爱吃吗/爱吃泡菜 Code Library

Dark History

目录

1 Data Structure Forever

1.1 Partition Tree

```
int val[19][100100] = {0};
int lsize[19][100100] = {0};
int sorted[100100] = {0}; // [1,N], sorted needed

// build_dt(1,N)
int build_dt(int l,int r,int depth=0)
{
  if(l == r) return 0;
  int mid = (l+r)/2;
  int x = sorted[mid];
  int samecnt = mid-l+1;
  for(int i = l;i <= mid;i++) if(sorted[i] < x) samecnt--;

  int pl = l;
  int pr = mid+1;
  for(int i = l;i <= r;i++)
  {
    lsize[depth][i] = lsize[depth][i-1];
    if(val[depth][i] < x || (val[depth][i] == x && samecnt))
    {
      if(val[depth][i] == x) samecnt--;
      val[depth+1][pl++] = val[depth][i];
      lsize[depth][i]++;
    }
    else val[depth+1][pr++] = val[depth][i];
  }
  build_dt(l,mid,depth+1);
  build_dt(mid+1,r,depth+1);
  return 0;
}

// query_kth(1,N,l,r,k)
int query_kth(int L,int R,int l,int r,int k,int depth=0)
{
  if(l == r) return val[depth][l];
  int mid = (L+R)/2;
  int lc = lsize[depth][l-1] - lsize[depth][L-1];
  int rc = lsize[depth][r] - lsize[depth][L-1];
  int lr = l-L-lc;
  int rr = r-L-rc+1;
  if(rc - lc >= k) return query_kth(L,mid,L+lc,L+rc-1,k,depth+1);
  return query_kth(mid+1,R,mid+1+lr,mid+rr,k-(rc-lc),depth+1);
}
```

1.2 Splay

如果需要建初始树，记得 x->update()

```
class SNode
{
public:
  int val;
  int size;
  bool rev;

  SNode* child[2];
  SNode* fa;

  int update()
  {
    pushdown();
    size = 1;
    for(int i = 0;i < 2;i++)
      if(child[i])
      {
        child[i]->pushdown();
        size += child[i]->size;
      }
    return 0;
  }
  int pushdown()
  {
    if(rev)
    {
      swap(child[0],child[1]);
      for(int i = 0;i < 2;i++)
        if(child[i]) child[i]->rev ^= 1;
      rev = false;
    }
    return 0;
  }
};

int Rotate(SNode* x,int dir)
{
  SNode* p = x->fa;
  p->pushdown();
  x->pushdown();

  p->child[dir] = x->child[dir^1];
  if(x->child[dir^1]) x->child[dir^1]->fa = p;
  x->child[dir^1] = p;

  x->fa = p->fa;
```

```
  if(!p->fa) Root = x;
  else if(p->fa->child[0] == p) p->fa->child[0] = x;
  else p->fa->child[1] = x;
  p->fa = x;
  p->update(); x->update();
  return 0;
}

SNode* Splay(SNode* x,SNode* Tar)
{
  while(x->fa != Tar)
  {
    int dir = 0;
    if(x->fa->child[0] == x) dir = 0;
    else dir = 1;
    if(x->fa->fa == Tar) Rotate(x,dir);
    else if(x->fa->fa->child[dir] == x->fa)
    {
      Rotate(x->fa,dir);
      Rotate(x,dir);
    } else {
      Rotate(x,dir);
      Rotate(x,dir^1);
    }
  }
  return x;
}

SNode* Select(SNode* x,int k)
{
  while(1)
  {
    x->pushdown();
    int xrank = 1;
    if(x->child[0]) xrank += x->child[0]->size;
    if(xrank == k) break;
    else if(k < xrank) x = x->child[0];
    else
    {
      x = x->child[1];
      k -= xrank;
    }
  }
  return x;
}
```

1.3 BIT Kth

```
int Kth(int k)
{
  int cnt = 0;
  int ans = 0;
  for(int p = (1<<logcnt);p > 0;p >>= 1)
  {
    ans += p;
    if(ans > scorecnt || cnt+BIT[ans] >= k) ans -= p;
    else cnt += BIT[ans];
  }
  return ans+1-1;
}
```

1.4 KD Tree

如果被卡可以考虑写上 minx,maxx,miny,maxy 维护矩形，修改 KDTree_Build 加上对应的维护。

```
struct POINT { int x,y,id; };
inline bool cmp_x(const POINT& a,const POINT& b) { return a.x == b.x ? a.y <
  b.y : a.x < b.x; }
inline bool cmp_y(const POINT& a,const POINT& b) { return a.y == b.y ? a.x <
  b.x : a.y < b.y; }

struct KDNODE
{
  POINT p;
//  int minx,maxx,miny,maxy;

  KDNODE* Child[2];
  KDNODE* fa;
};
KDNODE NPool[111111];
KDNODE* NPTop = NPool;
KDNODE* Root;

inline KDNODE* AllocNode()
{
  memset(NPTop,0,sizeof(KDNODE));
  return NPTop++;
}

inline ll PDist(const POINT& a,const POINT& b) { return sqr((ll)(a.x-b.x))+
  sqr((ll)(a.y-b.y)); }

POINT pnt[111111];

KDNODE* KDTree_Build(int l,int r,int depth=0)
{
```

```
  if(l >= r) return NULL;

  if(depth&1) sort(pnt+l,pnt+r,cmp_y);
  else sort(pnt+l,pnt+r,cmp_x);

  int mid = (l+r)/2;
  KDNODE* t = AllocNode();

  t->Child[0] = KDTree_Build(l,mid,depth+1);
  t->Child[1] = KDTree_Build(mid+1,r,depth+1);
  for(int i = 0;i < 2;i++)
    if(t->Child[i]) t->Child[i]->fa = t;

  return t;
}

int KDTree_Insert(KDNODE* cur,POINT& P,int depth=0)
{
  KDNODE* node = AllocNode(); node->p = P;
  while(cur)
  {
    if(cur->p.x == P.x && cur->p.y == P.y && cur->p.id == P.id) break;
    int dir = 0;
    if(depth&1) dir = cmp_y(x->p,P);
    else dir = cmp_x(x->p,P);
    if(!cur->Child[dir])
    {
      cur->Child[dir] = node;
      node->fa = cur;
      break;
    }
    else
    {
      cur = cur->Child[dir];
      depth++;
    }
  }
  return 0;
}

ll KDTree_Nearest(KDNODE* x,const POINT& q,int depth=0)
{
  KDNODE* troot = x->fa;
  int dir = 0;
  while(x)
  {
    if(depth&1) dir = cmp_y(x->p,q);
    else dir = cmp_x(x->p,q);

    if(!x->Child[dir]) break;
    x = x->Child[dir];
    depth++;
  }
  ll ans = ~0ULL>>1;
  while(x != troot)
  {
    ll tans = PDist(q,x->p);
    if(tans < ans) ans = tans;
    KDNODE* oside = x->Child[dir^1];
    if(oside)
    {
      ll ldis = 0;
      /*if(depth&1) ldis = min(sqr((ll)q.y-oside->miny),sqr((ll)q.y-oside->
        maxy));
      else ldis = min(sqr((ll)q.x-oside->minx),sqr((ll)q.x-oside->maxx));*/
      if(depth & 1) ldis = sqr<ll>(x->p.y-q.y);
      else ldis = sqr<ll>(x->p.x-q.x);
      if(ldis < ans)
      {
        tans = KDTree_Nearest(oside,q,depth+1);
        if(tans && tans < ans) ans = tans;
      }
    }

    if(x->fa && x == x->fa->Child[0]) dir = 0;
    else dir = 1;
    x = x->fa;
    depth--;
  }
  return ans;
}
```

1.5 Light-Heavy Decomposition

非递归版本，NodeID 为全局 ID。

```
int BlockRoot[111111];
int NodeID[111111];
int TreeSize[111111];
int Depth[111111];
int HeavyChild[111111]; // 0 if not set
int fa[111111];
int Queue[111111];
int idx = 0;
int Decomposition(int s)
{
  int qfront = 0;
  int qend = 0;
```

```
  Queue[qend++] = s;
  while(qfront < qend)
  {
    int x = Queue[qfront++];
    TreeSize[x] = 1;
    for(EDGE* e = E[x];e;e = e->Next)
    {
      int y = e->y;
      if(y == fa[x]) continue;

      fa[y] = x;
      Depth[y] = Depth[x]+1;
      Queue[qend++] = y;
    }
  }
  for(int i = qend-1;i >= 0;i--)
  {
    int x = Queue[i];
    for(EDGE* e = E[x];e;e = e->Next)
    {
      int y = e->y;
      if(y == fa[x]) continue;
      TreeSize[x] += TreeSize[y];
      if(TreeSize[HeavyChild[x]] < TreeSize[y]) HeavyChild[x] = y;
    }
  }

  for(int i = qend-1;i >= 0;i--)
  {
    int x = Queue[i];
    if(x == HeavyChild[fa[x]]) continue;
    int t = x;
    while(t)
    {
      BlockRoot[t] = x;
      NodeID[t] = ++idx;
      t = HeavyChild[t];
    }
  }
  return 0;
}

int ColorNode(int x,int y,int nc)
{
    while(1)
    {
        if(Depth[BlockRoot[x]] > Depth[BlockRoot[y]]) swap(x,y);

        if(BlockRoot[x] == BlockRoot[y])
        {
            if(Depth[x] > Depth[y]) swap(x,y);
            Seg_Modify(NodeID[x],NodeID[y],nc,1,idx);
            break;
        }
        Seg_Modify(NodeID[BlockRoot[y]],NodeID[y],nc,1,idx);
        y = fa[BlockRoot[y]];
    }
    return 0;
}
```

1.6 Merge-Split Treap (Incomplete)

```
struct TNODE
{
  int val;
  int rd;
  int size;

  TNODE* left;
  TNODE* right;

  inline int update()
  {
    size = 1;
    if(left) size += left->size;
    if(right) size += right->size;
    return 0;
  }
};

typedef pair<TNODE*,TNODE*> ptt;

stack<TNODE*> GCPool;
TNODE TPool[11111111];
TNODE* TPTop = TPool;

TNODE* newNode(int val,int rd,TNODE* left,TNODE* right)
{
  TNODE* result = NULL;
  if(GCPool.size()) { result = GCPool.top(); GCPool.pop(); }
  else result = TPTop++;
  result->val = val; result->rd = rd; result->left = left; result->right =
    right;
  result->update();
  return result;
}
```

```
TNODE* Merge(TNODE* t1,TNODE* t2)
{
  if(!t1) return t2;
  if(!t2) return t1;

  if(t1->rd <= t2->rd) return newNode(t1->val,t1->rd,t1->left,Merge(t1->right
    ,t2));
  else return newNode(t2->val,t2->rd,Merge(t1,t2->left),t2->right);
}

// split after pos nodes
ptt Split(TNODE* x,int pos)
{
  if(pos == 0) return ptt(NULL,x);
  if(pos == x->size) return ptt(x,NULL);

  int lsize = x->left ? x->left->size : 0;
  int rsize = x->right ? x->right->size : 0;
  if(lsize == pos) return ptt(x->left,x->right);
  if(pos < lsize)
  {
    ptt st = Split(x->left,pos);
    return ptt(st.first,newNode(x->val,x->rd,st.second,x->right));
  }
  else
  {
    ptt st = Split(x->right,pos-lsize-1);
    return ptt(newNode(x->val,x->rd,x->left,st.first),st.second);
  }
}
```

1.7 XHM_Splay

```
struct node {
  int f,ch[2],v,nl,nr,ans,s;
  node() {}
  void Init(int _v,int _f) {
    v = _v; f = _f; ch[0] = ch[1] = 0; s = abs(_v);
    nl = nr = 0; if (v > 0) nr = v; else nl = -v;
    ans = 0;
  }
}pt[MaxNode];

struct Splay {
  int root;
  void update(int t) {
    pt[t].s = pt[pt[t].ch[0]].s + pt[pt[t].ch[1]].s + abs(pt[t].v);
    pt[t].nr = max(0,pt[pt[t].ch[0]].nr + pt[t].v - pt[pt[t].ch[1]].nl) + pt[
      pt[t].ch[1]].nr;
    pt[t].nl = max(0,pt[pt[t].ch[1]].nl - pt[t].v - pt[pt[t].ch[0]].nr) + pt[
      pt[t].ch[0]].nl;
    if (pt[t].v > 0) { // node of boy
      pt[t].ans = pt[pt[t].ch[0]].ans + pt[pt[t].ch[1]].ans + min(pt[pt[t].ch
        [0]].nr + pt[t].v,pt[pt[t].ch[1]].nl);
    } else { // otherwise
      pt[t].ans = pt[pt[t].ch[0]].ans + pt[pt[t].ch[1]].ans + min(pt[pt[t].ch
        [0]].nr,pt[pt[t].ch[1]].nl - pt[t].v);
    }
  }
  void zig(int x,bool w) {
    int y = pt[x].f; if (root == y) root = x;
    pt[y].ch[!w] = pt[x].ch[w]; if (pt[x].ch[w]) pt[pt[x].ch[w]].f = y;
    pt[x].f = pt[y].f; if (root != x) pt[pt[y].f].ch[y == pt[pt[y].f].ch[1]]
      = x;
    pt[x].ch[w] = y; pt[y].f = x; update(y);
  }
  void splay(int x) {
    while (x != root) {
      if (pt[x].f == root) zig(x,x == pt[pt[x].f].ch[0]);
      else {
        int y = pt[x].f, z = pt[y].f;
        if (y == pt[z].ch[0]) if (x == pt[y].ch[0]) { zig(y,1); zig(x,1); }
          else { zig(x,0); zig(x,1); }
        else if (x == pt[y].ch[0]) { zig(x,1); zig(x,0); } else { zig(y,0);
          zig(x,0); }
      }
    } update(x);
  }
  void splay(int x,int f) {
    while (pt[x].f != f) {
      if (pt[pt[x].f].f == f) zig(x,x == pt[pt[x].f].ch[0]);
      else {
        int y = pt[x].f, z = pt[y].f;
        if (y == pt[z].ch[0]) if (x == pt[y].ch[0]) { zig(y,1); zig(x,1); }
          else { zig(x,0); zig(x,1); }
        else if (x == pt[y].ch[0]) { zig(x,1); zig(x,0); } else { zig(y,0);
          zig(x,0); }
      }
    } update(x);
  }
  int selFlag;
  int sel(int Key) {
    int t = root;
    while (1) {
      int ls = pt[pt[t].ch[0]].s;
      if (ls < Key && ls + abs(pt[t].v) >= Key) {
```

```
      selFlag = Key - ls;
      return t;
    }
    if (Key <= ls) t = pt[t].ch[0]; else {
      Key -= ls + abs(pt[t].v);
      t = pt[t].ch[1];
    }
  } return t;
}
void Del(int t) {
  while (pt[t].ch[0] + pt[t].ch[1]) if (pt[t].ch[0]) zig(pt[t].ch[0],1);
    else zig(pt[t].ch[1],0);
  if (root == t) {
    root = 0; return ;
  }
  pt[pt[t].f].ch[t == pt[pt[t].f].ch[1]] = 0; splay(pt[t].f);
}
int bound(int x,bool w) {
  splay(x);
  int ret = pt[x].ch[w];
  while (pt[ret].ch[!w]) ret = pt[ret].ch[!w];
  return ret;
}
PII Split(int t,int pos) {// break node t at postion pos
  int L = bound(t,0), R = bound(t,1); Del(t);
  splay(L,0); splay(R,L);
  int s = abs(pt[t].v); int c = (pt[t].v > 0) ? 1 : -1;
  if (pos >= 1) {
    pt[++now].Init(c * (pos),R); pt[R].ch[0] = now;
    splay(now); L = now; splay(R,L);
  }
  if (pos < abs(pt[t].v)) {
    pt[++now].Init(c * (abs(pt[t].v) - pos),R); pt[R].ch[0] = now;
    splay(now); R = now;
  }
  return MP(L,R);
}
}Tab;
```

2 Graph

2.1 Bridge

无向图求桥，支持重边。直接拆掉桥就是边 BCC。

```
int DFN[MAXN],Low[MAXN];
bool vis[MAXN],isBridge[MAXM];
int idx = 0;
int tarjan(int x,int peid=-1)
{
  vis[x] = true;
  DFN[x] = Low[x] = ++idx;
  for(EDGE* e = E[x];e;e = e->Next)
  {
    int y = e->y; int eid = e->id;
    if(eid == peid) continue;
    if(!vis[y])
    {
      tarjan(y,eid);
      Low[x] = min(Low[x],Low[y]);
    }
    else Low[x] = min(Low[x],DFN[y]);
  }
  if(peid != -1 && Low[x] == DFN[x]) isBridge[peid] = true;
  return 0;
}
```

2.2 Cut Point

求割点/点 BCC，同样支持重边。BCCId 为某条边在哪个 BCC 内。

```
int DFN[MAXN],Low[MAXN],Stack[MAXM],BCCId[MAXM];
bool vis[MAXN],isCP[MAXN];
int idx = 0,BCCidx = 0,STop = 0;
int tarjan(int x,int peid=-1)
{
  vis[x] = true;
  DFN[x] = Low[x] = ++idx;
  int ecnt = 0;
  for(EDGE* e = E[x];e;e = e->Next)
  {
    int y = e->y; int eid = e->id;
    if(eid == peid) continue;
    if(DFN[y] < DFN[x]) Stack[STop++] = eid;
    if(!vis[y])
    {
      tarjan(y,eid);
      Low[x] = min(Low[x],Low[y]);
      ecnt++;
      if(DFN[x] <= Low[y])
      {
        BCCidx++;
        while(Stack[--STop] != e->eid) BCCId[Stack[STop]] = BCCidx;
        BCCId[e->eid] = BCCidx;

        if(peid != -1) isCP[x] = true;
      }
```

```
    }
    else Low[x] = min(Low[x],DFN[y]);
  }
  if(peid == -1 && ecnt > 1) isCP[x] = true;
  return 0;
}
```

2.3 MMC (Karp)

$O(nm+n^2)$ 最大平均权值环需要存边但是不需要边表。

```
int d[677][677] = {0};
double Karp(int n,int m)
{
  memset(d,0,sizeof(d));

  // init all d[0][i] with 0 if no memset or reversing

  for(int i = 1;i <= n;i++)
    for(int j = 0;j < m;j++)
      if(d[i][E[j].y] < d[i-1][E[j].x]+E[j].k) d[i][E[j].y] = d[i-1][E[j].x]+
        E[j].k;

  double u = 0.0;
  for(int i = 0;i < n;i++)
  {
    double t = 1e100;
    for(int j = 0;j < n;j++)
    {
      if(d[j][i] >= 0)
      {
        double k = (double)(d[n][i]-d[j][i])/(n-j);
        if(k < t) t = k;
      }
    }
    if(t > u) u = t;
  }
  return u;
}
```

2.4 LCA (Tarjan)

$O(n)$ 仅在需要顺手维护点别的东西的时候用。

```
bool vis[40000] = {0};
int djs[40000] = {0};
int djs_find(int x) { return (djs[x] == x ? x : djs[x] = djs_find(djs[x])); }

int tarjan_lca(int root)
{
  djs[root] = root;
  vis[root] = true;

  for(QLINK* i = QLink[root];i != NULL;i = i->Next)
  {
    int qx = i->q->x;
    int qy = i->q->y;
    if(qx == root && vis[qy]) i->q->lca = djs_find(qy);
    if(qy == root && vis[qx]) i->q->lca = djs_find(qx);
  }

  for(EDGE* i = E[root];i != NULL;i = i->Next)
  {
    int y = i->y;
    if(y == fa[root]) continue;

    tarjan_lca(y);
  }
  djs[root] = fa[root];
  return 0;
}
```

2.5 LCA (sqr)

倍增 LCA $O(n\log n)$ 只要维护的是树，可以动态添加

```
int fa[111111][18];
int depth[111111];
int lca(int x,int y)
{
  if(depth[x] < depth[y]) swap(x,y);
  int delta = depth[x]-depth[y];
  for(int i = 0;i < 16;i++)
  {
    if(delta&(1<<i)) x = fa[x][i];
  }
  for(int i = 15;i >= 0;i--)
  {
    if(fa[x][i] != fa[y][i]) { x = fa[x][i]; y = fa[y][i]; }
  }
  if(x != y) x = fa[x][0];
  return x;
}
int Queue[111111];
int build_lca(int root)
{
  int front = 0;
```

```
  int end = 0;
  Queue[end++] = root;
  fa[root][0] = 0; // -1
  while(front != end)
  {
    int x = Queue[front++];
    for(EDGE* e = E[x];e;e = e->Next)
    {
      int y = e->y;
      fa[y][0] = x;
      depth[y] = depth[x]+1;
      Queue[end++] = y;
    }
  }
  for(int i = 1;i < 18;i++)
    for(int j = 0;j < end;j++)
    {
      int x = Queue[j];
      fa[x][i] = fa[fa[x][i-1]][i-1];
    }
  return 0;

}
```

2.6 Stable Marriage

求的是男性最优的稳定婚姻解。稳定即没有汉子更喜欢的妹子和妹子更喜欢的汉子两情相悦的情况。男性最优即不存在所有汉子都得到了他更喜欢的妹子的解。

orderM[i][j] 为汉子 i 第 j 喜欢的妹子，preferF[i][j] 为妹子 i 心中汉子 j 是第几位

不停的让汉子在自己的偏好列表里按顺序去找妹子，妹子取最优即可 $O(n^2)$

```
int stableMarriage(int n)
{
  memset(pairM,-1,sizeof(pairM));
  memset(pairF,-1,sizeof(pairF));
  int pos[MAXN] = {0};
  for(int i = 0;i < n;i++)
  {
    while(pairM[i] == -1) // or can be implemented using queue...
    {
      int wife = orderM[i][pos[i]++];
      int ex = pairF[wife];
      if(ex == -1 || preferF[wife][i] < preferF[wife][ex])
      {
        pairM[i] = wife;
        pairF[wife] = i;

        if(ex != -1)
        {
          pairM[ex] = -1;
          i = ex; // take GREAT care
        }
      }
    }
  }
  return 0;
}
```

2.7 Arborescence

最小树形图，注意对 EPool 的需求是 $|V| \times |E|$ 的。不定根的情况，造一个虚拟根，MAXINT 连上所有的点，最后答案减去 MAXINT 。求有向森林的同上，插 0 边即可。可以支持负边权求最大。

```
bool arborescence(int n,int root,double& ans)
{
  ans = 0;
  while(1)
  {
    double minIn[MAXN] = {0};
    int prev[MAXN] = {0};
    fill(minIn,minIn+n,MAXW);
    for(int i = 0;i < n;i++)
    {
      for(EDGE* e = E[i];e;e = e->Next)
      {
        int y = e->y;
        if(e->w < minIn[y])
        {
          minIn[y] = e->w;
          prev[y] = i;
        }
      }
    }
    for(int i = 0;i < n;i++)
    {
      for(EDGE* e = E[i];e;e = e->Next)
      {
        int y = e->y;
        if(y == root) continue;
        e->w -= minIn[e->y];
      }

      if(i == root) continue;
      if(minIn[i] == MAXW) return false; // does not exist
      ans += minIn[i];
    }
    int SCC[MAXN] = {0};
```

```
    int vis[MAXN] = {0};
    prev[root] = root;
    int sccidx = 0; int vidx = 0;
    for(int i = 0;i < n;i++)
    {
      if(vis[i]) continue;
      int x = i; vidx++;
      while(!vis[x])
      {
        vis[x] = vidx;
        SCC[x] = sccidx++;
        x = prev[x];
      }
      if(vis[x] == vidx) // circle
      {
        int ori = x;
        sccidx = SCC[x]+1;
        do
        {
          SCC[x] = SCC[ori];
          x = prev[x];
        } while(x != ori);
      }
    }
    if(sccidx == n) break; // found
    // rebuild
    EDGE* TE[MAXN] = {0};
    for(int i = 0;i < n;i++)
    {
      for(EDGE* e = E[i];e;e = e->Next)
      {
        if(SCC[i] != SCC[e->y]) insert_edge(SCC[i],SCC[e->y],e->w,TE);
      }
    }
    memcpy(E,TE,sizeof(E));

    n = sccidx;
    root = SCC[root];
  }
  return true;
}
```

2.8 Stoer_Wagner

无向图全局最小割。调用前建立邻接矩阵 G，跑完后会破坏 G。可记录点集。$O(n^3)$

```
int Stoer_Wagner(int n)
{
  int mincut = 0x7FFFFFFF;
  int id[MAXN] = {0};
  int b[MAXN] = {0};
  for(int i = 0;i < n;i++) id[i] = i;
  for(;n > 1;n--)
  {
    memset(b,0,sizeof(b));
    for(int i = 0;i < n-1;i++)
    {
      int p = i+1;
      for(int j = i+1;j < n;j++)
      {
        b[id[j]] += G[id[i]][id[j]];
        if(b[id[p]] < b[id[j]]) p = j;
      }
      swap(id[i+1],id[p]);
    }
    if(b[id[n-1]] < mincut) {
      // ufs_union(st.first,st.second);
      mincut = b[id[n-1]];
      // st = pii(id[n-1],id[n-2]);
    }
    //else ufs_union(id[n-1],id[n-2]);
    for(int i = 0;i < n-2;i++)
    {
      G[id[i]][id[n-2]] += G[id[i]][id[n-1]];
      G[id[n-2]][id[i]] += G[id[n-1]][id[i]];
    }
  }
  return mincut;
}
```

2.9 MaxFlow (ISAP)

最大流，时间复杂度 $O(n^2m)$ 。多次使用记得初始化。

```
const int MAXM = 1000000;
const int MAXN = 25000;
const int INF = 0x7FFFFFFF;

struct ARC
{
  int y,c;
  ARC* Next,R;
};

ARC APool[MAXM*2];
ARC* APTop = APool;
ARC* Arc[MAXN];
```

```
int insert_arc(int x,int y,int c,int rc=0)
{
  ARC* fore = APTop++;
  fore->y = y; fore->c = c; fore->Next = Arc[x]; Arc[x] = fore;
  ARC* back = APTop++;
  back->y = x; back->c = rc; back->Next = Arc[y]; Arc[y] = back;

  fore->R = back; back->R = fore;
  return 0;
}

int dis[MAXN],pre[MAXN],gap[MAXN];
ARC* curArc[MAXN];
int init_distance_mark(int s,int t,int n)
{
  fill(dis,dis+MAXN,n);
  queue<int> q;
  q.push(t);
  dis[t] = 0;
  while(!q.empty())
  {
    int x = q.front(); q.pop();
    for(ARC* a = Arc[x];a;a = a->Next)
    {
      if(a->R->c <= 0) continue;
      if(dis[a->y] > dis[x]+1)
      {
        dis[a->y] = dis[x]+1;
        q.push(a->y);
      }
    }
  }
  memset(gap,0,sizeof(gap));
  for(int i = 0;i < n;i++) gap[dis[i]]++;
  return 0;
}
int max_flow(int s,int t,int n)
{
  memset(dis,0,sizeof(dis));
  memset(curArc,0,sizeof(curArc));
  // memset(gap,0,sizeof(gap));
  // gap[0] = n;
  init_distance_mark(s,t,n);

  int maxflow = 0;
  int x = s;
  while(dis[s] < n)
  {
    if(x == t)
    {
      int tflow = INF;
      while(x != s)
      {
        tflow = min(tflow,curArc[pre[x]]->c);
        x = pre[x];
      }
      x = t;
      while(x != s)
      {
        curArc[pre[x]]->c -= tflow;
        curArc[pre[x]]->R->c += tflow;
        x = pre[x];
      }
      maxflow += tflow;
      continue;
    }
    if(!curArc[x]) curArc[x] = Arc[x];
    ARC* ar = curArc[x];
    for(;ar;ar = ar->Next)
    {
      int y = ar->y;
      int c = ar->c;
      if(!c) continue;
      if(dis[y]+1 == dis[x]) break;
    }
    curArc[x] = ar;
    if(!ar)
    {
      int mindis = n+1; // relabel
      for(ARC* a = Arc[x];a;a = a->Next) if(a->c) mindis = min(mindis,dis[a->
        y]+1);
      gap[dis[x]]--;
      if(!gap[dis[x]]) break;
      gap[dis[x] = mindis]++;
      if(x != s) x = pre[x];
    }
    else
    {
      pre[ar->y] = x;
      x = ar->y;
    }
  }
  return maxflow;
}
```

2.10 LT Dominator Tree

有向图，redge 是反向边。最后附有用法说明，idom 是输出结果，即每个点的直接 dominator 点。全部标号 0 起始。复杂度是 $O(NlogN)$

```
int fa[MAXN],nodeName[MAXN],nodeID[MAXN]; // ID->Name || Name->ID || ID = dfs order(DFN)
bool vis[MAXN]; int ncnt = 0;
vector<int> edges[MAXN],redges[MAXN];
int dfs(int x)
{
  vis[x] = true;
  nodeID[x] = ncnt; nodeName[ncnt++] = x;
  for(vit it = edges[x].begin();it != edges[x].end();++it)
  {
    if(vis[*it]) continue;
    fa[*it] = x; dfs(*it);
  }
  return 0;
}
int semi[MAXN],idom[MAXN],ufs[MAXN];
int mnsemi[MAXN]; // maintained during ufs_merge
vector<int> bucket[MAXN];

// x -> y
int ufs_union(int x,int y) { ufs[x] = y; return 0; }
int ufs_internal_find(int x)
{
  if(ufs[ufs[x]] == ufs[x]) return 0;
  ufs_internal_find(ufs[x]);
  if(semi[mnsemi[ufs[x]]] < semi[mnsemi[x]]) mnsemi[x] = mnsemi[ufs[x]];
  ufs[x] = ufs[ufs[x]];
  return 0;
}
int ufs_find(int x)
{
  if(ufs[x] == x) return x;
  ufs_internal_find(x);
  return mnsemi[x];
}

int calc_dominator_tree(int n)
{
  for(int i = 0;i < n;i++) { semi[i] = i; mnsemi[i] = i; ufs[i] = i; }
  for(int x = n-1;x > 0;x--)
  {
    int tfa = nodeID[fa[nodeName[x]]];
    for(vit it = redges[nodeName[x]].begin();it != redges[nodeName[x]].end()
      ;++it)
    {
      if(!vis[*it]) continue;
      int fy = ufs_find(nodeID[*it]);
      if(semi[fy] < semi[x]) semi[x] = semi[fy];
    }
    bucket[semi[x]].push_back(x);
    ufs_union(x,tfa);

    for(vit it = bucket[tfa].begin();it != bucket[tfa].end();++it)
    {
      int fy = ufs_find(*it);
      idom[nodeName[*it]] = nodeName[semi[fy] < semi[*it] ? fy : tfa];
    }
    bucket[tfa].clear();
  }
  for(int x = 1;x < n;x++)
  {
    if(idom[nodeName[x]] != nodeName[semi[x]])
    {
      idom[nodeName[x]] = idom[idom[nodeName[x]]];
    }
  }
  idom[nodeName[0]] = -1;
  return 0;
}

memset(fa,-1,sizeof(fa[0])*(n+10));
memset(idom,-1,sizeof(idom[0])*(n+10));
memset(vis,0,sizeof(vis[0])*(n+10));
for(int i = 0;i < n;i++) bucket[i].clear();
ncnt = 0;
dfs(n-1);
calc_dominator_tree(ncnt);
```

2.11 K-short Loopless Path

k 短无环路径。邻接矩阵 G 存图，然后调用 yenLoopless 即可，s 是起点，t 终点，n 点数，k 是 k。

```
const int MAXN = 50;
const int INF = 0x3F3F3F3F;

class PATH
{
public:
  int node[MAXN];
  int nodecnt;
  int block[MAXN];
  int blockcnt;
  int len;
```

```
  int dev;

  PATH(int v = 0) { memset(this,0,sizeof(PATH)); node[nodecnt++] = v; }
  bool operator>(const PATH& p) const
  {
    if(len != p.len)
      return len > p.len;
    else
    {
      for(int i = p.nodecnt-1,j = nodecnt-1;i >= 0 && j >= 0;i--,j--)
      {
        if(p.node[i] != node[j]) return node[j] > p.node[i];
      }
      return nodecnt > p.nodecnt;
    }
    return false;
  }
};

int dis[MAXN];
int pre[MAXN];
int G[MAXN][MAXN];
bool vis[MAXN];

bool block[MAXN][MAXN];

// O(n^2)
int dijkstra(int n)
{
  for (int p = 0;p < n;p++)
  {
    int minV = -1;
    for (int i = 0; i < n; i++)
    {
      if (!vis[i] && (minV == -1 || dis[i] < dis[minV])) minV = i;
    }
    if (minV == -1) break;
    vis[minV] = true;

    for(int to = 0;to < n;to++)
    {
      if(!vis[to] && !block[minV][to])
      {
        int len = G[minV][to];
        if(dis[to] > dis[minV]+len || (dis[to] == dis[minV]+len && minV < pre
          [to]))
        {
          dis[to] = dis[minV]+len;
          pre[to] = minV;
        }
      }
    }
  }
  return 0;
}

PATH shortestPath(int v)
{
  PATH p(v);
  p.len = dis[v];
  for (v = pre[v];v != -1;v = pre[v]) p.node[p.nodecnt++] = v;
  reverse(p.node,p.node+p.nodecnt);
  return p;
}

int delSubpath(const PATH& p, int dev)
{
  int last = p.node[0];
  vis[last] = true;
  int v;
  for (int i = 1; dev != i; i++)
  {
    v = p.node[i];
    pre[v] = last;
    dis[v] = dis[last]+G[last][v];
    vis[v] = true;
    last = v;
  }
  vis[last] = false;
  return 0;
}

int initSingleSrc(int s)
{
  memset(dis,0x3F,sizeof(dis));
  memset(pre,-1,sizeof(pre));
  memset(vis,0,sizeof(vis));
  dis[s] = 0;
  return 0;
}

int yenLoopless(int s,int t,int n,int k)
{
  PATH result[201];
  int cnt = 0;

  priority_queue< PATH, vector<PATH>, greater<PATH> > candidate;
  memset(block,0,sizeof(block));
```

```
  initSingleSrc(s);
  dijkstra(n);
  if (dis[t] < INF)
  {
    PATH sh = shortestPath(t);
    sh.dev = 1;
    sh.block[sh.blockcnt++] = sh.node[sh.dev];
    candidate.push(sh);
  }
  while (cnt < k && !candidate.empty())
  {
    PATH p = candidate.top();
    candidate.pop();

    memset(block, 0, sizeof(block));
    int dev = p.dev;
    while (dev < p.nodecnt)
    {
      int last = p.node[dev-1];
      if (dev == p.dev)
      {
        for (int i = 0; i < p.blockcnt; i++)
        {
          block[last][p.block[i]] = true;
        }
      }
      else block[last][p.node[dev]] = true;

      initSingleSrc(s);
      delSubpath(p,dev);
      dijkstra(n);

      if (dis[t] < INF)
      {
        PATH newP = shortestPath(t);
        newP.dev = dev;
        if (dev == p.dev)
        {
          newP.blockcnt = p.blockcnt;
          memcpy(newP.block,p.block,sizeof(newP.block));
        }
        else newP.block[newP.blockcnt++] = p.node[dev];
        newP.block[newP.blockcnt++] = newP.node[dev];
        candidate.push(newP);
      }

      dev++;
    }
    result[cnt++] = p;
  }
  if (cnt < k) puts("No");
  else
  {
    int len = result[k-1].nodecnt;
    printf("%d",result[k-1].node[len-1]+1);
    for (int i = len-2; i >= 0; i--)
      printf("-%d",result[k-1].node[i]+1);
    putchar('\n');
  }
  return 0;
}
```

3 Strings

3.1 KMP

求出 next 并返回 str 的循环周期。用于匹配过程一样。

```
int k_next[1111111];
int kmp(char* str,int len)
{
  int now = 0;
  for(int i = 1;i < len;i++)
  {
    while(now && str[i] != str[now]) now = k_next[now-1];
    if(str[i] == str[now]) now++;
    k_next[i] = now;
  }
  int period = len-(k_next[len-1]);
  if(len % period == 0) return period;
  return len;
}
```

3.2 MinimalCycleExp

返回 text 的所有循环同构中字典序最小的起始位置。$O(n)$

```
int MinimalRep(char* text,int len=-1)
{
  if(len == -1) len = strlen(text);

  int i = 0;
  int j = 1;
  while(i < len && j < len)
  {
    int k = 0;
    while(k < len && text[(i+k)%len] == text[(j+k)%len]) k++;
```

```
    if(k >= len) break;

    if(text[(i+k)%len] > text[(j+k)%len]) i = max(i+k+1,j+1);
    else j = max(i+1,j+k+1);
  }
  return min(i,j);
}
```

3.3 Gusfield

Also known as "Extended KMP". Usage: z_i = lcp(text+i,pattern) Run zFunction(z_pat,pat,pat,patLen,patLen) for self matching.

```
int z_pat[2222222] = {0};
int zFunction(int* z,char* text,char* pat,int textLen=-1,int patLen=-1)
{
  if(textLen == -1) textLen = strlen(text);
  if(patLen == -1) patLen = strlen(pat);

  int self = (text == pat && textLen == patLen);
  if(!self) zFunction(z_pat,pat,pat,patLen,patLen);
  else z[0] = patLen;

  int farfrom = 0;
  int far = self; // self->[farfrom,far) else [farfrom,far]
  for(int i = self;i < textLen;i++)
  {
    if(i+z_pat[i-farfrom] >= far)
    {
      int x = max(far,i);
      while(x < textLen && x-i < patLen && text[x] == pat[x-i]) x++;
      z[i] = x-i;
      if(i < x) { farfrom = i; far = x; }
    }
    else z[i] = z_pat[i-farfrom];
  }
  return 0;
}
```

3.4 Aho-Corasick

大部分应用基于一个性质：fail 指向与当前串的后缀相等的前缀最长的节点。另外可以模仿匹配过程在 Trie 上 DP 进行统计。Build a Trie then run the code below.

```
TNODE* Queue[66666];
int build_ac_automaton()
{
  int front = 0;
  int end = 0;
  Queue[end++] = Root;
  while(front != end)
  {
    TNODE* x = Queue[front++];
    for(int i = 0;i < 26;i++)
    {
      if(x->Child[i])
      {
        x->Child[i]->Fail = x->Fail?x->Fail->Child[i]:Root;
        // Spread additional info here for trie graph
        //x->Child[i]->Readable |= x->Child[i]->Fail->Readable;
        Queue[end++] = x->Child[i];
      }
      else x->Child[i] = x->Fail?x->Fail->Child[i]:Root; // trie graph
    }
  }
  return 0;
}
```

3.5 Manacher

rad_i 为以 i/2 为中心向两端延伸的最长回文长度。使用 rad 时注意是按照 ab->aabb 的 Pattern 填充过的，值二倍了。返回值为 Text 串中的最长回文长度，不需要除以 2 。

```
int rad[2222222];
int Manacher(char* Text,int len)
{
  len *= 2;

  int k = 0;
  for(int i = 0,j = 0;i < len;i += k,j = max(j-k,0))
  {
    while((i-j)/2 >= 0 && (i+j+1)/2 < len && Text[(i-j)/2] == Text[(i+j+1)
      /2]) j++;
    rad[i] = j;
    for(k = 1;i-k >= 0 && rad[i]-k >= 0 && rad[i-k] != rad[i]-k;k++)
      rad[i+k] = min(rad[i-k],rad[i]-k);
  }
  return *max_element(rad,rad+len);
}
```

3.6 Suffix Automaton

```
// Suffix Automaton //
// 自行定义 SAMNODE 结构体和相关 pool ，like a trie: child[],fa,len
SAMNODE* Root,*Last; // Must be inited!
```

```
int append_char(int ch)
{
  SAMNODE* x = Last;
  SAMNODE* t = SPTop++;
  t->len = x->len+1;
  while(x && !x->child[ch])
  {
    x->child[ch] = t;
    x = x->fa;
  }
  if(!x) t->fa = Root;
  else
  {
    SAMNODE* bro = x->child[ch];
    if(x->len+1 == bro->len) t->fa = bro; // actually it's fa.
    else
    {
      SAMNODE* nfa = SPTop++;
      nfa[0] = bro[0];
      nfa->len = x->len+1;
      bro->fa = t->fa = nfa;

      while(x && x->child[ch] == bro)
      {
        x->child[ch] = nfa;
        x = x->fa;
      }
    }
  }
  Last = t;
  return 0;
}

// SAM::Match //
SAMNODE* x = Root;
int mlen = 0;
for(int j = 0;j < len;j++)
{
  int ch = Text[j];
  /*// 强制后撤一个字符，部分情况下可能有用
  if(mlen == qlen) {
    mlen--;
    while(mlen <= x->fa->len) x = x->fa;
  } */
  if(x->child[ch]) { mlen++; x = x->child[ch]; }
  else
  {
    while(x && !x->child[ch]) x = x->fa;
    if(!x)
    {
      mlen = 0;
      x = Root;
    }
    else
    {
      mlen = x->len+1;
      x = x->child[ch];
    }
  }
  Match[j] = mlen;
} // End of SAM::Match //

// 基排方便上推一些东西，比如出现次数 //
SAMNODE* order[2222222];
int lencnt[1111111];
int post_build(int len)
{
  for(SAMNODE* cur = SPool;cur < SPTop;cur++) lencnt[cur->len]++;
  for(int i = 1;i <= len;i++) lencnt[i] += lencnt[i-1];
  int ndcnt = lencnt[len];
  for(SAMNODE* cur = SPTop-1;cur >= SPool;cur--) order[--lencnt[cur->len]] =
    cur;
  for(int i = ndcnt-1;i >= 0;i--) {
    // 此处上推
    if(order[i]->fa) order[i]->fa->cnt += order[i]->cnt;
  }
  return 0;
}
```

3.7 Suffix Array

```
int aa[222222];
int ab[222222];
int* rank,last_rank,ysorted;

int sa[222222];
char Str[222222];

int cmp(int l,int r,int step)
{
  return last_rank[l] == last_rank[r] && last_rank[l+step] == last_rank[r+
    step];
}

int rw[222222];
int rsort(int n,int m)
{
```

```
  for(int i = 0;i < m;i++) rw[i] = 0;
  for(int i = 0;i < n;i++) rw[rank[ysorted[i]]]++;
  for(int i = 1;i < m;i++) rw[i] += rw[i-1];
  for(int i = n-1;i >= 0;i--) sa[--rw[rank[ysorted[i]]]] = ysorted[i]; // keep order
  return 0;
}

int da(int n,int m) // n = strlen, m = alphabet size
{
  rank = aa; last_rank = ab; ysorted = ab;
  for(int i = 0;i < n;i++) { rank[i] = Str[i]; ysorted[i] = i; }
  rsort(n,m);

  int p = 0; // different suffix cnt.
  for(int step = 1;p < n;step *= 2)
  {
    ysorted = last_rank; // recycle use

    int cnt = 0;
    for(int i = n-step;i < n;i++) ysorted[cnt++] = i;
    for(int i = 0;i < n;i++) if(sa[i] >= step) ysorted[cnt++] = sa[i]-step;
    rsort(n,m);

    last_rank = rank;
    rank = ysorted;
    p = 1;
    rank[sa[0]] = 0;
    for(int i = 1;i < n;i++) rank[sa[i]] = cmp(sa[i],sa[i-1],step)?p-1:p++;

    m = p; // take care.
  }
  return 0;
}

int height[222222]; // lcp of suffix_i and suffix_{i-1}
int get_height(int n)
{
  int k = 0;
  for(int i = 0;i < n;i++)
  {
    if(rank[i] == 0) k = height[rank[i]] = 0;
    else
    {
      if(k > 0) k--;
      int j = sa[rank[i]-1];
      while(Str[i+k]==Str[j+k]) k++;
      height[rank[i]] = k;
    }
  }
  return 0;
}

int lcp(int i,int j)
{
  if(i == j) return n-i;
  if(rank[i] > rank[j]) swap(i,j);
  return rmq_querymin(rank[i]+1,rank[j]);
}
```

4 Geometry

4.1 Common

```
const double eps = 1e-8;

template<typename T>
inline T valsign(T x) { return x < 0 ? -1 : (x > 0 ? 1 : 0); }
inline double valsign(double x) { return x < -eps ? -1 : (x > eps ? 1 : 0); }

// hit on the edge will return true
bool is_segment_intersect(const POINT& A,const POINT& B,const POINT& C,const POINT& D)
{
  if(max(C.x,D.x) < min(A.x,B.x) || max(C.y,D.y) < min(A.y,B.y)) return false;
  if(max(A.x,B.x) < min(C.x,D.x) || max(A.y,B.y) < min(C.y,D.y)) return false;
  if(valsign((B-A)*(C-A))*valsign((B-A)*(D-A)) > 0) return false;
  if(valsign((D-C)*(A-C))*valsign((D-C)*(B-C)) > 0) return false;
  return true;
}

POINT get_perpfoot(const POINT& LineA,const POINT& LineB,const POINT& P)
{
  if(LineA.x == LineB.x) return POINT(LineA.x,P.y);
  if(LineA.y == LineB.y) return POINT(P.x,LineA.y);
  double k = (LineA.y-LineB.y)/(LineA.x-LineB.x);
  double x = (k*(k*LineA.x+(P.y-LineA.y))+P.x)/(k*k+1.0);
  return POINT(x,k*(x-LineA.x)+LineA.y);
}

bool is_point_onseg(const POINT& LineA,const POINT& LineB,const POINT& P)
{
  if(! (min(LineA.x,LineB.x) <= P.x && P.x <= max(LineA.x,LineB.x) &&
      min(LineA.y,LineB.y) <= P.y && P.y <= max(LineA.y,LineB.y)) )
```

```
    return false;
  if(valsign((P-LineA)*(LineB-LineA)) == 0) return true;
  return false;
}
```

4.2 Convex Hull

```
// P is input and Hull is output.
// return point count on hull
int Graham(POINT* P,POINT* Hull,int n)
{
  sort(P,P+n);
  int HTop = 0;
  for(int i = 0;i < n;i++)
  {
    // delete collinear points
    while(HTop > 1 && valsign((P[i]-Hull[HTop-2])*(Hull[HTop-1]-Hull[HTop-2])
      ) >= 0) HTop--;
    Hull[HTop++] = P[i];
  }
  int LTop = HTop;
  for(int i = n-2;i >= 0;i--)
  {
    while(HTop > LTop && valsign((P[i]-Hull[HTop-2])*(Hull[HTop-1]-Hull[HTop
      -2])) >= 0) HTop--;
    if(i) Hull[HTop++] = P[i];
  }
  return HTop;
}
```

4.3 Euclid Nearest

```
/* Usage:
   for(int i = 0;i < N;i++) yOrder[i] = i;
   sort(P,P+N,cmp_x);
   double result = closest_pair(0,N); // Won't change array "P" */

POINT P[111111];
int yOrder[111111];
inline bool cmp_x(const POINT& a,const POINT& b)
{
  return a.x==b.x?a.y<b.y:a.x<b.x;
}

inline bool cmp_y(const int a,const int b)
{
  POINT& A = P[a];
  POINT& B = P[b];
  return A.y==B.y?A.x<B.x:A.y<B.y;
}

int thisY[111111];
// [l,r)
double closest_pair(int l,int r)
{
  double ans = 1e100;
  if(r-l <= 6)
  {
    // just brute force_
    for(int i = l;i < r;i++)
    {
      for(int j = i+1;j < r;j++)
      {
        ans = min(ans,(P[i]-P[j]).hypot());
      }
    }
    sort(yOrder+l,yOrder+r,cmp_y);
    return ans;
  }

  int mid = (l+r)/2;
  ans = min(closest_pair(l,mid),closest_pair(mid,r));
  inplace_merge(yOrder+l,yOrder+mid,yOrder+r,cmp_y);

  int top = 0;
  double ll = P[mid].x;
  for(int i = l;i < r;i++)
  {
    double xx = P[yOrder[i]].x;
    if(ll-ans <= xx && xx <= ll+ans) thisY[top++] = yOrder[i];
  }

  for(int i = 0;i < top;i++)
  {
    for(int j = i+1;j < i+4 && j < top;j++)
    {
      ans = min(ans,(P[thisY[j]]-P[thisY[i]]).hypot());
    }
  }
  return ans;
}
```

4.4 Minimal Circle Cover

```
int getcircle(POINT& a,POINT& b,POINT& c,POINT& O,double& r)
{
  double a1 = 2.0*(a.x-b.x);
  double b1 = 2.0*(a.y-b.y);
  double c1 = a.x*a.x-b.x*b.x + a.y*a.y-b.y*b.y;
  double a2 = 2.0*(a.x-c.x);
  double b2 = 2.0*(a.y-c.y);
  double c2 = a.x*a.x-c.x*c.x + a.y*a.y-c.y*c.y;
  O.x = (c1*b2-c2*b1)/(a1*b2-a2*b1);
  O.y = (c1*a2-c2*a1)/(b1*a2-b2*a1);
  r = eudis(a,O);
  return 0;
}

POINT pt[100010] = {0};

int main(void)
{
  int n = 0;
  scanf("%d",&n);
  for(int i = 0;i < n;i++) scanf("%lf %lf",&pt[i].x,&pt[i].y);
  random_shuffle(pt,pt+n);

  double r = 0.0;
  POINT O = pt[0];
  for(int i = 1;i < n;i++)
  {
    if(eudis(pt[i],O)-r > -eps)
    {
      O.x = (pt[0].x+pt[i].x)/2.0;
      O.y = (pt[0].y+pt[i].y)/2.0;
      r = eudis(O,pt[0]);
      for(int j = 0;j < i;j++)
      {
        if(eudis(pt[j],O)-r > -eps)
        {
          O.x = (pt[i].x+pt[j].x)/2.0;
          O.y = (pt[i].y+pt[j].y)/2.0;
          r = eudis(O,pt[i]);
          for(int k = 0;k < j;k++)
          {
            if(eudis(pt[k],O)-r > -eps)
            {
              getcircle(pt[i],pt[j],pt[k],O,r);
            }
          }
        }
      }
    }
  }
  printf("%.10f\n%.10f %.10f\n",r,O.x,O.y);
  while(getchar() != EOF);
  return 0;
}
```

4.5 Rotate Carbin

返回凸包上最远点对距离

```
double RC(int N)
{
  double ans = 0.0;
  Hull[N] = Hull[0];
  int to = 1;
  for(int i = 0;i < N;i++)
  {
    while((Hull[i+1]-Hull[i])*(Hull[to]-Hull[i]) < (Hull[i+1]-Hull[i])*(Hull[
      to+1]-Hull[i])) to = (to+1)%N;
    //ans = max(ans,maxarea(i,to,N));
    ans = max(ans,(Hull[i]-Hull[to]).hypot2());
    ans = max(ans,(Hull[i+1]-Hull[to]).hypot2());
  }
  return sqrt(ans);
}
```

4.6 Simpson

```
inline double simpson(double fl,double fr,double fmid,double l,double r) {
  return (fl+fr+4.0*fmid)*(r-l)/6.0; }
double rsimpson(double slr,double fl,double fr,double fmid,double l,double r)
{
  double mid = (l+r)*0.5;
  double fml = f((l+mid)*0.5);
  double fmr = f((mid+r)*0.5);
  double slm = simpson(fl,fmid,fml,l,mid);
  double smr = simpson(fmid,fr,fmr,mid,r);
  if(fabs(slr-slm-smr) < eps) return slm+smr;
  return rsimpson(slm,fl,fmid,fml,l,mid)+rsimpson(smr,fmid,fr,fmr,mid,r);
}
```

5 Yangyue's Geometry Template

5.1 Common

```cpp
int dcmp(double a, double b) {
  if (fabs(a − b) < eps) return 0;
  if (b − a > eps) return −1;
  return 1;
}
int dcmp0(double x) {
  if (x > eps) return 1;
  if (x < −eps) return −1;
  return 0;
}

struct Point {
  double x, y;
  Point(){}
  Point(double x, double y) : x(x), y(y) {}
  double operator * (const Point &b) {
    return x * b.y − y * b.x;
  }
  Point operator − (const Point &b) {
    return Point(x − b.x, y − b.y);
  }
  Point operator + (const Point &b) {
    return Point(x + b.x, y + b.y);
  }
  Point operator / (const double &t) {
    return Point(x / t, y / t);
  }
  Point operator * (const double &t) {
    return Point(x * t, y * t);
  }
  double operator %(const Point &b) {
    return x * b.x + y * b.y;
  }
  bool operator < (const Point &b) const {
    if (dcmp(x, b.x) != 0) return dcmp(x, b.x) < 0;
    return dcmp(y, b.y) < 0;
  }
  bool operator == (const Point &b) const {
    if (dcmp(x, b.x) != 0) return 0;
    if (dcmp(y, b.y) != 0) return 0;
    return 1;
  }
};
```

5.2 Intersection

```cpp
//判规范相交
bool stdcross(Point A, Point B, Point C, Point D) {
  int t1, t2;
  t1 = dcmp0((C − A) * (B − A));
  t2 = dcmp0((D − A) * (B − A));
  if (t1 * t2 >= 0) return 0;
  t1 = dcmp0((B − C) * (D − C));
  t2 = dcmp0((A − C) * (D − C));
  if (t1 * t2 >= 0) return 0;
  return 1;
}

//点是否在线段上
bool PinSeg(Point P, Point A, Point B) {
  if (dcmp0((P − A) * (B − A)) != 0) return 0;
  if (dcmp(P.x, min(A.x, B.x)) < 0 || dcmp(P.x, max(A.x, B.x)) > 0) return 0;
  if (dcmp(P.y, min(A.y, B.y)) < 0 || dcmp(P.y, max(A.y, B.y)) > 0) return 0;
  return 1;
}

//非规范相交
bool unstdcross(Point A, Point B, Point C, Point D) {
  if (stdcross(A, B, C, D)) return 1;
  if (PinSeg(A, C, D)) return 1;
  if (PinSeg(B, C, D)) return 1;
  if (PinSeg(C, A, B)) return 1;
  if (PinSeg(D, A, B)) return 1;
}

//相交不重合线段求交点
Point getcrossPoint(Point A, Point B, Point C, Point D) {
  if (PinSeg(A, C, D)) return A;
  if (PinSeg(B, C, D)) return B;
  if (PinSeg(C, A, B)) return C;
  if (PinSeg(D, A, B)) return D;
  double S1 = fabs((C − A) * (B − A));
  double S2 = fabs((D − A) * (B − A));
  double k = S2 / (S1 + S2);
  Point vect = C − D;
  return Point(D.x + k * vect.x, D.y + k * vect.y);
}
```

5.3 Point3D

```cpp
struct Point3D {
  double x, y, z;
  Point3D(){}
  Point3D(double x, double y, double z)
    : x(x), y(y), z(z) {}
```

```
  Point3D operator -(const Point3D &b) {
    return Point3D(x - b.x, y - b.y, z - b.z);
  }
  Point3D operator /(const double &t) {
    return Point3D(x / t, y / t, z / t);
  }
  Point3D operator +(const double &t) {
    return Point3D(x + t, y + t, z + t);
  }
  Point3D operator +(const Point3D &b) {
    return Point3D(x + b.x, y + b.y, z + b.z);
  }
  Point3D operator *(const double &t) {
    return Point3D(x * t, y * t, z * t);
  }
  double operator %(const Point3D &b) {
    return x * b.x + y * b.y + z * b.z;
  }
  Point3D operator *(const Point3D &b) {
    double i = y * b.z - b.y * z;
    double j = z * b.x - x * b.z;
    double k = x * b.y - b.x * y;
    return Point3D(i, j, k);
  }
  double len() {
    return sqrt( sqr(x) + sqr(y) + sqr(z) );
  }
  void init() {
    scanf("%lf%lf%lf",&x,&y,&z);
  }
};
```

5.4 Graham

二维凸包 solve to b[].

```
void graham() {

  sort(b, b + n);
  Point t = b[0];
  for (int i = 0; i < n; ++i) b[i] = b[i] - t;
  int m = unique(b, b + n) - b;
  //printf("%d\n", cmp(b[0], b[1]));

  //printf("%.10lf\n%.10lf\n", atan2(0, 0), atan2(0.70, -0.40));
  sort(b + 1, b + m, cmp);

  top = 2;
  stack[1] = 0;
  stack[2] = 1;
  for (int i = 2; i < m; ++i) {
    while (top > 1 && dcmp0(xcross2D(b[stack[top - 1]], b[stack[top]], b[i]))
       <= 0 ) --top;
    stack[++top] = i;
  }
}
```

5.5 3D Convex Hull

```
struct Hull3D {
  struct Plane {
    int a, b, c;
    bool ok;
    Plane(){}
    Plane(int a, int b, int c, bool ok)
      : a(a), b(b), c(c), ok(ok) {}
  };
  int n, tricnt;        //初始点数
  int vis[MaxN][MaxN];  //点i到点j是属于哪个面
  Plane tri[MaxN << 2]; //凸包三角形
  Point3D Ply[MaxN];    //初始点
  double dist(Point3D a) {
    return sqrt(a.x * a.x + a.y * a.y + a.z * a.z);
  }
  double area(Point3D a, Point3D b, Point3D c) {
    return dist((b - a) * (c - a));
  }
  double volume(Point3D a, Point3D b, Point3D c, Point3D d) {
    return ((b - a) * (c - a)) % (d - a);
  }
  double PtoPlane(Point3D &P, Plane f) {  // 正 : 面同向{
    Point3D m = Ply[f.b] - Ply[f.a];
    Point3D n = Ply[f.c] - Ply[f.a];
    Point3D t = P - Ply[f.a];
    return (m * n) % t;
  }
  void deal(int p, int a, int b) {
    int f = vis[a][b];
    Plane add;
    if (tri[f].ok) {
      if ((PtoPlane(Ply[p], tri[f])) > eps) dfs(p, f);
      else {
        add = Plane(b, a, p, 1);
        vis[p][b] = vis[a][p] = vis[b][a] = tricnt;
        tri[tricnt++] = add;
      }
    }
```

```
  }
  void dfs(int p, int cnt) { // 维护凸包，如果点p在凸包外更新凸包
    tri[cnt].ok = 0;
    deal(p, tri[cnt].b, tri[cnt].a);
    deal(p, tri[cnt].c, tri[cnt].b);
    deal(p, tri[cnt].a, tri[cnt].c);
  }
  bool same(int s, int e) { //判面是否相同
    Point3D a = Ply[tri[s].a];
    Point3D b = Ply[tri[s].b];
    Point3D c = Ply[tri[s].c];
    return fabs(volume(a, b, c, Ply[tri[e].a])) < eps
      && fabs(volume(a, b, c, Ply[tri[e].b])) < eps
      && fabs(volume(a, b, c, Ply[tri[e].c])) < eps;
  }
  void construct() { //构造凸包
    tricnt = 0;
    if (n < 4) return;
    bool tmp = 1;
    for (int i = 1; i < n; ++i) { // 两两不共点
      if (dist(Ply[0] - Ply[i]) > eps) {
        swap(Ply[1], Ply[i]);
        tmp = 0;
        break;
      }
    }
    if (tmp) return;
    tmp = 1;
    for (int i = 2; i < n; ++i) { //前三点不共线
      if ((dist((Ply[0] - Ply[1]) * (Ply[1] - Ply[i]))) > eps) {
        swap(Ply[2], Ply[i]);
        tmp = 0;
        break;
      }
    }
    if (tmp) return;
    tmp = 1;
    for (int i = 3; i < n; ++i) { //前四点不共面
      if (fabs((Ply[0] - Ply[1]) * (Ply[1] - Ply[2]) % (Ply[0] - Ply[i])) >
        eps) {
        swap(Ply[3], Ply[i]);
        tmp = 0;
        break;
      }
    }
    if (tmp) return;
    Plane add;
    for (int i = 0; i < 4; ++i) { //初始四面体
      add = Plane((i + 1) % 4, (i + 2) % 4, (i + 3) % 4, 1);
      if (PtoPlane(Ply[i], add) > 0) swap(add.b, add.c);
      vis[add.a][add.b] = vis[add.b][add.c] = vis[add.c][add.a] = tricnt;
      tri[tricnt++] = add;
    }
    for (int i = 4; i < n; ++i) { //构建凸包
      for (int j = 0; j < tricnt; ++j) {
        if (tri[j].ok && (PtoPlane(Ply[i], tri[j])) > eps) {
          dfs(i, j);
          break;
        }
      }
    }
    int cnt = tricnt; tricnt = 0;
    for (int i = 0; i < cnt; ++i) { //删除无用的面
      if (tri[i].ok) {
        tri[tricnt++] = tri[i];
      }
    }
  }
  int Planepolygon() { //多少个面
    int res = 0;
    for (int i = 0; i < tricnt; ++i) {
      bool yes = 1;
      for (int j = 0; j < i; ++j) {
        if (same(i, j)) {
          yes = 0;
          break;
        }
      }
      if (yes) ++res;
    }
    return res;
  }
  // Volume = sigma(volume(p, a, b, c));  i = 0..tricnt - 1;
} Hull;
```

5.6 Halfplane

半平面交.. 直线的左侧需注意半平面的方向！不等式有解等价与 $cnt > 1$

```
struct Segment {
  Point s, e;
  double angle;
  Segment(){}
  Segment(Point s, Point e)
    : s(s), e(e) {
      angle = atan2(e.y - s.y, e.x - s.x);
    }
} segment[MaxN], seg[MaxN];
```

```
Point get_intersect(Segment s1, Segment s2) {
  double u = xmul(s1.s, s1.e, s2.s);
  double v = xmul(s1.e, s1.s, s2.e);
  Point t;
  t.x = (s2.s.x * v + s2.e.x * u) / (u + v);
  t.y = (s2.s.y * v + s2.e.y * u) / (u + v);
  return t;
}
bool cmp(Segment a, Segment b) {
  if (dcmp(a.angle - b.angle) == 0) return dcmp(xmul(a.s, a.e, b.s)) > 0;
  return dcmp(a.angle - b.angle) < 0;
  return 0;
}
bool IsParallel(Segment P, Segment Q) {
  return dcmp((P.e - P.s) * (Q.e - Q.s)) == 0;
}
Segment deq[MaxN];
int HalfPlaneIntersect(Segment seg[], int n) {
  sort(seg, seg + n, cmp);
  int tmp = 1;
  for (int i = 1; i < n; ++i) {
    if (dcmp(seg[i].angle - seg[tmp - 1].angle) != 0) {
      seg[tmp++] = seg[i];
    }
  }
  n = tmp;
  //for (int i = 0; i < n; ++i) printf("%.0lf %.0lf %.0lf %.0lf\n", seg[i].s.
    x, seg[i].s.y, seg[i].e.x, seg[i].e.y);
  deq[0] = seg[0]; deq[1] = seg[1];
  int front = 0, tail = 1;
  for (int i = 2; i < n; ++i) {
    if(IsParallel(deq[tail], deq[tail-1]) || IsParallel(deq[front], deq[front
      +1])) return 0;
    while (front < tail && dcmp(xmul(seg[i].s, seg[i].e, get_intersect(deq[
      tail], deq[tail - 1]))) < 0) --tail;
    while (front < tail && dcmp(xmul(seg[i].s, seg[i].e, get_intersect(deq[
      front], deq[front+1]))) < 0) ++front;
    deq[++tail] = seg[i];
  }
  while(front < tail && xmul(deq[front].s, deq[front].e, get_intersect(deq[
    tail], deq[tail-1])) < -eps) tail--;
  while(front < tail && xmul(deq[tail].s, deq[tail].e, get_intersect(deq[
    front], deq[front+1])) < -eps) front++;
  int cnt = 0;
  deq[++tail] = deq[front];
  for (int i = front; i + 1 < tail; ++i) p[cnt++] = get_intersect(deq[i], deq
    [i+1]);
  return cnt;
}
```

6 Damn Math

6.1 DFT

n 需要为 2 的次幂，sign 传入 1 时正变换，传入 -1 时逆变换，逆变换后需要手动除以 n。

```
typedef complex<double> cplx;
inline unsigned int intrev(unsigned x)
{
  x = ((x & 0x55555555U) << 1) | ((x & 0xAAAAAAAAU) >> 1);
  x = ((x & 0x33333333U) << 2) | ((x & 0xCCCCCCCCU) >> 2);
  x = ((x & 0x0F0F0F0FU) << 4) | ((x & 0xF0F0F0F0U) >> 4);
  x = ((x & 0x00FF00FFU) << 8) | ((x & 0xFF00FF00U) >> 8);
  x = ((x & 0x0000FFFFU) << 16) | ((x & 0xFFFF0000U) >> 16);
  return x;
};
void fft(int sign, cplx* data, int n)
{
  int d = 1+__builtin_clz(n);
  double theta = sign * 2.0 * PI / n;
  for(int m = n;m >= 2;m >>= 1, theta *= 2)
  {
    cplx tri = cplx(cos(theta),sin(theta));
    cplx w = cplx(1,0);
    for(int i = 0, mh = m >> 1; i < mh; i++)
    {
      for(int j = i;j < n;j += m)
      {
        int k = j+mh;
        cplx tmp = data[j]-data[k];

        data[j] += data[k];
        data[k] = w * tmp;
      }
      w *= tri;
    }
  }
  for(int i = 0;i < n;i++)
  {
    int j = intrev(i) >> d;
    if(j < i) swap(data[i],data[j]);
  }
  return;
}
```

6.2 Linear Eratosthenes Sieve

```
int MinDivi[11111111];
int Prime[1111111];
int PCnt = 0;
int Miu[11111111];
int Phi[11111111];

int era(int N)
{
  for(int i = 2;i <= N;i++)
  {
    if(!MinDivi[i])
    {
      Prime[PCnt++] = i;
      MinDivi[i] = i;
      Miu[i] = -1;
      Phi[i] = i-1;
    }
    for(int j = 0;j < PCnt && Prime[j] <= MinDivi[i] && i*Prime[j] <= N;j++)
    {
      MinDivi[i*Prime[j]] = Prime[j];
      Miu[i*Prime[j]] = -Miu[i];
      if(Prime[j] == MinDivi[i]) Miu[i*Prime[j]] = 0;
      Phi[i*Prime[j]] = Phi[i]*(Prime[j]-(Prime[j] != MinDivi[i]));
    }
  }
  return 0;
}
```

6.3 Miller-Rabin

```
// Always call "IsPrime" unless you know what are you doing

int MillerRabin(ull a,ull n)
{
  if(n == 2) return 1;
  if(n == 1 || (n & 1) == 0) return 0;
  ull d = n-1;
  while((d & 1) == 0) d >>= 1;
  ull t = powmod(a,d,n);
  while(d != n-1 && t != 1 && t != n-1)
  {
    t = mulmod(t,t,n);
    d <<= 1;
  }
  return (t == n-1) || ((d & 1) == 1);
}

int LPrimes[] = {2,3,5,7,11,13,17,19,23};
int IsPrime(ull n)
{
  int result = 1;
  for(int i = 0;i < sizeof(LPrimes)/sizeof(int);i++)
  {
    if(LPrimes[i] >= n) break;
    result &= MillerRabin(LPrimes[i],n);
    if(!result) return result;
  }
  return result;
}
```

6.4 NTT

最终结果 $\bmod P$. $N \le 200000$

$E_i \equiv g^{(P_i-1)\div N_1} \pmod{P_i}$ $F_i \equiv 1 \div E_i \pmod{P_i}$ $I_i \equiv 1 \div N_1 \pmod{P_i}$

```
namespace NTT {
  const int P = 1000003; const int N = 100010; const int N1 = 262144;
  const int P1 = 998244353; const int P2 = 995622913;
  const int E1 = 996173970; const int E2 = 88560779;
  const int F1 = 121392023; const int F2 = 840835547;
  const int I1 = 998240545; const int I2 = 995619115;
  const LL M1 = 397550359381069386LL; const LL M2 = 596324591238590904LL;
  const LL MM = 993874950619660289LL;
  LL mul(LL x, LL y, LL z) {
    return (x * y - (LL) (x / (long double) z * y + 1e-3) * z + z) % z;
  }
  int crt(int x1, int x2) {
    return (mul(M1, x1, MM) + mul(M2, x2, MM)) % MM % P;
  }
  void NTT(int *A, int PM, int PW) {
    for (int m = N1, h; h = m / 2, m >= 2; PW = (LL) PW * PW % PM, m = h) {
      for (int i = 0, w = 1; i < h; ++i, w = (LL) w * PW % PM)
      for (int j = i; j < N1; j += m) {
        int k = j + h, x = (A[j] - A[k] + PM) % PM;
        A[j] += A[k]; A[j] %= PM;
        A[k] = (LL) w * x % PM;
      }
    }
    for (int i = 0, j = 1; j < N1 - 1; ++j) {
      for (int k = N1 / 2; k > (i^=k); k /= 2);
      if (j < i) swap(A[i], A[j]);
    }
  }
  int A1[MaxN], B1[MaxN], C1[MaxN];
  void mul(int *A, int *B, int *C, int n) {
    memset(C, 0, sizeof(*C)*N1);
```

```
    memcpy(A1, A, sizeof(*A)*N1);
    memcpy(B1, B, sizeof(*B)*N1);
    NTT(A1, P1, E1);
    NTT(B1, P1, E1);
    for (int i = 0; i < N1; ++i) C1[i] = (LL) A1[i] * B1[i] % P1;
    NTT(C1, P1, F1);
    for (int i = 0; i < N1; ++i) C1[i] = (LL) C1[i] * I1 % P1;
    NTT(A, P2, E2);
    NTT(B, P2, E2);
    for (int i = 0; i < N1; ++i) C[i] = (LL) A[i] * B[i] % P2;
    NTT(C, P2, F2);
    for (int i = 0; i < N1; ++i) C[i] = (LL) C[i] * I2 % P2;
    for (int i = 0; i < N1; ++i) C[i] = crt(C1[i], C[i]);
    for (int i = n; i < N1; ++i) C[i] = 0;
  }
}
```

6.5 Pollard-Rho

```
ull PollardRho(ull n,int c)
{
  ull x = 2;
  ull y = 2;
  ull d = 1;
  //printf("%d\n",c);
  while(d == 1)
  {
    x = (mulmod(x,x,n)+c)%n;
    y = (mulmod(y,y,n)+c)%n;
    y = (mulmod(y,y,n)+c)%n;
    //printf("%I64u %I64u %I64u\n",x,y,d);

    if(x > y) d = gcd(x-y,n);
    else d = gcd(y-x,n);
  }
  return d;
}

// DO NOT CALL THIS WITH A PRIME!
ull Factorize(ull n)
{
  ull d = n;
  while(d == n) d = PollardRho(n,rand()+1);
  return d;
}

ull dv[111111];
int dvcnt = 0;

// call sort if sorted results needed.
ull FullFactorize(ull n)
{
  if(n%2 == 0)
  {
    dv[dvcnt++] = 2;
    while(n%2 == 0) n /= 2;
    return FullFactorize(n);
  }
  ull t = 0;
  while(n != 1 && !IsPrime(n))
  {
    t = Factorize(n);
    int cdvc = dvcnt;
    if(!IsPrime(t)) FullFactorize(t);
    else dv[dvcnt++] = t;
    for(int i = cdvc;i < dvcnt;i++)
    {
      while(n % dv[i] == 0) n /= dv[i];
    }
  }
  if(n != 1) dv[dvcnt++] = n;
  return 0;
}
```

6.6 Simplex

```
// Linar programming, array all indexed from 0
double a[MaxN][MaxN], b[MaxN], c[MaxN], d[MaxN][MaxN];
int ix[MaxN + MaxN];
// max{cx|Ax<=b,x>=0}, n: constraints, m: vars
double simplex(double a[][MaxN], double b[], double c[], int n, int m) {
  ++m;
  int r = n, s = m - 1;
  memset(d, 0, sizeof(d));
  for (int i = 0; i < n + m; ++i) ix[i] = i;
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < m - 1; ++j) d[i][j] = -a[i][j];
    d[i][m - 1] = 1;
    d[i][m] = b[i];
    if (d[r][m] > d[i][m]) r = i;
  }
  for (int j = 0; j < m - 1; ++j) d[n][j] = c[j];
  d[n + 1][m - 1] = -1;
  for (double dd;; ) {
    if (r < n) {
      int t = ix[s]; ix[s] = ix[r + m]; ix[r + m] = t;
```

```
      d[r][s] = 1.0 / d[r][s];
      for (int j = 0; j <= m; ++j) if (j != s) d[r][j] *= -d[r][s];
      for (int i = 0; i <= n + 1; ++i) if (i != r) {
        for (int j = 0; j <= m; ++j) if (j != s) d[i][j] += d[r][j] * d[i][s
          ];
        d[i][s] *= d[r][s];
      }
    }
    r = -1; s = -1;
    for (int j = 0; j < m; ++j) if (s < 0 || ix[s] > ix[j]) {
      if (d[n + 1][j] > eps || (d[n + 1][j] > -eps && d[n][j] > eps)) s = j;
    }
    if (s < 0) break;
    for (int i = 0; i < n; ++i) if (d[i][s] < -eps) {
      if (r < 0 || (dd = d[r][m] / d[r][s] - d[i][m] / d[i][s]) < -eps || (dd
         < eps && ix[r + m] > ix[i + m])) r = i;
    }
    if (r < 0) return -1; // not bounded
  }
  if (d[n + 1][m] < -eps) return -1; // not executable
  double ans = 0;
  for (int i = m; i < n + m; ++i) {
    if (ix[i] < m - 1) ans += d[i - m][m] * c[ix[i]];
  }
  return ans;
}
```

7 Others

7.1 DLX

```
const int MAXINT = 0x7FFFFFFF;

struct DLXNODE
{
  union
  {
    int S;
    DLXNODE* C;
  };
  int Row;
  DLXNODE *U,*D,*L,*R;
};

DLXNODE H;
DLXNODE NodePool[10000] = {0};
int PoolTop = 0;

DLXNODE* node_alloc()
{
  memset(&NodePool[PoolTop],0,sizeof(DLXNODE));
  return &NodePool[PoolTop++];
}

int ans[100] = {0}; // 9x9

int remove(DLXNODE* c)
{
  c->L->R = c->R;
  c->R->L = c->L;

  for(DLXNODE* i = c->D;i != c;i = i->D)
  {
    for(DLXNODE* j = i->R;j != i;j = j->R)
    {
      j->U->D = j->D;
      j->D->U = j->U;
      j->C->S--;
    }
  }
  return 0;
}

int resume(DLXNODE* c)
{
  for(DLXNODE* i = c->D;i != c;i = i->D)
  {
    for(DLXNODE* j = i->L;j != i;j = j->L)
    {
      j->U->D = j;
      j->D->U = j;
      j->C->S++;
    }
  }

  c->L->R = c;
  c->R->L = c;
  return 0;
}

bool dfs(int k)
{
  if(H.R == &H)
  {
    // found!
    return true;
```

```
  }

  DLXNODE* tc = NULL;
  int ts = MAXINT;
  for(DLXNODE* i = H.R;i != &H;i = i->R)
  {
    if(i->S < ts)
    {
      ts = i->S;
      tc = i;
    }
  }
  if(ts == MAXINT) return true;
  remove(tc);
  for(DLXNODE* i = tc->U;i != tc;i = i->U)
  {
    ans[k] = i->Row; // store state here
    for(DLXNODE* j = i->R;j != i;j = j->R)
    {
      remove(j->C);
    }
    if(dfs(k+1)) return true;
    for(DLXNODE* j = i->L;j != i;j = j->L)
    {
      resume(j->C);
    }
  }
  resume(tc);
  return false;
}

DLXNODE* insert_to_col(DLXNODE* c,int RowNo,DLXNODE* rl)
{
  DLXNODE* node = node_alloc();
  // c->U is last node
  node->U = c->U;
  node->D = c;
  if(!rl) node->L = node->R = node;
  else
  {
    node->L = rl;
    node->R = rl->R;
    rl->R->L = node;
    rl->R = node;
  }
  node->C = c;
  node->Row = RowNo;
  c->S++;
  c->U->D = node;
  c->U = node;
  return node;
}

// 对应 9x9 数独的建图
int main(void)
{
  char Scene[100] = {0};
  while(scanf("%s",Scene) != EOF && strcmp(Scene,"end"))
  {
    PoolTop = 0;
    memset(ans,0,sizeof(ans));

    H.L = &H;
    H.R = &H;
    H.D = &H;
    H.U = &H;
    DLXNODE* cFind[324] = {0};
    DLXNODE* last = &H;
    for(int i = 0;i < 324;i++)
    {
      DLXNODE* tn = node_alloc();
      cFind[i] = tn;
      tn->S = 0;
      tn->D = tn;
      tn->U = tn;
      tn->L = last;
      tn->R = last->R;
      last->R->L = tn;
      last->R = tn;
      last = tn;
    }
    for(int i = 0;i < 9;i++)
    {
      for(int j = 0;j < 9;j++)
      {
        int s = 1; int e = 9;
        if(Scene[i*9+j] != '.')
        {
          s = e = Scene[i*9+j]-'0';
        }
        for(int k = s;k <= e;k++)
        {
          int b = (i/3)*3+j/3;

          int RowNo = i*9*9+j*9+k-1;

          DLXNODE* ln = NULL;
          ln = insert_to_col(cFind[i*9+j],RowNo,ln);
```

```
          ln = insert_to_col(cFind[81+i*9+k-1],RowNo,ln);
          ln = insert_to_col(cFind[162+j*9+k-1],RowNo,ln);
          ln = insert_to_col(cFind[243+b*9+k-1],RowNo,ln);
        }
      }
    }
    dfs(0);
    for(int i = 0;i < 81;i++)
    {
      int RNo = ans[i];
      int k = RNo % 9 + 1;
      int j = RNo / 9 % 9;
      int r = RNo / 81;
      Scene[r*9+j] = '0' + k;
    }
    printf("%s\n",Scene);
  }
  return 0;
}
```

7.2 FastIO For Java

```
BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
StringTokenizer tokenizer = null;

public String next()
{
  while (tokenizer == null || !tokenizer.hasMoreTokens()) {
    try {
      tokenizer = new StringTokenizer(reader.readLine());
    } catch(IOException e) {
      throw new RuntimeException(e);
    }
  }
  return tokenizer.nextToken();
}

public int nextInt() {
  return Integer.parseInt(next()); // Double. .....
}
```

7.3 Java References

有一个叫 DecimalFormat 的东西。
有一个叫 BufferedInputStream 的东西。
有一个叫 FileInputStream 的东西。

8 外挂

8.1 mulmod

```
/* return x*y%mod. no overflow if x,y < mod
 * remove 'i' in "idiv"/"imul" if change to unsigned*/
inline int mulmod(int x,int y,int mod)
{
  int ans = 0;
  __asm__
  (
    "movl %1,%%eax\n"
    "imull %2\n"
    "idivl %3\n"

    :"=d"(ans)
    :"m"(x),"m"(y),"m"(mod)
    :"%eax"
  );
  return ans;
}
```

8.2 stack

修改 esp 到手动分配的内存。慎用！可能违反某些规则或造成不必要的 RE/WA 。

```
int main(void)
{
  char* SysStack = NULL;
  char* MyStack = new char[33554432];
  MyStack += 33554432-1048576; // 32M
  __asm__(
    "movl %%esp,%%eax\n\t"
    "movl %1,%%esp\n\t"
    :"=a"(SysStack)
    :"m"(MyStack)
  );
  mmain();
  __asm__(
    "movl %0,%%esp\n\t"
    ::"m"(SysStack)
  );
  return 0;
}
```