))
     {
     a[k]=i;
      place(k+1);
     }
  }
 return ;
}

int main()
{
  cin>>n;
  place(1);//use place.
  ul=(1<<n)-1;
  pt=0;
  compute(0,0,0);
  cout<<pt<<"\n";

  cin.close();
  cout.close();
  return 0;
}
```

Shell Command tips

一个通用的对拍脚本

```
# A general data test bash
# 2012-04-05
# write by gestapolur
# <prog1_name> <prog2_name> <generator>
# plz use bash to run this script e.g: bash ./<this_script>
# $1 = prog1 $2 = prog2 $3 = generator

./$3 > test.in
./$1 < test.in > $1.out
./$2 < test.in > $2.out
diff $1.out $2.out

while [" `diff $1.out $2.out` " == ""]
do
   ./$3 > test.in
   ./$1 < test.in > $1.out
```

```
  ./$2 < test.in > $2.out
  diff $1.out $2.out
  echo "test okay"
done
```

文件流读入控制和对拍的简单操作

```
while ["`diff out1.out out2.out `" == ""]
do
  ./gen > input.in
  ./prob1 <./input.in > ./out1.out
  ./prob2 <./input.in > ./out2.out
  echo "test okay"
done
```

if 使用方法

```
if [<expression1>] <operator > < = <> >
[<expression2>];
<...>
endif
```

while 循环的使用

```
declare -i cnt
cnt=1
while [ $cnt != 10 ]
do
  cnt=$[$cnt+1]
  echo $cnt
done
```

C/CPP tips

g++ 的编译命令 -D 可以定义宏 比如 -DmarcoName 就定义了一个名为 marcoName 的宏

I/O Methods

*注意输出一个序列时最后有可能多输出一个空格导致 WA

*getline(),getchar(),读入前如果有其他的输入需要相应的 getline()来吃掉换行

*每行输出的要求

*空格的读入，getline(),gets()都可以，但是 gets()在某些版本的 g++下会报警

*G++版本问题

*cin,cout 取消同步：bool sync_with_stdio(bool sync = false);

构造函数

Python Tips

现在 ubuntu 上预装的是 python2.7+的版本，和 python3 有一些不同，所以这里还是用 python2.7+的版本作为例子好了。

Console I/O

```
#python3
var_name = input()
splite()#delete some char
```

文件读写

```
f = open('<filename>' , '<w/r/r+>')#r+ could both read & write
f.write(<str>)#write can only output string, but could change others to string( str(<variety_name>)) before output.
```

随机数

```
>>> random.random()       # Random float x, 0.0 <= x < 1.0
0.37444887175646646
>>> random.uniform(1, 10)  # Random float x, 1.0 <= x < 10.0
1.1800146073117523
>>> random.randint(1, 10)  # Integer from 1 to 10, endpoints included
7
>>> random.randrange(0, 101, 2)  # Even integer from 0 to 100
26
>>> random.choice('abcdefghij')  # Choose a random element
'c'

>>> items = [1, 2, 3, 4, 5, 6, 7]
>>> random.shuffle(items)
>>> items
[7, 3, 2, 5, 6, 4, 1]

>>> random.sample([1, 2, 3, 4, 5],  3)  # Choose 3 elements
[4, 1, 5]
```

Java Tips

Console I/O

读入以空格分隔的数据

使用 Scanner.nextInt() , Scanner.next()

注意读入的异常抛出 throws Exception

```
//Console I/O sample
import java.io.*;
import java.util.Scanner;

public class test {
  public static void main( String[] args ) throws Exception
{

    String ca , cb , line;
    int i , n;
    int []arr = new int[ 100 ];

    Scanner in = new Scanner( System.in );

    ca = in.next();
    cb = in.nextLine();
    line = in.nextLine();
    n = in.nextInt();
    for( i = 1 ; i <= n ; ++ i )
      arr[ i ] = in.nextInt( );
    System.out.printf( "%d\n" , n );
    for( i = 1 ; i <= n ; ++ i )
      System.out.printf( "%d " , arr[ i ] );
    System.out.printf( "\n" );
  }
}
```

读取整行的数据

文件读写

```
import java.io.*;

public class fileio{
  public static void main(String[] args) throws Exception {

    BufferedReader br = null;

    String sCurrentLine;

    br = new BufferedReader(new
FileReader("./input.in"));

    while ((sCurrentLine = br.readLine()) != null)
      System.out.println(sCurrentLine);
```

```
  }
}
```

高精度调用

```
import java.io.*;
import java.util.Scanner;
import java.math.BigDecimal;//高精度实数
/*
构造器
BigDecimal(double val)
      Translates a double into a BigDecimal.
BigDecimal(String val)
      Translates the String repre sentation of a BigDecimal
into a BigDecimal.

常用方法
a.add(b);
a.subtract(b);
a.multiply(b);
a.divide(b,scale,BigDecimal.ROUND_HALF_UP);
a.compareTo(b);
a.doubleValue();
a.toString();
*/
public class test {
  public static void main( String[] args ) throws Exception
{
  String ca , cb;
  BigDecimal a , b;

  Scanner in = new Scanner( System.in );

  ca = in.next();
  cb = in.next();

  a = new BigDecimal( ca );
  b = new BigDecimal( cb );

  /*
   a.add(b);
   a.subtract(b);
   a.multiply(b);
   a.divide(b,scale,BigDecimal.ROUND_HALF_UP);
   a.compareTo(b);
   a.doubleValue();
   a.toString();
  */

  System.out.println( a.add( b ) );
  System.out.println( a.multiply( b ) );
  System.out.println( a.divide( b , 10 ,
```

```
BigDecimal.ROUND_HALF_UP ) );

   }
}
```