Contents

0.1 myvimrc.txt

```
" 檔案編碼
set fileencodings=utf8,gb18030,big5,latin1,default
set fileencoding=big5
set <C-u>=^U
set <C-b>=^B
map <C-u> :set fileencoding=utf8
map <C-b> :set fileencoding=gb18030

" 編輯喜好設定
syntax on        " 語法上色顯示
let g:kolor_italic=1                    " Enable italic. Default: 1
let g:kolor_bold=1                      " Enable bold. Default: 1
let g:kolor_underlined=0                " Enable underline. Default: 0
let g:kolor_alternative_matchparen=0    " Gray 'MatchParen' color. Default: 0
colorscheme werks
set nocompatible " VIM 不使用和 VI 相容的模式
"set ai           " 自動縮排
set shiftwidth=4 " 設定縮排寬度 = 4
set tabstop=4    " tab 的字元數
set softtabstop=4
set expandtab   " 用 space 代替 tab

set ruler        " 顯示右下角設定值
set backspace=2  " 在 insert 也可用 backspace
set ic           " 設定搜尋忽略大小寫
set ru           " 第幾行第幾個字
set incsearch    " 在關鍵字還Ｆ完全輸入完畢前就顯示結果
set smartindent  " 設定 smartindent
set confirm      " 操作過程有衝突時, 以明確的文字來詢問
set history=100  " 保留 100 個使用過的指令
set nu
set clipboard=unnamedplus
```

1 Black Magic

1.1 Black Magic/ranktree.cpp

```
#include <bits/extc++.h>
using namespace __gnu_pbds;
typedef tree<int,null_type,less<int>,rb_tree_tag,tree_order_statistics_node_update> set_t;
int main()
{
  // Insert some entries into s.
  set_t s;
  s.insert(12);
  s.insert(505);

  // The order of the keys should be: 12, 505.
  assert(*s.find_by_order(0) == 12);
  assert(*s.find_by_order(3) == 505);
}
```

2 Computational Geometry

2.1 Computational Geometry/Computational Geometry.cpp

```
int getCircleCircleIntersection(Circle C1,Circle C2,vector<Point>& sol)
{
  DB d=Length(C1.c-C2.c);
  if(dcmp(d)==0)
  {
    if(dcmp(C1.r-C2.r)==0) return -1;
    return 0;
  }
  if(dcmp(C1.r+C2.r-d)<0) return 0;
  if(dcmp(fabs(C1.r-C2.r)-d)>0) return 0;

  DB a=angle(C2.c-C1.c);
  DB da= acos((C1.r*C1.r+d*d-C2.r*C2.r)/(2*C1.r*d));
  Point p1=C1.point(a-da),p2=C1.point(a+da);

  sol.push_back(p1);
```

```
  if(p1==p2) return 1;
  sol.push_back(p2);
  return 2;
}

int PointCircleTangents(Point p,Circle C,Vector* v)
{
  Vector u=C.c-p;
  DB dist=Length(u);
  if(dist<C.r) return 0;
  else if(dcmp(dist-C.r)==0)
  {
    v[0]=Rotate(u,PI/2);
    return 1;
  }
  else
  {
    DB ang=asin(C.r/dist);
    v[0]=Rotate(u,-ang);
    v[1]=Rotate(u,ang);
    return 2;
  }
}

int CircleCircleTangents(Circle A,Circle B,Point* a,Point *b)
{
  int cnt=0;
  if(A.r<B.r) {swap(A,B);swap(a,b);}
  int d2=(A.c.x-B.c.x)*(A.c.x-B.c.x) + (A.c.y-B.c.y)*(A.c.y-B.c.y);
  int rdiff=A.r-B.r;
  int rsum=A.r+B.r;
  if(d2<rdiff*rdiff) return 0;

  DB base=atan2(B.c.y-A.c.y,B.c.x-A.c.x);
  if(d2==0 && A.r==B.r) return -1;
  if(d2==rdiff*rdiff)
  {
    a[cnt]=A.point(base);b[cnt]=B.point(base);cnt++;
    return 1;
  }

    DB ang = acos((A.r-B.r)/sqrt(d2));
  a[cnt]=A.point(base+ang);b[cnt]=B.point(base+ang);cnt++;
  a[cnt]=A.point(base-ang);b[cnt]=B.point(base-ang);cnt++;
  if(d2==rsum*rsum)
  {
    a[cnt]=A.point(base);b[cnt]=B.point(PI+base);cnt++;
  }
  else if(d2>rsum*rsum)
  {
    DB ang=acos((A.r+B.r)/sqrt(d2));
    a[cnt]=A.point(base+ang);b[cnt]=B.point(PI+base+ang);cnt++;
    a[cnt]=A.point(base-ang);b[cnt]=B.point(PI+base-ang);cnt++;
  }
  return cnt;
}

///x=rcos(theta)cos(phi)
///y=rcos(theta)sin(phi)
///z=rsin(theta)
DB torad(DB deg)
{
  return deg/180 * acos(-1);
}
void get_coord(DB R,DB lat,DB lng,DB &x,DB &y,DB &z)
{
  lat=torad(lat);
  lng=torad(lng);
  x=R*cos(lat)*cos(lng);
  y=R*cos(lat)*sin(lng);
  z=R*sin(lat);
}

typedef vector<Point> Polygon;
int isPointInPolygon(Point p,Polygon poly)
{
  int wn=0;
  int n=poly.size();
  for(int i=0;i<n;i++)
  {
    if(OnSegment(p,poly[i],poly[(i+1)%n])) return -1;
    int k=dcmp(Cross(poly[i]-p,poly[(i+1)%n]-p));
```

```
    int d1=dcmp(poly[i].y-p.y);
    int d2=dcmp(poly[(i+1)%n].y-p.y);
    if(k>0&&d1<=0 && d2>0) wn++;
    if(k<0&&d2<=0 && d1>0) wn--;
  }
  if(wn!=0) return 1;
  return 0;
}
int ConvexHull(Point* p,int n,Point* ch)
{
  sort(p,p+n);
  int m=0;
  for(int i=0;i<n;i++)
  {
    while(m>1 && Cross(ch[m-1]-ch[m-2],p[i]-ch[m-2])<=0) m--;
    ch[m++]=p[i];
  }
  int k=m;
  for(int i=n-2;i>=0;i--)
  {
    while(m > k && Cross(ch[m-1]-ch[m-2],p[i]-ch[m-2])<=0) m--;
    ch[m++]=p[i];
  }
  if(n>1) m--;
  return m;
}
```

2.2 Computational Geometry/HalfplaneIntersection.cpp

```
bool OnLeft(const Line& L,const Point& p)
{
  return Cross(L.v,p-L.P)>0;
}
Point GetIntersection(Line a,Line b)
{
  Vector u=a.P-b.P;
  DB t=Cross(b.v,u)/Cross(a.v,b.v);
  return a.P+a.v*t;
}
int HalfplaneIntersection(Line* L,int n,Point* poly)
{
  sort(L,L+n);

  int first,last;
  Point *p= new Point[n];
  Line *q=new Line[n];
  q[first=last=0] = L[0];
  for(int i=1;i<n;i++)
  {
    while(first < last && !OnLeft(L[i],p[last-1])) last--;
    while(first < last && !OnLeft(L[i],p[first])) first++;
    q[++last]=L[i];
    if(fabs(Cross(q[last].v,q[last-1].v))<eps)
    {
      last--;
      if(OnLeft(q[last],L[i].P)) q[last]=L[i];
    }
    if(first < last) p[last-1]=GetIntersection(q[last-1],q[last]);
  }
  while(first<last && !OnLeft(q[first],p[last-1])) last--;
  if(last-first<=1) return 0;
  p[last]=GetIntersection(q[last],q[first]);

  int m=0;
  for(int i=first;i<=last;i++) poly[m++]=p[i];
  return m;
}
```

3 Data Structure

3.1 Data Structure/DominatorTree.cpp

```
namespace DominatorTree
{
    int dfn[maxn], fa[maxn], id[maxn], dfs_clock;
    int semi[maxn], dsu[maxn], idom[maxn], best[maxn];
```

```
    int findset(int x)
    {
        if(x==dsu[x]) return x;
        int y=findset(dsu[x]);
        if(semi[best[x]] > semi[best[dsu[x]]]) best[x]=best[dsu[x]];
        return dsu[x]=y;
    }
    void dfs(int now, const vi* G)
    {
        id[dfs_clock]=now;
        dfn[now]=dfs_clock++;
        for (const auto &to:G[now]) if (dfn[to]==-1)
        {
            dfs(to,G);
            fa[dfn[to]] = dfn[now];
        }
    }
    void tarjan(const vi* pre,vi* dom)
    {
        for(int i=dfs_clock-1;i;i--)
        {
            int now=id[i];
            for(auto j:pre[now]) if(dfn[j]!=-1)
            {
                j=dfn[j];
                findset(j);
                getmin(semi[i],semi[best[j]]);
            }
            dom[semi[i]].PB(i);

            int x = dsu[i] = fa[i];
            for(const auto& z: dom[x])
            {
                findset(z);
                if(semi[best[z]] < x) idom[z] = best[z];
                else idom[z] = x;
            }
            dom[x].clear();
        }
        for(int i=1;i<dfs_clock;i++)
        {
            if(semi[i]!=idom[i]) idom[i]=idom[idom[i]];
            dom[idom[i]].PB(i);
        }
    }
    void build(int n, int s, const vi* const suc, const vi* const pre, vi* dom)
    {
        for(int i=0;i<n;i++)
        {
            dfn[i]=-1;
            dom[i].clear();
            dsu[i]=best[i]=semi[i]=i;
        }
        dfs_clock = 0;
        dfs(s, suc);
        tarjan(pre, dom);
    }
}
```

3.2 Data Structure/IntervalTree2D.cpp

```
struct SegmentTree2D
{
  int Max[maxn][maxn],Min[maxn][maxn],n,m;
  int xo,xleaf,x1,x2,y1,y2,x,y,v,vmax,vmin;
  void query1D(int o,int l,int r)
  {
    if(y1<=l && r<=y2) {vmax=max(Max[xo][o],vmax);vmin=min(Min[xo][o],vmin);return;}
    int mid=l+(r-l)/2;
    if(y1<=mid) query1D(o*2,l,mid);
    if(mid<y2) query1D(o*2+1,mid+1,r);
  }
  void query2D(int o,int l,int r)
  {
    if(x1<=l && r<=x2) {xo=o;query1D(1,1,m);return;}
    int mid=l+(r-l)/2;
    if(x1<=mid) query2D(o*2,l,mid);
    if(mid<x2) query2D(o*2+1,mid+1,r);
  }
  void modify1D(int o,int l,int r)
```

```
  {
    if(l==r)
    {
      if(xleaf) {Max[xo][o]=Min[xo][o]=v;return;}
      Max[xo][o]=max(Max[xo*2][o],Max[xo*2+1][o]);
      Min[xo][o]=min(Min[xo*2][o],Min[xo*2+1][o]);
      return;
    }
    int mid=l+(r-l)/2;
    if(y<=mid) modify1D(o*2,l,mid);
    else modify1D(o*2+1,mid+1,r);

    Max[xo][o]=max(Max[xo][o*2],Max[xo][o*2+1]);
    Min[xo][o]=min(Min[xo][o*2],Min[xo][o*2+1]);
  }
  void modify2D(int o,int l,int r)
  {
    if(l==r) {xo=o;xleaf=1;modify1D(1,1,m);return;}
    int mid=l+(r-l)/2;
    if(x<=mid) modify2D(o*2,l,mid);
    else modify2D(o*2+1,mid+1,r);
    xo=o;xleaf=0;modify1D(1,1,m);
  }
  void modify() {modfiy2D(1,1,n);}
  void query() {vmin=INF;vmax=-INF;query2D(1,1,n);}
};
```

3.3 Data Structure/Persistent Treap.cpp

```
struct Node
{
  Node* ch[2];
  int v,s;
  Node(int v):v(v),s(1) {ch[0]=ch[1]=NULL;}
  Node(Node* a) {*this=*a;}
  #define size(x) ((x)?((x)->s):0)
  void maintain() {s=size(ch[0])+size(ch[1])+1;}
}*root[maxn];
inline bool randchoice(DB probability) {return (DB)rand() / RAND_MAX <= probability;}
Node* merge(Node *left,Node *right)
{
  if(!left) return left;
  if(!right) return right;
  Node* res;
  if(randchoice((DB)size(left)/(size(left)+size(right))))
  {
    res=new Node(left->v);
    res->ch[0]=left->ch[0];
    res->ch[1]=merge(left->ch[1],right);
  }
  else
  {
    res=new Node(right->v);
    res->ch[0]=merge(left,right->ch[0]);
    res->ch[1]=right->ch[1];
  }
  res->maintain();
  return res;
}

void split(Node* o,int k,Node* &left,Node* &right)
{
  if(!k) {left=right=NULL;return;}
  Node *p=new Node(k);
  if(size(o->ch[0])<k)
  {
    split(o->ch[1],k-size(o->ch[0])-1,left,right);
    p->ch[1]=left;
    p->maintain();
    left=p;
  }
  else
  {
    split(o->ch[0],k,left,right);
    p->ch[0]=right;
    p->maintain();
    right=p;
  }
}
```

3.4 Data Structure/Splay tree.cpp

```
struct Node
{
  Node* ch[2];
  int v;
  int s;
  int cmp(int k)
  {
    int d=k-ch[0]->s;
    if(d==1) return -1;
    return d<=0? 0:1;
  }
  void maintain()
  {
    s=1;
    if(ch[0]!=NULL) s+=ch[0]->s;
    if(ch[1]!=NULL) s+=ch[1]->s;
  }
};
void rotate(Node* &o,int d)
{
  Node* k = o->ch[d^1]; o->ch[d^1] = k->ch[d]; k->ch[d] = o;
  o->maintain(); k->maintain(); o = k;
}
void splay(Node* &o,int k)
{
  int d=o->cmp(k);
  if(d==1) k-=o->ch[0]->s+1;
  if(d!=-1)
  {
    Node* p=o->ch[d];
    int d2=p->cmp(k);
    int k2=(d2==0?k:k-p->ch[0]->s-1);
    if(d2!=-1)
    {
      splay(p->ch[d2],k2);
      if(d==d2) rotate(o,d^1);else rotate(o->ch[d],d);
    }
    rotate(o,d^1);
  }
}

Node* merge(Node* left,Node* right)
{
  splay(left,left->s);
  left->ch[1]=right;
  left->maintain();
  return left;
}

void split(Node* o,int k,Node* &left,Node* &right)
{
  splay(o,k);
  left=o;
  right=o->ch[1];
  o->ch[1]=NULL;
  left->maintain();
}
```

3.5 Data Structure/Treap(Based on split and merge).cpp

```
unsigned ran()
{
  static unsigned x=20150214;
  return x=x*0xdefaced+1;
}
struct Node
{
  static Node mem[N], *pmem;
  Node *l,*r;
  int pri,val,size,Max;
  Node() {}
  Node(int _val):l(NULL),r(NULL),pri(ran()),val(_val),size(1) {}
} Node::mem[N], *Node::pmem=Node::mem ;

int size(Node *t) {return t?t->size:0;}
int Max(Node *t){return t?t->Max:-INF;}
void pull(Node *t)
```

```
{
  t->size=size(t->l)+size(t->r)+1;
}
Node* merge(Node *a,Node *b)
{
  if(!a || !b) return a?a:b;
  else if(a->pri > b->pri)
  {
    a->r=merge(a->r,b);
    pull(a);
    return a;
  }
  else
  {
    b->l=merge(a,b->l);
    pull(b);
    return b;
  }
}

void split(Node *t,int k,Node* &a,Node* &b)
{
  if(!t) a=b=NULL;
  else if(t->val<=k)
  {
    a=t;
    split(t->r,k,a->r,b);
    pull(a);
  }
  else
  {
    b=t;
    split(t->l,k,a,b->l);
    pull(b);
  }
}

Node* insert(Node *t,int k)
{
  Node *a,*b;
  split(t,k,a,b);
  return merge(merge(a,new Node(k)),b);
}
Node* remove(Node *t,int k)
{
  Node *a,*b,*c;
  split(t,k-1,a,b);
  split(b,k,b,c);
  return merge(a,c);
}
int kth(Node *t,int k)
{
  if(k<=size(t->l)) return kth(t->l,k);
  else if(k==size(t->l)+1) return t->val;
  else return kth(t->r,k-size(t->l)-1);
}
```

4 Graph Theorem

4.1 Graph Theorem/Biconnected Component.cpp

```
struct Edge
{
  int u,v;
};
int pre[maxn],iscut[maxn],bccno[maxn],dfs_clock,bcc_cnt;
vector<int> G[maxn],bcc[maxn];
stack<Edge> S;

int dfs(int u,int fa)
{
  int lowu=pre[u]=++dfs_clock;
  int child=0;
  for(int i=0;i<G[u].size();i++)
  {
    int v=G[u][i];
    Edge e= (Edge){u,v};
    if(!pre[v])
```

```
    {
      S.push(e);
      child++;
      int lowv=dfs(v,u);
      lowu=min(lowu,lowv);
      if(lowv >= pre[u]) /// is cut vertex --> sons and grandsons are one bcc
      {
        iscut[u]=1;
        bcc_cnt++;bcc[bcc_cnt].clear();
        for(;;)
        {
          Edge x=S.top();S.pop();
          if(bccno[x.u]!=bcc_cnt){bcc[bcc_cnt].push_back(x.u);bccno[x.u]=bcc_cnt;}
          if(bccno[x.v]!=bcc_cnt){bcc[bcc_cnt].push_back(x.v);bccno[x.v]=bcc_cnt;}
          if(x.u==u&&x.v==v) break;
        }
      }
    }
    else if(pre[v]<pre[u] && v!=fa) {S.push(e);lowu=min(lowu,pre[v]);}
  }
  if(fa<0 && child==1) iscut[u]=0;
  return lowu;
}
void find_bcc(int n)
{
  memset(pre,0,sizeof(pre));
  memset(iscut,0,sizeof(iscut));
  memset(bccno,0,sizeof(bccno));
  dfs_clock=bcc_cnt=0;
  for(int i=1;i<=n;i++)
    if(!pre[i]) dfs(i,-1);
}
```

4.2 Graph Theorem/Chu-Liu.cpp

```
struct Edge{int from,to,w;};
struct Chu_Liu
{
  int n,size;
  vector<Edge> edges;
  void init(int n)
  {
    this->n=n;
    edges.clear();
  }
  void AddEdge(int from,int to,int w)
  {
    edges.push_back((Edge){from,to,w});
  }
  int fa[maxn],no[maxn],v[maxn],in[maxn];
  bool removed[maxn];
  int MDST(int s)
  {
    int w1=0,w2=0;
    memset(removed,0,sizeof removed);
    size=edges.size();
    for(;;)
    {
      w1=0;
      memset(fa,-1,sizeof fa);
      memset(no,-1,sizeof no);
      memset(v,-1,sizeof v);
      for(int i=0;i<n;i++) in[i]=INF;
      for(int i=0;i<size;i++)
      {
        Edge& e=edges[i];
        if(e.from!=e.to && in[e.to]>e.w)
        {
          fa[e.to]=e.from;
          in[e.to]=e.w;
        }
      }

      bool loop=0;
      for(int i=0;i<n;i++)
      {
        if(removed[i]||i==s) continue;
        if(fa[i]==-1) return -1;
        else w1+=in[i];
        int now=i;
```

```
        while(now!=-1 && v[now]==-1) v[now]=i,now=fa[now];
        if(now!=-1 && v[now]==i)
        {
          loop=1;
          int j=now;
          for(;;)
          {
            no[j]=now,removed[j]=1,w2+=in[j];
            j=fa[j];
            if(j==now) break;
          }
          removed[now]=0;
        }
      }
      if(!loop) break;

      for(int i=0;i<size;i++)
      {
        Edge& e=edges[i];
        if(no[e.to]>=0) e.w-=in[e.to],e.to=no[e.to];
        if(no[e.from]>=0) e.from=no[e.from];
        if(e.from==e.to) edges[i--]=edges[--size];
      }
    }
    return w1+w2;
  }
};
```

4.3 Graph Theorem/Dinic.cpp

```
struct Edge
{
  int from,to,cap,flow;
};
struct Dinic
{
  int n,s,t;
  vector<Edge> edges;
  vector<int> G[maxn];
  bool vis[maxn];
  int d[maxn],cur[maxn];
  void init(int n)
  {
    this->n=n;
    for(int i=0;i<n;i++) G[i].clear();
    edges.clear();
  }
  void AddEdge(int from,int to,int cap)
  {
    edges.push_back((Edge){from,to,cap,0});
    edges.push_back((Edge){to,from,0,0});
    int m=edges.size();
    G[from].push_back(m-2);
    G[to].push_back(m-1);
  }
  bool BFS()
  {
    memset(vis,0,sizeof(vis));
    queue<int> Q;
    Q.push(s);
    d[s]=0;
    vis[s]=1;
    while(!Q.empty())
    {
      int now=Q.front();Q.pop();
      for(int i=0;i<G[now].size();i++)
      {
        Edge& e=edges[G[now][i]];
        if(!vis[e.to] && e.cap>e.flow)
        {
          vis[e.to]=1;
          d[e.to]=d[now]+1;
          Q.push(e.to);
        }
      }
    }
    return vis[t];
  }

  int DFS(int now,int a)
```

```
  {
    if(now==t || a==0) return a;
    int flow=0,f;
    for(int& i=cur[now];i<G[now].size();i++)
    {
      Edge& e=edges[G[now][i]];
      if(d[now]+1==d[e.to] && (f=DFS(e.to,min(a,e.cap-e.flow)))>0)
      {
        e.flow+=f;
        edges[G[now][i]^1].flow-=f;
        flow+=f;
        a-=f;
        if(a==0) break;
      }
    }
    return flow;
  }
  int Maxflow(int s,int t)
  {
    this->s=s,this->t=t;
    int flow=0;
    while(BFS())
    {
      memset(cur,0,sizeof(cur));
      flow+=DFS(s,INF);
    }
    return flow;
  }
};
```

4.4 Graph Theorem/Kuhn-Munkres.cpp

```
struct Hungary
{
  int M,N;
  LL EDGE[500][500];
  LL WA[500],WB[500],LOW[500];
  int MATCH[500];
  bool VA[500],VB[500];
  bool Match(const int u)
  {
    if(VA[u])return false;
    VA[u]=true;
    for(int nxt=0;nxt<N;nxt++)
    {
      const LL &edge=EDGE[u][nxt];
      if(edge==WA[u]+WB[nxt])
      {
        VB[nxt]=true;
        if(MATCH[nxt]==-1||Match(MATCH[nxt]))
        {
          MATCH[nxt]=u;
          return true;
        }
      }
      else
      {
        assert(edge<WA[u]+WB[nxt]);
        getmin(LOW[nxt],WA[u]+WB[nxt]-edge);
      }
    }
    return false;
  }
  void Update()
  {
    LL low=INF;
    for(int i=0;i<N;i++)if(!VB[i])getmin(low,LOW[i]);
    assert(low!=INF);
    for(int i=0;i<M;i++)if(VA[i])WA[i]-=low;
    for(int i=0;i<N;i++)if(VB[i])WB[i]+=low;
  }
  void Solve()
  {
    for(int i=0;i<M;i++)
    {
      WA[i]=0;
      for(int j=0;j<N;j++)getmax(WA[i],EDGE[i][j]);
    }
    for(int i=0;i<N;i++)WB[i]=0,MATCH[i]=-1;
    for(int i=0;i<M;i++)
```

```
    {
      for(;;)
      {
        for(int j=0;j<M;j++)VA[j]=false;
        for(int j=0;j<N;j++)VB[j]=false,LOW[j]=INF;
        if(Match(i))break;
        else Update();
      }
    }
  }
}
```

4.5 Graph Theorem/Link_Cut_Tree.cpp

```
const int maxnode=3000000+5;
struct Node
{
    Node *ch[2],*fa;
    int val,Max,add;
    bool rev;
    Node() {}
    inline void clear();
    inline void push();
    inline void pull();
}mem[maxnode],*EMPTY=mem;
inline void Node::clear() {val=Max=add=rev=0;ch[0]=ch[1]=fa=EMPTY;}
inline void Node::push()
{
    val+=add;
    ch[0]->add+=add;ch[1]->add+=add;
    add=0;
    if(rev)
    {
        ch[1]->rev^=1;ch[0]->rev^=1;
        swap(ch[0],ch[1]);
        rev=0;
    }
}
inline void Node::pull()
{
    Max=max(val,max(ch[0]==EMPTY?0:(ch[0]->Max+ch[0]->add), ch[1]==EMPTY?0:(ch[1]->Max+ch[1]->add)));
}
inline bool isroot(Node* now) {return now->fa->ch[0]!=now && now->fa->ch[1]!=now;}
inline void rotate(Node* now,int d)
{
    Node* fa=now->fa,*gfa=fa->fa;
    fa->ch[d^1]=now->ch[d];
    if(now->ch[d]!=EMPTY) now->ch[d]->fa=fa;
    fa->fa=now;
    now->ch[d]=fa;

    now->fa=gfa;
    if(gfa->ch[0]==fa) gfa->ch[0]=now;
    else if(gfa->ch[1]==fa) gfa->ch[1]=now;
    fa->pull();now->pull();gfa->pull();
}
inline void splay(Node* now)
{
    now->push();
    for(Node *fa,*gfa;!isroot(now) && (fa=now->fa)!=EMPTY;)
        if(!isroot(fa) && (gfa = fa->fa)!=EMPTY)
        {
            gfa->push();fa->push();now->push();
            int d = (gfa->ch[0]==fa);
            if(fa->ch[d^1]==now) rotate(fa,d),rotate(now,d); // collinear
            else rotate(now,d^1),rotate(now,d);
        }
        else
        {
            fa->push();now->push();
            rotate(now,fa->ch[0]==now);
            break;
        }
    now->pull();
}
inline Node* access(Node* now)
{
    Node* last=EMPTY;
    for(;now!=EMPTY;now=now->fa)
    {
```

```
        splay(now);
        now->ch[1]=last;
        last->fa=now;
        (last=now)->pull();
    }
    return last;
}
inline Node* find_root(Node* now)
{
    for(now=access(now);now->push(),now->ch[0]!=EMPTY;now=now->ch[0]);
    return now;
}
inline void make_root(Node* now)
{
    access(now)->rev^=1;
    splay(now);
}
inline void link(Node* x,Node* y)
{
    make_root(y);
    y->fa=x;
    access(y);
}
inline void cut(Node* x,Node* y)
{
    make_root(x);
    access(y);splay(y);
    y->ch[0]->fa=EMPTY;
    y->ch[0]=EMPTY;
    y->pull();
}
inline void modify(Node* x,Node* y,int w)
{
    make_root(x);
    access(y);splay(y);
    y->add+=w;
}
inline int query(Node* x,Node* y)
{
    make_root(x);
    access(y);splay(y);
    return y->Max+y->add;
}
```

4.6 Graph Theorem/MCMF.cpp

```
struct Edge
{
  int from,to,cap,flow,cost;
};
struct MCMF
{
  int n,s,t;
  vector<Edge> edges;
  vector<int> G[maxn];
  int d[maxn],p[maxn],a[maxn];
  bool inq[maxn];
  void init(int n)
  {
    this->n=n;
    for(int i=0;i<n;i++) G[i].clear();
    edges.clear();
  }
  void AddEdge(int from,int to,int cap,int cost)
  {
    edges.push_back((Edge){from,to,cap,0,cost});
    edges.push_back((Edge){to,from,0,0,-cost});
    int m=edges.size();
    G[from].push_back(m-2);
    G[to].push_back(m-1);
  }
  bool bellmanford(int& flow,LL& cost)
  {
    for(int i=0;i<n;i++) d[i]=INF;
    memset(inq,0,sizeof(inq));
    d[s]=0;inq[s]=1;p[s]=0;a[s]=INF;

    queue<int> Q;
    Q.push(s);
    while(!Q.empty())
```

```
    {
      int now=Q.front();Q.pop();
      inq[now]=0;
      for(int i=0;i<G[now].size();i++)
      {
        Edge& e=edges[G[now][i]];
        if(e.cap>e.flow && d[e.to]>d[now]+e.cost)
        {
          d[e.to]=d[now]+e.cost;
          p[e.to]=G[now][i];
          a[e.to]=min(a[now],e.cap-e.flow);
          if(!inq[e.to]) {Q.push(e.to);inq[e.to]=1;}
        }
      }
    }
    if(d[t]==INF) return false;
    flow+=a[t];
    cost+=(LL)d[t]*(LL)a[t];
    int now=t;
    while(now!=s)
    {
      edges[p[now]].flow+=a[t];
      edges[p[now]^1].flow-=a[t];
      now=edges[p[now]].from;
    }
    return 1;
  }
  int Mincost(int s,int t,LL& cost)
  {
    int flow=0;
    this->s=s,this->t=t;
    while(bellmanford(flow,cost));
    return flow;
  }
};
```

4.7 Graph Theorem/stable marriage problem - Gale-Shapley.cpp

```
//Propose-and-reject algorithm
int pref[maxn][maxn],order[maxn][maxn],next[maxn];
int future_husband[maxn], future_wife[maxn];
queue<int> q;

void engage(int man,int woman)
{
  int m=future_husband[woman];
  if(m)
  {
    future_wife[m]=0;
      q.push(m);
  }
  future_wife[man] = woman;
  future_husband[woman] = man;
}
int main()
{
  int T;
  scanf("%d",&T);
  while(T--)
  {
    int n;
    scanf("%d",&n);

    for(int i=1;i<=n;i++)
    {
      for(int j=1;j<=n;j++)
        scanf("%d",&pref[i][j]);
      next[i]=1;
      future_wife[i]=0;
      q.push(i);
    }

    for(int i=1;i<=n;i++)
    {
      for(int j=1;j<=n;j++)
      {
        int x;
        scanf("%d",&x);
        order[i][x]=j;
      }
```

```
      future_husband[i]=0;
    }

    while(!q.empty())
    {
      int man=q.front();q.pop();
      int woman = pref[man][next[man]++];
      if(!future_husband[woman])
        engage(man,woman);
      else if(order[woman][man] < order[woman][future_husband[woman]])
        engage(man,woman);
      else q.push(man);
    }
    while(!q.empty()) q.pop();

    for(int i=1;i<=n;i++) printf("%d\n",future_wife[i]);
    if(T) printf("\n");
  }
  return 0;
}
```

4.8 Graph Theorem/Strongly Connected Component - Kosaraju.cpp

```
/// to make sure every dfs gets one SCC
/// we want to traversal by the topological order of SCC
vector<int> G[maxn],G2[maxn];/// G2 is G1's transpose
vector<int> S;
int vis[maxn],sccno[maxn],scc_cnt;

void dfs1(int u)
{
  if(vis[u]) return;
  vis[u]=1;
  for(int i=0;i<G[u].size();i++) dfs(G[u][i]);
  S.push_back(u);
}

void dfs2(int u)
{
  if(sccno[u]) return;
  sccno[u] = scc_cnt;
  for(int i=0;i<G2[u].size();i++) dfs2(G2[u][i]);
}

void find_scc(int n)
{
  scc_cnt=0;
  S.clear();
  memset(sccno,0,sizeof(sccno));
  memset(vis,0,sizeof(vis));
  for(int i=0;i<n;i++) dfs1(i);
  for(int i=n-1;i>=0;i--)
    if(!sccno[S[i]]) {scc_cnt++;dfs2(S[i]);}
}
```

4.9 Graph Theorem/Strongly Connected Component - Tarjan.cpp

```
int dfs_clock,scc_cnt,pre[maxn],sccno[maxn],lowlink[maxn];
stack<int> S;
void dfs(int now)
{
  lowlink[now]=pre[now]=++dfs_clock;
  S.push(now);
  for(int i=0;i<G[now].size();i++)
  {
    int to=G[now][i];
    if(!pre[to])
    {
      dfs(to);
      lowlink[now]=min(lowlink[to],lowlink[now]);
    }
    else if(!sccno[to]) lowlink[now]=min(lowlink[now],pre[to]);
  }

  if(lowlink[now]==pre[now])
  {
    scc_cnt++;
    while(1)
```

```
    {
      int x=S.top();S.pop();
      sccno[x]=scc_cnt;
      if(x==now) break;
    }
  }
}
void find_scc()
{
  memset(pre,0,sizeof(pre));
  memset(sccno,0,sizeof(sccno));
  memset(lowlink,0,sizeof(lowlink));
  scc_cnt=dfs_clock=0;
  for(int i=0;i<n;i++) if(!pre[i]) dfs(i);
}
```

5 Math

5.1 Math/FFT.cpp

```
struct FFT
{
    complex<DB> epsilon[maxn],buffer[maxn],temp[maxn];
    void fft(int n,int offset,int step)
    {
      if(n == 1) return;
      int m=n>>1;

        fft(m,offset,step<<1);
      fft(m,offset+step,step<<1);

      for(int k=0;k<m;k++)
        {
        int pos = 2 * step * k;
        temp[k] = buffer[pos + offset] + epsilon[k * step] * buffer[pos + offset + step];
        temp[k + m] = buffer[pos + offset] - epsilon[k * step] * buffer[pos + offset + step];
      }
        for(int i=0;i<n;i++) buffer[i * step + offset] = temp[i];
    }
    inline void init_epsilon(const int& n)
    {
      for(int i=0;i<n;i++) epsilon[i] = complex<DB>(cos(2.0 * pi * i / n), sin(2.0 * pi * i / n));
    }
    inline void dft(DB* coef,const int& n,complex<DB>* pv)
    {
        for(int i=0;i<n;i++) buffer[i]=complex<DB>(coef[i],0);
        fft(n,0,1);
        for(int i=0;i<n;i++) pv[i]=buffer[i];
    }
    inline void idft(complex<DB>* pv,const int& n,DB* coef)
    {
        for(int i=0;i<n;i++) buffer[i]=pv[i];
        fft(n,0,1);
        coef[0]=real(buffer[0])/n;
        for(int i=1;i<n;i++) coef[i]=real(buffer[n-i])/n;
    }
};
```

5.2 Math/Gauss elimination.cpp

```
typedef double Matrix[maxn][maxn];
void gauss_elimination(Matrix A,int n)
{
  int i,j,k,r;
  for(i=0;i<n;i++)
  {
    r=i;
    for(j=i+1;j<n;j++)
      if(fabs(A[j][i])>fabs(A[r][i])) r=j;
    if(r!=i) for(j=0;j<=n;j++) swap(A[r][j],A[i][j]);

    #define High_precision
    #ifndef High_precision
    for(k=i+1;k<n;k++)
    {
      DB f=A[k][i]/A[i][i];
```

```
      for(j=i;j<=n;j++) A[k][j]-=f*A[i][j];
    }
    #else
    for(j=n;j>=i;j--)
      for(k=i+1;k<n;k++)
        A[k][j]-=A[k][i]/A[i][i] * A[i][j];
    #endif
  }

  for(i=n-1;i>=0;i--)
  {
    for(j=i+1;j<n;j++) A[i][n]-=A[j][n]*A[i][j];
    A[i][n]/=A[i][j];
  }
}
```

5.3 Math/Miller_Rabin.cpp

```
typedef long long LL;
inline LL mul(LL a,LL b,const LL& n)
{
    LL res=0;a%=n,b%=n;
    for(;b;b>>=1)
    {
        if(b&1) res = (res+a>=n?res+a-n:res+a);
        a = (a+a>=n? a+a-n : a+a );
    }
    return res;
}
inline LL fast_pow(LL a,LL x,const LL& n)
{
    LL res=1;
    for(;x;x>>=1,a=mul(a,a,n)) if(x&1) res=mul(res,a,n);
    return res;
}
inline bool miller_rabin(const LL& n,const LL& a)
{
    if(__gcd(a,n)==n) return 1;
    if(__gcd(a,n)!=1) return 0;
    LL u=n-1,t=0;
    while(!(u&1)) u>>=1,t++;

    LL x=fast_pow(a,u,n);
    if(x==1 || x==n-1) return 1;
    for(int i=0;i<t;i++)
    {
        x=mul(x,x,n);
        if(x==1) return 0;
        if(x==n-1) return 1;
    }
    return 0;
}
inline bool isprime(const LL& n)
{
    const static vector<LL> as={2,325,9375,28178,450775,9780504,1795265022};
    for(const LL& a:as) if(!miller_rabin(n,a)) return 0;
    return 1;
}
```

5.4 Math/NTT.cpp

```
class NTT
{
    private:
    LL epsilon[maxn],buffer[maxn],temp[maxn];
    static const LL P=(7LL<<50)+1;
    static const LL G=6;
    inline LL add(const LL& a,const LL b)
    {
        if(a+b>P) return a+b-P;
        else if(a+b<0) return a+b+P;
        return a+b;
    }
    inline LL mul(const LL& a,const LL b)
    {
        LL y=(LL)((double)a*b/P+0.5);
        LL r=(a*b-y*P)%P;
        return r<0?r+P:r;
```

```
    }
    LL fast_pow(LL x,LL a)
    {
        LL res=1;
        for(;a;a>>=1,x=mul(x,x)) if(a&1) res=mul(res,x);
        return res;
    }
    inline void init(LL n)
    {
        epsilon[0]=1;
        epsilon[1]=fast_pow(G,(P-1)/n);
        for(LL i=2;i<n;i++) epsilon[i]= mul(epsilon[1],epsilon[i-1]);
    }
    void fft(LL n,LL offset,LL step)
    {
        for(LL i=0,j=0;i<n;i++)
        {
            if(i>j) swap(buffer[i],buffer[j]);
            for(LL k=n>>1;(j^=k)<k;k>>=1);
        }
        for(LL i=2,m=1;i<=n;i<<=1,m<<=1)
            for(LL offset=0;offset<n;offset+=i)
                for(LL k=0;k<m;k++)
                {
                    LL rgt= mul(epsilon[n / i * k ] , buffer[offset + m + k ]);
                    buffer[offset + m + k] = add(buffer[offset + k], -rgt);
                    buffer[offset + k] = add(buffer[offset + k] , rgt);
                }
    }

    public:
    void dft(const vector<LL>& coef,LL n,vector<LL>& pv)
    {
        for(LL i=0;i<n;i++) buffer[i]= (i<(int)coef.size()?coef[i]:0);
        fft(n,0,1);

        pv.resize(n);
        for(LL i=0;i<n;i++) pv[i]=buffer[i];
    }
    void idft(const vector<LL>& pv,LL n,vector<LL>& coef)
    {
        for(LL i=0;i<n;i++) buffer[i]= (i<(int)pv.size()?pv[i]:0);
        fft(n,0,1);

        coef.resize(n);
        LL inv=fast_pow(n,P-2);
        coef[0]=mul(buffer[0],inv);
        for(LL i=1;i<n;i++) coef[i]=mul(buffer[n-i],inv);
    }
    void conv(const vector<LL>& fx,const vector<LL>& gx,vector<LL>& hx)
    {
        LL n;
        for(n=1;n<(int)fx.size()+(int)gx.size();n<<=1);


        vector<LL> pv[3];
        init(n);
        dft(fx,n,pv[0]);
        dft(gx,n,pv[1]);

        pv[2].resize(n);
        for(LL i=0;i<n;i++) pv[2][i]=mul(pv[0][i],pv[1][i]);

        idft(pv[2],n,hx);
    }
}ntt;
```

6 Others

6.1 Others/___builtin__.cpp

```
int __builtin_ffs (unsigned int x)
Returns one plus the index of the least significant 1-bit of x, or if x is zero, returns zero.

int __builtin_clz (unsigned int x)
Returns the number of leading 0-bits in x, starting at the most significant bit position. If x is 0, the result
    is undefined.
```

```
int __builtin_ctz (unsigned int x)
Returns the number of trailing 0-bits in x, starting at the least significant bit position. If x is 0, the result
     is undefined.

int __builtin_popcount (unsigned int x)
Returns the number of 1-bits in x.

int __builtin_parity (unsigned int x)
Returns the parity of x, i.e. the number of 1-bits in x modulo 2.

int __builtin_ffsll (unsigned long long)
Similar to __builtin_ffs, except the argument type is unsigned long long.

int __builtin_clzll (unsigned long long)
Similar to __builtin_clz, except the argument type is unsigned long long.

int __builtin_ctzll (unsigned long long)
Similar to __builtin_ctz, except the argument type is unsigned long long.

int __builtin_popcountll (unsigned long long)
Similar to __builtin_popcount, except the argument type is unsigned long long.

int __builtin_parityll (unsigned long long)
Similar to __builtin_parity, except the argument type is unsigned long long.
```

6.2 Others/bitwise operations - collection.cpp

```cpp
// https://blog.kuoe0.tw/posts/2012/01/28/bitwise-operation-set-operation/

// 列舉集合 S 的所有子集合
int temp = S;
do {
    // proccess
    temp = (temp - 1) & S;
} while (temp != S);

// 列舉有 n 個元素的宇集合 U 中所有大小🄵 k 的子集合
int temp = (1 << k) - 1;
while (temp < (1 << n)) {
    // proccess
    int last_1 = temp & -temp;
    int carry = temp + last_1;
    int cont_bits = temp & (~carry);
    int trail = (cont_bit / last_1) >> 1;
    temp = carry | trail;
}
```

6.3 Others/Dancing Links.cpp

```cpp
struct DLX
{
    int ansd,ans[maxr],n,sz,S[maxc];
    int row[maxnode],col[maxnode];
    int L[maxnode],R[maxnode],U[maxnode],D[maxnode];
    void init(int n,int m)
    {
        this->n=n;

        for(int i=1;i<=m;i++) column[i].clear();
        for(int i=0;i<=n;i++) U[i]=i,D[i]=i,L[i]=i-1,R[i]=i+1;
        L[0]=n,R[n]=0;
        sz=n+1;
        for(int i=0;i<=n;i++) S[i]=0;
    }
    void addRow(int r,vector<int>& column)
    {
        int first=sz;
        for(int i=0;i<column.size();i++)
        {
            int c=column[i];
            L[sz]=sz-1,R[sz]=sz+1;
            D[sz]=c,U[sz]=U[c];
            D[U[c]]=sz,U[c]=sz;
            row[sz]=r,col[sz]=c;
            S[c]++;sz++;
        }
        L[first]=sz-1,R[sz-1]=first;
    }
```

```
    #define FOR(i,A,st) for(int i=A[st];i!=st;i=A[i])
    void remove(int c)
    {
        L[R[c]]=L[c];
        R[L[c]]=R[c];
        FOR(i,D,c) FOR(j,R,i) {U[D[j]]=U[j];D[U[j]]=D[j];--S[col[j]];}
    }
    void restore(int c)
    {
        FOR(i,U,c) FOR(j,L,i) {U[D[j]]=j;D[U[j]]=j;++S[col[j]];}
        L[R[c]]=c;
        R[L[c]]=c;
    }
    bool dfs(int d)
    {
        if(R[0]==0)
        {
            ansd=d;
            return 1;
        }
        int c=R[0];
        FOR(i,R,0) if(S[i]<S[c]) c=i;

        remove(c);
        FOR(i,D,c)
        {
            ans[d]=row[i];
            FOR(j,R,i) remove(col[j]);
            if(dfs(d+1)) return 1;
            FOR(j,L,i) restore(col[j]);
        }
        restore(c);
  return 0;
    }
    bool solve()
    {
  if(!dfs(0)) return 0;
  return 1;
    }
}solver;
```

6.4 Others/gcc_optimise.cpp

```
#pragma GCC optimize("O3")
```

7 String

7.1 String/Aho-Corasick automaton.cpp

```
const int SIGMA_SIZE=26;
int fail[1000010],last[1000010];
int ch[1000010][26];
int val[1000010];
int idx(char c){return c-'a';}
void getfail()
{
  queue<int> q;
  fail[0]=0;
  for(int c=0;c<SIGMA_SIZE;c++)
  {
    int u=ch[0][c];
    if(u) {fail[u]=0;q.push(u);last[u]=0;}
  }
  while(!q.empty())
  {
    int r=q.front();q.pop();
    for(int c=0;c<SIGMA_SIZE;c++)
    {
      int u=ch[r][c];
      if(!u) continue;
      q.push(u);
      int v=fail[r];
      while(v && !ch[v][c]) v=fail[v];
      fail[u]= ch[v][c];
      last[u]=val[fail[u]]? fail[u]:last[fail[u]];
    }
```

```
  }
}
void find(char* T)
{
  int n=strlen(T);
  int j=0;
  for(int i=0;i<n;i++)
  {
    int c=idx(T[i]);
    while(j && !ch[j][c]) j=fail[j];
    j = ch[j][c];
    if(val[j]) print(j);
    else if(last[j]) print(last[j]);
  }
}
```

7.2 String/KMP.cpp

```
void getfail(char* P,int* f)
{
  int m=strlen(P);
  f[0]=0,f[1]=0;
  for(int i=1;i<m;i++)
  {
    int j=f[i];
    while(j&&P[i]!=P[j]) j=f[j];
    f[i+1]= (P[i]==P[j]?j+1:0);
  }
}
void find(char* T,char* P,int* f)
{
  int n=strlen(T),m=strlen(P);
  getfail(P,f);
  int j=0;
  for(int i=0;i<n;i++)
  {
    while(j&&P[j]!=T[i]) j=f[j];
    if(P[j]==T[i]) j++;
    if(j==m) printf("%d\n",i-m+1);
  }
}
```

7.3 String/Suffix Array.cpp

```
struct SuffixArray
{
  int maxn=1000000;
  char s[maxn];
  int sa[maxn],rank[maxn],pri[maxn],c[maxn],n;
  void build_sa(int m)
  {
    n=strlen(s);s[n++]=0;
    memset(c,0,sizeof(c));
    for(int i=0;i<n;i++) c[rank[i]=s[i]]++;
    for(int i=1;i<m;i++) c[i]+=c[i-1];
    for(int i=n-1;i>=0;i--) sa[--c[rank[i]]]=i;

    for(int k=1;k<=n;k<<=1)
    {
      int p=0;
      for(int i=n-k;i<n;i++) pri[p++]=i;
      for(int i=0;i<n;i++) if(sa[i]>=k) pri[p++]=sa[i]-k;

      memset(c,0,sizeof(c));
      for(int i=0;i<n;i++) c[rank[pri[i]]]++;
      for(int i=1;i<m;i++) c[i]+=c[i-1];
      for(int i=n-1;i>=0;i--) sa[--c[rank[pri[i]]]]=pri[i];

      swap(pri,rank);
      p=1;rank[sa[0]]=0;
      for(int i=1;i<n;i++)
        rank[sa[i]] = pri[sa[i-1]]==pri[sa[i]] && pri[sa[i-1]+k]==pri[sa[i]+k]?p-1:p++;
      if(p>=n) break;
      m=p;
    }
  }
  int m;
  int cmp_suffix(char* pattern,int p)
```

```
  {
    return strncmp(pattern,s+sa[p],m);
  }
  int find(char* p)
  {
    m=strlen(p);
    if(cmp_suffix(p,0)<0) return -1;
    if(cmp_suffix(p,n-1)>0) return -1;
    int l=0,r=n-1;
    while(r>=l)
    {
      int mid=l+(r-l)/2;
      int res=cmp_suffix(p,mid);
      if(!res) return mid;
      if(res<0) r=mid-1;
      else l=mid+1;
    }
    return -1;
  }

  int height[maxn];
  void getheight()
  {
    int k=0;
    for(int i=1;i<n;i++) rank[sa[i]]=i;
    for(int i=0;i<n-1;i++)
    {
      if(k) k--;
      int j=sa[rank[i]-1];
      while(s[i+k]==s[j+k]) k++;
      height[rank[i]]=k;
    }
  }
}SA;
```

7.4 String/Suffix Automaton.cpp

```
const int maxnode=1000000+5;
const int SIGMA_SIZE=26;
struct Node
{
    Node *fail,*ch[SIGMA_SIZE];
    int Max;
    Node() {Max=0,fail=0;memset(ch,0,sizeof ch);}
} node[maxnode],*root,*last;
struct SuffixAutomaton
{
    int size;
    inline int idx(char c) {return c-'a';}
    inline void init() {size=0;last=root=&node[0];}
    inline void add(char c)
    {
        c=idx(c);
        Node *p=last;
        Node *np=&node[++size]; np->Max=p->Max+1;
        while(p && !p->ch[c]) p->ch[c]=np,p=p->fail;
        if(!p) np->fail=root;
        else
        {
            Node *q=p->ch[c];
            if(q->Max==p->Max+1) np->fail=q;
            else
            {
                Node* nq=&node[++size];nq->Max=p->Max+1;
                memcpy(nq->ch,q->ch,sizeof(q->ch));
                nq->fail=q->fail;
                q->fail=np->fail=nq;
                while(p && p->ch[c]==q) p->ch[c]=nq,p=p->fail;
            }
        }
        last=np;
    }
}SAM;
```

7.5 String/Trie.cpp

```
#include<bits/stdc++.h>
using namespace std;
```

```
const int maxnode=100000;
const int sigma_size=26;
struct Trie
{
  int ch[maxnode][sigma_size];
  int val[maxnode];
  int sz; // number of nodes
  Trie() {sz=1;memset(ch[0],0,sizeof(ch[0]));}// only root
  int idx(char c){return c-'a';}

  // insert string s , information is v. v is not zero.
  void insert(char *s,int v)
  {
    int u=0,n=strlen(s);
    for(int i=0;i<n;i++)
    {
      int c=idx(s[i]);
      if(!ch[u][c])
      {
        memset(ch[sz],0,sizeof(ch[sz]));
        val[sz]=0; // intermediate nodes' information is zero(not exist)
        ch[u][c]=sz++; // new node
      }
      u = ch[u][c]; // keep going!!
    }
    val[u]=v; // val assigned to the last character
  }
};
int main()
{

}
```

7.6 String/Z algorithm.cpp

```
char s[maxs];
int Z[maxs];
void build_Z()
{
  int n=strlen(s);
  Z[0]=0;
  int L,R;
  L=R=0;
  for(int i=1;i<n;i++)
  {
    if(i>R) Z[i]=0;
    else
    {
      int ip=i-L;
      if(ip+Z[ip]<Z[L]) Z[i]=Z[ip];
      else Z[i]=R-i+1;
    }
    while(i+Z[i]<n && s[i+Z[i]]==s[Z[i]]) Z[i]++;

    if(i+Z[i]-1>R)
    {
      L=i;
      R=i+Z[i]-1;
    }
  }
}
```