爱吃吗/爱吃泡菜 Code Library

Dark History

March 25, 2014

目录

1 Data Structure

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

递归版本，NodeID 为全局 ID，保证了 dfs 序。如果需要非递归版本，先写 bfs 算好 fa/Depth/TreeSize/HeavyChild，然后抄下面 Hint 部分。

```
int BlockRoot[111111];
int NodeID[111111];
int NodeID_Out[111111]; // 离开节点的时候的序dfs
int IndexToNode[111111];
int TreeSize[111111];
int Depth[111111];
int HeavyChild[111111]; // 0 if not set
int fa[111111];
int idx = 0;
int dfs_size(int x)
{
  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;
    dfs_size(y);
    TreeSize[x] += TreeSize[y];
    if(TreeSize[HeavyChild[x]] < TreeSize[y]) HeavyChild[x] = y;
  }
  return 0;
}
int dfs_lh(int x,int block)
{
  BlockRoot[x] = block;
  NodeID[x] = ++idx;
  IndexToNode[idx] = x;
  if(HeavyChild[x]) dfs_lh(HeavyChild[x],block);
  for(EDGE* e = E[x];e;e = e->Next)
  {
    int y = e->y;
    if(y == fa[x] || y == HeavyChild[x]) continue;
    dfs_lh(y,y);
  }
  NodeID_Out[x] = idx;
  return 0;
}
int Decomposition(int s,int N)
{
  idx = 0; fa[s] = 0;
  memset(HeavyChild,0,sizeof(HeavyChild[0])*(N+10));
  dfs_size(s); dfs_lh(s,s);
  return 0;
}

// 如果需要非递归的，一点提示，都会写，后面的：bfs
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];
  }
}

// 参考用爬树过程
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)

需要改成持久化的话每次修改的时候新建节点即可，然后去掉对 fa 的维护即可。必要的情况下在 newNode 里面加上 GC。

```
struct TNODE
{
  int val,rd,size;
  TNODE* left,*right,*fa;

  inline int update()
  {
    size = 1;
    if(left) { size += left->size; left->fa = this; }
    if(right) { size += right->size; right->fa = this; }
    fa = NULL;
    return 0;
  }
};
typedef pair<TNODE*,TNODE*> ptt;
TNODE TPool[233333];
TNODE* TPTop = TPool;

inline int real_rand() { return ((rand()&32767)<<15)^rand(); }
TNODE* newNode(int val,TNODE* left=NULL,TNODE* right=NULL)
{
  TNODE* result = TPTop++;
  result->val = val; result->rd = real_rand(); result->left = left; result->
    right = right; result->fa = NULL;
  result->update();
  return result;
}
```

```

TNODE* Merge(TNODE* t1,TNODE* t2)
{
  if(!t1) return t2;
  if(!t2) return t1;
  if(t1->rd <= t2->rd) { t1->right = Merge(t1->right,t2); t1->update();
    return t1; }
  else { t2->left = Merge(t1,t2->left); t2->update(); return t2; }
}

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)
  {
    TNODE* oleft = x->left;
    if(x->left) x->left->update();
    x->left = NULL;
    x->update();
    return ptt(oleft,x);
  }
  if(pos < lsize)
  {
    ptt st = Split(x->left,pos);
    x->left = st.second; x->update(); if(st.first) st.first->update();
    return ptt(st.first,x);
  }
  else
  {
    ptt st = Split(x->right,pos-lsize-1);
    x->right = st.first; x->update(); if(st.second) st.second->update();
    return ptt(x,st.second);
  }
}

inline int Rank(TNODE* x)
{
  int ans = x->left ? x->left->size : 0;
  while(x->fa)
  {
    if(x == x->fa->right) ans += (x->fa->left ? x->fa->left->size : 0) + 1;
    x = x->fa;
  }
  return ans;
}
```

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。

对于一条边，如果他的两个点分属不同的连通分量且满流则这条边可属于网络的最小割。

如果他的两个点分属不同的联通分量且满流且两个点分别和 source，sink 属于同一个连通分量，则这条边必属于最小割。

```
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 KM

```
int n,nx,ny, m;
int link[MaxN],lx[MaxN],ly[MaxN],slack[MaxN];
int visx[MaxN],visy[MaxN],w[MaxN][MaxN];

int DFS(int x)
{
  visx[x] = 1;
  for (int y = 1;y <= ny;y ++)
  {
    if (visy[y])
      continue;
    int t = lx[x] + ly[y] - w[x][y];
    if (t == 0)       //
    {
      visy[y] = 1;
      if (link[y] == -1||DFS(link[y]))
      {
        link[y] = x;
        return 1;
      }
    }
    else if (slack[y] > t)
      slack[y] = t;
  }
  return 0;
}
void KM()
{
  int i,j;
  memset (link,-1,sizeof(link));
  memset (ly,0,sizeof(ly));
  for (i = 1;i <= nx;i ++)
    for (j = 1,lx[i] = -INF;j <= ny;j ++)
      if (w[i][j] > lx[i])
        lx[i] = w[i][j];

  for (int x = 1;x <= nx;x ++)
  {
    for (i = 1;i <= ny;i ++)
      slack[i] = INF;
    while (1)
    {
      memset (visx,0,sizeof(visx));
      memset (visy,0,sizeof(visy));
      if (DFS(x))
        break;
      int d = INF;
      for (i = 1;i <= ny;i ++)
        if (!visy[i]&&d > slack[i])
          d = slack[i];
      for (i = 1;i <= nx;i ++)
        if (visx[i])
          lx[i] -= d;
      for (i = 1;i <= ny;i ++)
        if (visy[i])
          ly[i] += d;
        else
          slack[i] -= d;
    }
  }
}
```

2.11 MinCostMaxFlow (Dijkstra SP)

如果点数真的很少，可以把 Dijkstra 改成 $O(n^2)$ 的实现，然后换成邻接矩阵，可以快不少。

```
bool vis[MAXN];
int d1[MAXN],dis[MAXN],pre[MAXN],Queue[MAXN];;
ARC* prearc[MAXN];
int dijkstra(int s,int t,int n)
{
  memset(vis,0,sizeof(vis[0])*n);
  memset(dis,0x7F,sizeof(*dis)*n);
  set<pii> q; dis[s] = 0;
  q.insert(pii(dis[s]-d1[s],s));
  while(!q.empty())
  {
    set<pii>::iterator it = q.begin();
    int x = it->second;
    int xdis = dis[x];
    q.erase(it);
    vis[x] = true;
    for(ARC* a = Arc[x];a;a = a->Next)
    {
      if(a->c <= 0 || vis[a->y]) continue;
      if(xdis + a->w < dis[a->y])
      {
        q.erase(pii(dis[a->y]-d1[a->y],a->y));
        dis[a->y] = xdis + a->w;
        q.insert(pii(dis[a->y]-d1[a->y],a->y));
```

```
      }
    }
  }
  return dis[t];
}

bool bfs(int s,int t,int n)
{
  memset(vis,0,sizeof(vis[0])*n);
  int qf = 0;
  int qe = 0;
  vis[s] = true; Queue[qe++] = s;
  while(qf < qe)
  {
    int x = Queue[qf++];
    if(x == t) break;

    for(ARC* a = Arc[x];a;a = a->Next)
    {
      if(a->c <= 0) continue;
      if(!vis[a->y] && dis[x] + a->w == dis[a->y])
      {
        vis[a->y] = true;
        pre[a->y] = x;
        prearc[a->y] = a;
        Queue[qe++] = a->y;
      }
    }
  }
  return vis[t];
}

pii mincost_maxflow(int s,int t,int n)
{
  memset(d1,0,sizeof(d1[0])*n);
  int maxflow = 0;
  int mincost = 0;
  while(dijkstra(s,t,n) < INF)
  {
    while(bfs(s,t,n))
    {
      int tflow = INF;
      int tcost = dis[t];
      for(int x = t;x != s;x = pre[x]) tflow = min(tflow,prearc[x]->c);
      for(int x = t;x != s;x = pre[x])
      {
        prearc[x]->c -= tflow;
        prearc[x]->R->c += tflow;
      }
      mincost += tcost * tflow;
      maxflow += tflow;
    }
    memcpy(d1,dis,sizeof(d1[0])*n);
  }
  return pii(mincost,maxflow);
}
```

2.12 Hopcroft-Karp

```
// 注意刷edges
int Level[MaxN], Queue[MaxN];
int LRPair[MaxN], Vis[MaxN], RLPair[MaxN];
int visidx = 0;
vector<int> edges[MaxN];

int dfs(int u) {
  Vis[u] = visidx;
  for (vector<int> :: iterator it = edges[u].begin(); it != edges[u].end();
    ++it) {
    int v = *it;
    int w = RLPair[v];
    if(w == -1 || (Vis[w] != visidx && Level[u] < Level[w] && dfs(w))) {
      LRPair[u] = v;
      RLPair[v] = u;
      return true;
    }
  }
  return false;
}

int hopcroftKarp(int n, int m) {
  memset(LRPair,-1,sizeof(LRPair[0])*(n+10));
  memset(RLPair,-1,sizeof(RLPair[0])*(m+10));
  for(int match = 0;;) {
    int qf = 0;
    int qe = 0;
    memset(Level,-1,sizeof(Level[0])*(n+10));
    for(int i = 1;i <= n;i++) {
      if(LRPair[i] == -1) {
        Level[i] = 0;
        Queue[qe++] = i;
      }
    }
    while(qf < qe) {
      int u = Queue[qf++];

      for (vector<int> :: iterator it = edges[u].begin(); it != edges[u].end
        (); ++it) {
```

```
        int v = *it;
        int rev = RLPair[v];
        if(rev != -1 && Level[rev] < 0) {
          Level[rev] = Level[u] + 1;
          Queue[qe++] = rev;
        }
      }
    }
    visidx++;
    int d = 0;
    for(int i = 1;i <= n;i++) if(LRPair[i] == -1 && dfs(i)) d++;
    if(d == 0) return match;
    match += d;
  }
  return -1;
}
```

2.13 Edmonds matching algorithm

一般图最大匹配模板
g[i][j] 存放邻接矩阵：i,j 是否有边,match[i] 存放 i 所匹配的点
调用 run(N) 返回最大匹配，N 为节点数.

```
const int MAXN = 55;
    queue<int> Q;
    bool g[MAXN][MAXN],inque[MAXN],inblossom[MAXN];
    int match[MAXN],pre[MAXN],base[MAXN];

    //公共祖先
    int findancestor(int u,int v){
      bool inpath[MAXN]={false};
      while(1){
        u=base[u];
        inpath[u]=true;
        if(match[u]==-1)break;
        u=pre[match[u]];
      }
      while(1){
        v=base[v];
        if(inpath[v])return v;
        v=pre[match[v]];
      }
    }

//压缩花
void reset(int u,int anc){
  while(u!=anc){
    int v=match[u];
    inblossom[base[u]]=1;
    inblossom[base[v]]=1;
    v=pre[v];
    if(base[v]!=anc)pre[v]=match[u];
    u=v;
  }
}

void contract(int u,int v,int n){
  int anc=findancestor(u,v);
  //SET(inblossom,0);
  memset(inblossom,0,sizeof(inblossom));
  reset(u,anc);reset(v,anc);
  if(base[u]!=anc)pre[u]=v;
  if(base[v]!=anc)pre[v]=u;
  for(int i=1;i<=n;i++)
    if(inblossom[base[i]]){
      base[i]=anc;
      if(!inque[i]){
        Q.push(i);
        inque[i]=1;
      }
    }
}

bool dfs(int S,int n){
  for(int i=0;i<=n;i++)pre[i]=-1,inque[i]=0,base[i]=i;
  inque[S]=1;
  while(!Q.empty()){
    int u=Q.front();Q.pop();
    for(int v=1;v<=n;v++){
      if(g[u][v]&&base[v]!=base[u]&&match[u]!=v){
        if(v==S||(match[v]!=-1&&pre[match[v]]!=-1))contract(u,v,n);
        else if(pre[v]==-1){
          pre[v]=u;
          if(match[v]!=-1) Q.push(match[v]),inque[match[v]]=1;
          else{
            u=v;
            while(u!=-1){
              v=pre[u];
              int w=match[v];
              match[u]=v;
              match[v]=u;
              u=w;
            }
            return true;
          }
        }
      }
    }
```

```
  }
  return false;
}
int run(int n) {
  int ans = 0;
  memset(match, -1, sizeof(match));
  for (int i = 1; i <= n; ++i) {
    if (match[i] == -1 && dfs(i,n)) ++ans;
  }
  return ans;
}
```

2.14 Planar Gragh

2.14.1 Euler Characteristic

$$\chi = V - E + F$$

其中，V 为点数，E 为边数，F 为面数，对于平面图即为划分成的平面数（包含外平面），χ 为对应的欧拉示性数，对于平面图有 $\chi = C + 1$，C 为连通块个数。

2.14.2 Dual Graph

将原图中所有平面区域作为点，每条边若与两个面相邻则在这两个面之间连一条边，只与一个面相邻连个自环，若有权值 (容量) 保留。

2.14.3 Maxflow on Planar Graph

连接 s 和 t，显然不影响图的平面性，转对偶图，令原图中 s 和 t 连接产生的新平面在对偶图中对应的节点为 s'，外平面对应的顶点为 t'，删除 s' 和 t' 之间直接相连的边。此时 s' 到 t' 的一条最短路就对应了原图上 s 到 t 的一个最大流。

2.15 Prufer Code

2.15.1 根据树构造

我们通过不断地删除顶点编过号的树上的叶子节点直到还剩下 2 个点为止的方法来构造这棵树的 Prüfer sequence。特别的，考虑一个顶点编过号的树 T，点集为 $1, 2, 3, \ldots, n$。在第 i 步中，删除树中编号值最小的叶子节点，设置 Prüfer sequence 的第 i 个元素为与这个叶子节点相连的点的编号。

2.15.2 还原

设 a_i 是一个 Prüfer sequence。这棵树将有 $n+2$ 个节点，编号从 1 到 $n+2$, 对于每个节点，计它在 Prüfer sequence 中出现的次数 +1 为其度数。然后，对于 a 中的每个数 a_i, 找编号最小的度数值为 1 节点 j，加入边 (j, a_i) , 然后将 j 和 a_i 的度数值减少 1。最后剩下两个点的度数值为 1，连起来即可。

2.15.3 一些结论

完全图 K_n 的生成树，顶点的度数必须为 $d_1, d_2, \ldots, d_n$，这样的生成树棵数为：

$$\frac{(n-2)!}{[(d_1-1)!(d_2-1)!(d_3-1)!\ldots(d_n-1)!]}$$

一个顶点编号过的树，实际上是编号的完全图的一棵生成树。通过修改枚举 Prüfer sequence 的方法，可以用类似的方法计算完全二分图的生成树棵数。如果 G 是完全二分图，一边有 n_1 个点，另一边有 n_2 个点，则其生成树棵数为 $n_1^{n_2-1} * n_2^{n_1-1}$。

2.16 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); // n-1 is source
calc_dominator_tree(ncnt);
```

2.17 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;
}
```

2.18 Spanning Tree Count

对于 n 个点的无向图的生成树计数，令矩阵 D 为图 G 的度数矩阵，即 $D = diag(deg_1, deg_2, \ldots, deg_n)$，$A$ 为 G 的邻接矩阵表示，则 $D - A$ 的任意一个 $n-1$ 阶主子式的行列式的值即为答案。

2.19 Chordal Graph

一些结论：

弦：连接环中不相邻的两个点的边。

弦图：一个无向图称为弦图当且仅当图中任意长度大于 3 的环都至少有一个弦。

单纯点：设 N(v) 表示与点 v 相邻的点集。一个点称为单纯点当 v + N(v) 的诱导子图为一个团。

完美消除序列：这是一个序列 v[i]，它满足 v[i] 在 v[i..n] 的诱导子图中为单纯点。

弦图的判定：存在完美消除序列的图为弦图。可以用 MCS 最大势算法求出完美消除序列。

最大势算法从 n 到 1 的顺序依次给点标号 (标号为 i 的点出现在完美消除序列的第 i 个)。设 label[i] 表示第 i 个点与多少个已标号的点相邻，每次选择 label[i] 最大的未标号的点进行标号。

判断一个序列是否为完美消除序列：设 vi+1,⋯,vn 中所有与 vi 相邻的点依次为 vj1,⋯, vjk。只需判断 vj1 是否与 vj2,⋯, vjk 相邻即可。弦图的最大点独立集——完美消除序列从前往后能选就选。最小团覆盖数 = 最大点独立集数。

```
int Label[10010] = {0};
int Order[10010] = {0};
int Seq[10010] = {0};
int Color[10010] = {0};
int Useable[10010] = {0};

int main(void)
{
  int N = 0;
  int M = 0;
  scanf("%d %d",&N,&M);
  for(int i = 0;i < M;i++)
  {
    int x = 0;
    int y = 0;
    scanf("%d %d",&x,&y);
    insert_edge(x,y);
    insert_edge(y,x);
  }

  Label[0] = -5555;
  for(int i = N;i > 0;i--)
  {
    int t = 0;
    for(int j = 1;j <= N;j++)
    {
      if(!Order[j] && Label[j] > Label[t]) t = j;
    }
    Order[t] = i;
    Seq[i] = t;
    for(EDGE* e = E[t];e != NULL;e = e->Next)
    {
      int y = e->y;
      Label[y]++;
    }
  }

  int ans = 0;
  for(int i = N;i > 0;i--)
  {
    for(EDGE* e = E[Seq[i]];e != NULL;e = e->Next)
    {
      int y = e->y;
      Useable[Color[y]] = i;
    }
    int c = 1;
    while(Useable[c] == i) c++;
    Color[Seq[i]] = c;
    if(c > ans) ans = c;
  }
  printf("%d\n",ans);
  while(getchar() != EOF);
  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 MinimumRepresentation

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 >= 0 && i+j+1 < len && Text[(i-j)>>1] == Text[(i+j+1)>>1]) 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 Formula

4.1.1 三角形内心

$$\frac{a\vec{A}+b\vec{B}+c\vec{C}}{a+b+c}$$

4.1.2 三角形外心

$$\frac{\vec{A}+\vec{B}-\frac{\overrightarrow{BC}\cdot\overrightarrow{CA}}{\overrightarrow{AB}\times\overrightarrow{BC}}\overrightarrow{AB}^T}{2}$$

4.1.3 三角形外接圆半径

$$R=\frac{abc}{4S}$$

4.1.4 Pick's theorem

$$S=\frac{B}{2}+I-1$$

4.1.5 超球坐标系

$$\begin{aligned}
x_1 &= r\cos(\phi_1) \\
x_2 &= r\sin(\phi_1)\cos(\phi_2) \\
x_3 &= r\sin(\phi_1)\sin(\phi_2)\cos(\phi_3) \\
&\dots \\
x_{n-1} &= r\sin(\phi_1)\cdots\sin(\phi_{n-2})\cos(\phi_{n-1}) \\
x_n &= r\sin(\phi_1)\cdots\sin(\phi_{n-2})\sin(\phi_{n-1}) \\
\phi_{n-1} &= 0..2*\pi \\
\forall i = 1..n-1 \phi_i &= 0..\pi
\end{aligned}$$

4.1.6 三维旋转公式

绕着 $(0,0,0)-(ux,uy,uz)$ 旋转 θ, (ux,uy,uz) 是单位向量

$$R = \begin{bmatrix} \cos\theta + u_x^2(1-\cos\theta) & u_xu_y(1-\cos\theta) - u_z\sin\theta & u_xu_z(1-\cos\theta) + u_y\sin\theta \\ u_yu_x(1-\cos\theta) + u_z\sin\theta & \cos\theta + u_y^2(1-\cos\theta) & u_yu_z(1-\cos\theta) - u_x\sin\theta \\ u_zu_x(1-\cos\theta) - u_y\sin\theta & u_zu_y(1-\cos\theta) + u_x\sin\theta & \cos\theta + u_z^2(1-\cos\theta) \end{bmatrix}.$$

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = R \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

4.2 2D-Geometry

```
struct Point {
  double x, y;
  Point (){}
  Point(double x, double y) : x(x), y(y) {}
  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 &b) { return Point(x * b, y * b); }
  Point operator / (const double &b) { return Point(x / b, y / b); }
  Point rot90(int t) { return Point(−y, x) * t; }
  Point rot(double ang) { return Point(x * cos(ang) − y * sin(ang), x * sin(
    ang) + y * cos(ang)); }
  double ang() { double res = atan2(y, x); if (dcmp(res) < 0) res += pi * 2;
    return res; }
  double operator * (const Point &b) { return x * b.y − y * b.x; }
  double operator % (const Point &b) { return x * b.x + y * b.y; }
  double len2() { return x * x + y * y; }
  double len() { return sqrt(x * x + y * y); }
};
struct Line {
  Point s, e;
  Line(){}
  Line(Point s, Point e) : s(s), e(e) {}
};
inline double xmul(Point a, Point b, Point c) {
  return (b − a) * (c − a);
}
//点p 到直线{p1, p2} 距离
double disLP(Point p1, Point p2, Point q) {
  return fabs((p1 − q) * (p2 − q)) / (p1 − p2).len();
}
// 平面几何
// 点q 到线段{p1, p2} 的距离
double dis_Seg_P(Point p1, Point p2, Point q) {
  if ((p2 − p1) % (q − p1) < eps) return (q − p1).len();
  if ((p1 − p2) % (q − p2) < eps) return (q − p2).len();
  return disLP(p1, p2, q);
}
// hit on the edge will return true
bool is_segment_intersect(Point A, Point B, Point C,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(dcmp((B−A)*(C−A))*dcmp((B−A)*(D−A)) > 0) return false;
  if(dcmp((D−C)*(A−C))*dcmp((D−C)*(B−C)) > 0) return false;
  return true;
}
//两直线交点
Point get_intersect(Line s1, Line 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;
}
// 点P 是否在线段 {p1, p2} 上
bool is_point_onseg(Point p1,Point p2,Point P)
{
  if(! (min(p1.x,p2.x) <= P.x && P.x <= max(p1.x,p2.x) &&
      min(p1.y,p2.y) <= P.y && P.y <= max(p1.y,p2.y)) )
    return false;
  if(dcmp((P−p1)*(p2−p1)) == 0) return true;
  return false;
}
// 点q 到直线 {p1, p2} 垂足
Point proj(Point p1, Point p2, Point q) {
  return p1 + ((p2 − p1) * ((p2 − p1) % (q − p1) / (p2 − p1).len()));
}
// 直线与圆的交点
```

```
vector<Point> getCL(Point c, double r, Point p1, Point p2) {
  vector<Point> res;
  double x = (p1 - c) % (p2 - p1);
  double y = (p2 - p1).len2();
  double d = x * x - y * ((p1 - c).len2() - r * r);
  if (d < -eps) return res;
  if (d < 0) d = 0;
  Point q1 = p1 - ((p2 - p1) * (x / y));
  Point q2 = (p2 - p1) * (sqrt(d) / y);
  res.push_back(q1 - q2);
  res.push_back(q1 + q2);
  return res;
}
// 圆与圆的交点
vector<Point> getCC(Point c1, double r1, Point c2, double r2) {
  vector<Point> res;
  double x = (c1 - c2).len2();
  double y = ((r1 * r1 - r2 * r2) / x + 1) / 2;
  double d = r1 * r1 / x - y * y;
  if (d < -eps) return res;
  if (d < 0) d = 0;
  Point q1 = c1 + (c2 - c1) * y;
  Point q2 = ((c2 - c1) * sqrt(d)).rot90();
  res.push_back(q1 - q2);
  res.push_back(q1 + q2);
  return res;
}
// 两圆公共面积.
double areaCC(Point c1, double r1, Point c2, double r2) {
  double d = (c1 - c2).len();
  if (r1 + r2 < d + eps) return 0;
  if (d < fabs(r1 - r2) + eps) {
    double r = min(r1, r2);
    return r * r * pi;
  }
  double x = (d * d + r1 * r1 - r2 * r2) / (2 * d);
  double t1 = acos(x / r1);
  double t2 = acos((d - x) / r2);
  return r1 * r1 * t1 + r2 * r2 * t2 - d * r1 * sin(t1);
}
// ccenter 返回{p1, p2, p3} 的外接圆圆心, formula
// 四点在同一圆周
bool onCir(Point p1, Point p2, Point p3, Point p4) {
  if (fabs((p2 - p1) * (p3 - p1)) < eps) return true;
  Point c = ccenter(p1, p2, p3);
  return fabs((c - p1).len2() - (c - p4).len2()) < eps;
}
//两圆公切线, 先返回内公切线, 后面是外公切线
vector<Line> getLineCC(Point c1, double r1, Point c2, double r2) {
  vector<Line> res;
  double d = (c1 - c2).len();
  if (fabs(d - r1 - r2) < eps) {
    Point o = (c1 + c2) * 0.5;
    res.push_back(Line(o, o + (c1 - c2).rot90()));
    res.push_back(res[res.size() - 1]);
  } else {
    double ang = acos((r1 + r2) / d);
    res.push_back(Line(c1 + ((c2 - c1) * (r1 / d)).rot(ang), c2 + ((c1 - c2)
      * (r2 / d)).rot(ang)));
    ang = -ang;
    res.push_back(Line(c1 + ((c2 - c1) * (r1 / d)).rot(ang), c2 + ((c1 - c2)
      * (r2 / d)).rot(ang)));
  }
  double ang = acos((r2 - r1) / d);
  res.push_back(Line(c1 + ((c1 - c2) * (r1 / d)).rot(ang), c2 + ((c1 - c2).
    rot(ang) * (r2 / d))));
  ang = -ang;
  res.push_back(Line(c1 + ((c1 - c2) * (r1 / d)).rot(ang), c2 + ((c1 - c2).
    rot(ang) * (r2 / d))));
  return res;
}
//圆 [(0,0), r] 与三角形 (0, p1, p2) 求公共面积
double rad(Point p1, Point p2) {
  double res = p2.ang() - p1.ang();
  if (res > pi - eps) res -= 2.0 * pi;
  else if (res + eps < -pi) res += 2.0 * pi;
  return res;
}
double areaCT(double r, Point p1, Point p2) {
  vector<Point> qs = getCL(Point(0,0), r, p1, p2);
  if (qs.size() == 0) return r * r * rad(p1, p2) / 2;
  bool b1 = p1.len() > r + eps, b2 = p2.len() > r + eps;
  if (b1 && b2) {
    if ((p1 - qs[0]) % (p2 - qs[0]) < eps &&
      (p1 - qs[1]) % (p2 - qs[1]) < eps)
    return (r * r * (rad(p1, p2) - rad(qs[0], qs[1])) + qs[0] * qs[1]) / 2;
    else return r * r * rad(p1, p2) / 2;
  } else if (b1) return (r * r * rad(p1, qs[0]) + qs[0] * p2) / 2;
  else if (b2) return (r * r * rad(qs[1], p2) + p1 * qs[1]) / 2;
  else return p1 * p2 / 2;
}
```

4.3 3D-Geometry

```
// 空间几何
struct Point {
  double x, y, z;
  Point(){}
```

```
  Point(double x, double y, double z) : x(x), y(y), z(z) {}
  Point operator + (const Point &b) { return Point(x + b.x, y + b.y, z + b.z); }
  Point operator - (const Point &b) { return Point(x - b.x, y - b.y, z - b.z); }
  Point operator * (const Point &b) { return Point(y * b.z - z * b.y, z * b.x - x * b.z, x * b.y - y * b.x); }
  Point operator * (const double &b) { return Point(x * b, y * b, z * b); }
  double operator % (const Point &b) { return x * b.x + y * b.y + z * b.z; }
  double len2() { return x * x + y * y + z * z; }
  double len() { return sqrt(x * x + y * y + z * z); }

};
// 返回直线{p1, p2} 上到{q1, q2} 的最近点
// 平行时 d = 0
Point getLL(Point p1, Point p2, Point q1, Point q2) {
  Point p = q1 - p1;
  Point u = p2 - p1;
  Point v = q2 - q1;
  //len2 means len^2
  double d = u.len2() * v.len2() - (u % v) * (u % v);
  //if (abs(d) < eps) return NULL;
  double s = ((p % u) * v.len2() - (p % v) * (u % v)) / d;
  return p1 + u * s;
}
// 面与线的交点, d = 0 时线在面上或与面平行
// p 为面上某点, o 是平面法向量. {q1, q2} 是直线.
Point getPL(Point p, Point o, Point q1, Point q2) {
  double a = o % (q2 - p);
  double b = o % (q1 - p);
  double d = a - b;
  //if (abs(d) < eps) return NULL;
  return ((q1 * a) - (q2 * b)) * (1. / d);
}
// 平面与平面的交线
vector<Point> getFF(Point p1, Point o1, Point p2, Point o2) {
  vector<Point> res;
  Point e = o1 * o2;
  Point v = o1 * e;
  double d = o2 % v;
  if (fabs(d) < eps) return res;
  Point q = p1 + v * ((o2 % (p1 - p1)) / d);
  res.push_back(q);
  res.push_back(q + e);
  return res;
}
// 射线p1, p2 与球(c,r) 的p 与p1 的距离. 不相交返回-1.
double get(Point c, double r, Point p1, Point p2) {
  if ((p2 - p1) % (c - p1) < -eps) return -1.;
  Point v = (p2 - p1); v = v * (1 / v.len());
  double x = (c - p1) % v;
  v = p1 + v * x;
  double d = (v - c).len2();
  if (dcmp(d - r * r) >= 0) return -1.;
  d = (p1 - v).len() - sqrt(r * r - d);
  return d;
}
```

4.4 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 && dcmp((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 && dcmp((P[i]-Hull[HTop-2])*(Hull[HTop-1]-Hull[HTop-2])
      ) >= 0) HTop--;
    if(i) Hull[HTop++] = P[i];
  }
  return HTop;
}
```

4.5 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;
}
```

4.6 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;
}
```

4.7 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];  //点到点是属于哪个面ij
  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，如果点在凸包外更新凸包p
    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;
```

4.8 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,(Hull[i]-Hull[to]).len2());
    ans = max(ans,(Hull[i+1]-Hull[to]).len2());
  }
  return sqrt(ans);
}
```

4.9 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);
    }
} ;
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, Point hull[]) {
  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;
  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 < tail; ++i) hull[cnt++] = get_intersect(deq[i], deq[
    i+1]);
  return cnt;
}
```

4.10 Simpson

如果怀疑被畸形数据了并且时间很多，试试

$$\int_a^b f(x)\,dx \approx \frac{(b-a)}{8}\left[f(a)+3f\left(\frac{2a+b}{3}\right)+3f\left(\frac{a+2b}{3}\right)+f(b)\right]$$

```
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);
}
```

4.11 Circle Union

圆的面积并. $area_i$ 表示恰好被覆盖 i 次的面积。普通面积并需要去除掉被包含或被内切的圆。getCC() 模板里有。eps 设成 $1e-8$ 需要对圆去重.

```
double cal(Point c, double r, double ang1, double ang2) {
  double ang = ang2 - ang1;
  if (dcmp(ang) == 0) return 0;
  Point p1 = c + Point(r, 0).rot(ang1);
  Point p2 = c + Point(r, 0).rot(ang2);
  return r * r * (ang - sin(ang)) + p1 * p2;
}
bool rm[MaxN];
pair<double, int> keys[MaxN * 10];
vector<Point> getCC(){}
bool cmp(const pair<double,int> &a, const pair<double,int> &b) {
  if (dcmp(a.fi - b.fi) != 0) return dcmp(a.fi - b.fi) < 0;
  return a.se > b.se;
}
double solve(int cur, int n) {
  if (rm[cur]) return 0;
  int m = 0;
  for (int i = 0; i < n; ++i) if (i != cur && !rm[i]) {
    // if (cir[cur] 被 cir[i] 包含或内切) { ++cover[cur]; continue; }
    vector<Point> root = getCC(cir[cur].c, cir[cur].r, cir[i].c, cir[i].r);
    if (root.size() == 0) continue;
    double ang1 = (root[0] - cir[cur].c).ang();
    double ang2 = (root[1] - cir[cur].c).ang();
    if (dcmp(ang1 - ang2) == 0) continue;
    if (dcmp(ang1 - ang2) >= 0) {
      keys[m++] = make_pair(0, 1);
      keys[m++] = make_pair(ang2, -1);
      keys[m++] = make_pair(ang1, 1);
      keys[m++] = make_pair(2*pi, -1);
    } else {
      keys[m++] = make_pair(ang1, 1);
      keys[m++] = make_pair(ang2, -1);
    }
  }
  keys[m++] = make_pair(0, 0);
  keys[m++] = make_pair(2 * pi, - 100000);
  sort(keys, keys + m, cmp);
  double res = cal(cir[cur].c, cir[cur].r, 0, keys[0].first);
  int cnt = 0;
  for (int i = 0; i < m; ++i) {
    cnt += keys[i].second;
    if (cnt == 0) res += cal(cir[cur].c, cir[cur].r, keys[i].first, keys[i
      +1].first);
    // area[cover[cur] + cnt] -= tarea;
    // area[cover[cur] + cnt + 1] += tarea;
  }
  return res;
}
```

4.12 Polygon Union

多边形面积并 $O(n^2logn)$

```
struct Polygon {
  int M;
  vector<Point> p;
  void init() {
    p.resize(M);
    for (int i = 0; i < M; ++i) p[i].init();
    if (dcmp((p[1]-p[0]) * (p[2]-p[1])) < 0) reverse(p.begin(), p.end());
    p.push_back(p[0]);
  }
```

```
  Point& operator [](const int &i) { return p[i]; }
} poly[MaxN];
double xmul(Point a, Point b, Point c) { return (b-a)*(c-a); }
pair<double, int> keys[MaxN];
double get(Point a, Point b, Point c) {
  double t;
  if (fabs(a.x-b.x) > eps) t = (c.x-a.x)/(b.x-a.x);
  else t = (c.y-a.y)/(b.y-a.y);
  t = max(min(t,1.0),0.0);
  return t;
}
double solve(int n) {
  double res = 0;
  for (int i = 0; i < n; ++i)
  for (int x = 0; x < poly[i].M; ++x) {
    int keysize = 0;
    for (int k = 0; k < n; ++k) if (k != i)
    for (int y = 0; y < poly[k].M; ++y) {
      int t1 = dcmp(xmul(poly[i][x], poly[i][x+1], poly[k][y]));
      int t2 = dcmp(xmul(poly[i][x], poly[i][x+1], poly[k][y+1]));
      if (!t1 && !t2) {
        if (k < i && dcmp((poly[k][y+1]-poly[k][y])%(poly[i][x+1]-poly[i][x])
          ) >= 0) {
          double d1 = get(poly[i][x], poly[i][x+1], poly[k][y]);
          double d2 = get(poly[i][x], poly[i][x+1], poly[k][y+1]);
          keys[keysize++] = make_pair(d1, 1);
          keys[keysize++] = make_pair(d2, -1);
        }
      } else if ((t1 >= 0 && t2 < 0) || (t1 < 0 && t2 >= 0)) {
        double d1 = xmul(poly[k][y], poly[k][y+1], poly[i][x]);
        double d2 = xmul(poly[k][y], poly[k][y+1], poly[i][x+1]);
        int t = 1; if (t2 >= 0) t = -1;
        keys[keysize++] = make_pair(max(min(d1/(d1-d2),1.),0.), t);
      }
    }
    sort(keys, keys + keysize);
    int cnt = 0;
    double s = 0, tmp = 0;
    bool f = 1;
    for (int j = 0; j < keysize; ++j) {
      cnt += keys[j].second;
      if (!cnt && !f) tmp = keys[j].first, f = 1;
      if (cnt && f) s += keys[j].first - tmp, f = 0;
    }
    s += 1. - tmp;
    res += (poly[i][x] * poly[i][x+1]) * s;
  }
  return res * 0.5;
}
```

5 Math

5.1 Formula

5.1.1 Catalan number

$$C_n = \frac{1}{n+1}\binom{2n}{n}$$

$$C_{(n,m)} = \binom{n+m}{m} - \binom{n+m}{m-1}$$

5.1.2 勾股数

n, m 互质且 m, n 至少有一个偶数则是苏勾股数

$$a = m^2 - n^2$$

$$b = 2mn$$

$$c = m^2 + n^2$$

5.1.3 容斥原理

$$g(A) = \sum_{S\,:\,S\subseteq A} f(S)$$

$$f(A) = \sum_{S\,:\,S\subseteq A} (-1)^{|A|-|S|} g(S)$$

5.1.4 Bell Number

$$Bell_{n+1} = \sum_{k=0}^{n} \binom{n}{k} Bell[k]$$

$$Bell_{p^m+n} \equiv mBell_n + Bell_{n+1} \pmod{P}$$

$$Bell_n = \sum_{k=1}^{n} S(n,k)$$

5.1.5 第一类 Stirling 数

n 个元素的项目分作 k 个环排列的方法数目

$$s(n+1,k) = s(n,k-1) + n\,s(n,k)$$

5.1.6 第二类 Stirling 数

第二类 Stirling 数是 n 个元素的集定义 k 个等价类的方法数目

$$S(n,k) = S(n-1,k-1) + kS(n-1,k)$$

$$S(n,k) = \frac{1}{k!}\sum_{j=1}^{k}(-1)^{k-j}\binom{k}{j}j^n$$

5.1.7 判断是否为二次剩余

若是 d 是 p 的二次剩余 $d^{\frac{p-1}{2}} \equiv 1 \mod p$ 否则 $d^{\frac{p-1}{2}} \equiv -1 \mod p$

5.1.8 级数展开式

$$
\begin{aligned}
e^x &= \sum_{n=0}^{\infty} \frac{x^n}{n!} \\
log(1+x) &= \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n} x^n (|x| < 1) \\
\sin x &= \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} \\
\cos x &= \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n} \\
\arcsin x &= \sum_{n=0}^{\infty} \frac{(2n)!}{4^n (n!)^2 (2n+1)} x^{2n+1}
\end{aligned}
$$

5.2 Factorial-Mod

素数 p, $n! = ap^k$ 时, 返回 $a(mod \quad p)$

```
int modFact(long long n, int p) {
  int res = 1;
  while (n > 0) {
    for (int i = 1, m = n % p; i <= m; ++i) res = (long long) res * i % p;
    if ((n /= p) % 2 > 0) res = p - res;
  }
  return res;
}
```

5.3 NTT

最终结果 $\bmod P$.
$E_i \equiv g^{(P_i-1)\div N_1} \pmod{P_i}$ $F_i \equiv 1 \div E_i \pmod{P_i}$ $I_i \equiv 1 \div N_1 \pmod{P_i}$

```
namespace NTT {
  const int P = MOD;
  const int P1 = 998244353;
  const int P2 = 995622913;
  const int g1 = 3;
  const int g2 = 5;
  const LL M1 = 397550359381069386LL;
  const LL M2 = 596324591238590904LL;
  const LL MM = 993874950619660289LL;
  int I1, I2;

  int N;
  int a[MaxN], b[MaxN], c[MaxN];
  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 g) {
    if (g < 0) g = fpow(-g, pm - 2, pm);
    int pw = fpow(g, (pm - 1) / N, pm);
    for (int m = N, 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 < N; 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 < N - 1; ++j) {
      for (int k = N / 2; k > (i^=k); k /= 2);
      if (j < i) swap(A[i], A[j]);
    }
  }
  void solve(int *A, int *B, int *C, int n) {

    N = 1;
    while (N < (n << 1)) N <<= 1;
    memset(C, 0, sizeof (*C)*N);
    for (int i = n; i < N; ++i) A[i] = B[i] = 0;
    memcpy(a, A, sizeof (*A)*N);
    memcpy(b, B, sizeof (*B)*N);

    NTT(a, P1, g1);
    NTT(b, P1, g1);
    for (int i = 0; i < N; ++i) c[i] = (LL) a[i] * b[i] % P1;
    NTT(c, P1, -g1);

    NTT(A, P2, g2);
    NTT(B, P2, g2);
    for (int i = 0; i < N; ++i) C[i] = (LL) A[i] * B[i] % P2;
    NTT(C, P2, -g2);

    I1 = fpow(N, P1 - 2, P1);
    I2 = fpow(N, P2 - 2, P2);
    for (int i = 0; i < n; ++i) {
      C[i] = crt((LL) c[i] * I1 % P1, (LL) C[i] * I2 % P2);
    }
  }
}
```

5.4 DFT

n 需要为 2 的次幂，sign 传入 1 时正变换，-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;
}
```

5.5 Baby-Step-Giant-Step

```
namespace BSGS {
  #define MaxNode 1200007
  #define HMD 1000007
  struct Thash{
    PII has[MaxNode];int next[MaxNode],h[HMD],tot;
    void clr(){
      memset(h, 0, sizeof h);
    }
    void ins(int p,int pos){
      int vex = p % HMD;
      for(int z = h[vex]; z; z = next[z]) if(has[z].fi == p) return;
      has[++tot] = MP(p,pos); next[tot] = h[vex]; h[vex] = tot;
    }
    int  find(int p){
      if (p == 0) return -1;
      for(int z = h[p % HMD]; z; z = next[z]) if(has[z].fi == p) return has[z
        ].se;
      return -1;
    }
  }Hash;
  void build(int y, int p){
    int m = 700000, now = 1;
    for (int i = 0; i <= m; ++i) {
      Hash.ins(now,i);
      now = (LL) now * y % p;
    }
  }
  int find(int y, int z, int p)
  {
    int D = fpow(y, m, p), now = 1;
    D = fpow(D, p - 2, p);
    for (int i = 0; i <= p / m + 1; ++i) {
      int t = Hash.find((long long)z * now % p);
      if (t + 1) return i * m + t;
      now=(long long) now * D % p;
    }
    return -1;
  }
};
```

5.6 CRT

lcm 是 mod 的那个数，FACTOR 是因子。只保证对于 Square-Free Number 的正确性。

```
int ans = 0;
int lcm = 999911658;
for(int i = 0;i < 4;i++)
{
    int t = lcm/FACTOR[i];

    int x = 0;
    int y = 0;
    int d = 0;
    exgcd(t,FACTOR[i],d,x,y);
    x = (ll)(x%lcm+lcm)%lcm*t*R[i]%lcm;
    ans = (ans+x)%lcm;
}
```

```
int Lucas(int n,int m,int mod)
{
    int md = FACTOR[mod];
    int ans = 1;
    while(n || m)
    {
        ans = ans*_C(n%md,m%md,mod)%md;
        n /= md;
        m /= md;
    }
    return ans;
}
```

5.7 Find-Square-Root

Tonelli–Shanks algorithm
Find such x that $x^2 \equiv n \pmod{p}$

```
int find_root(int n, int p) {
  n %= p;
  if (n == 0) return 0;
  if (fpow(n, (p - 1) / 2, p) != 1) return -1;
  int Q = p - 1, S = 0;
  for (; Q % 2 == 0; Q >>= 1) ++S;
  if (S == 1) return fpow(n, (p + 1) / 4, p);
  int z;
  while (1) {
    z = 1 + rand() % (p - 1);
    if (fpow(z, (p - 1) / 2, p) != 1) break;
  }
  int c = fpow(z, Q, p);
  int R = fpow(n, (Q + 1) / 2, p);
  int t = fpow(n, Q, p);
  int m = S;
  while (1) {
    if (t % p == 1) break;
    int i = 1;
    for (i = 1; i < m; ++i) if (fpow(t, 1 << i, p) == 1) break;
    int b = fpow(c, 1 << (m - i - 1), p);
    R = (LL) R * b % p;
    t = (LL) t * b % p * b % p;
    c = (LL) b * b % p;
    m = i;
  }
  return (R % p + p) % p;
}
```

5.8 Integer-Partition

整数划分。五边形数 $\frac{3j^2-j}{2}$

Generate function $\prod_{n=1}^{\infty} 1+x^n+x^{2n}+x^{3n}+\cdots = \prod_{n=1}^{\infty} \frac{1}{1-x^n}$

$\prod_{n=1}^{\infty}(1-x^n) = \sum_{k=-\infty}^{\infty}(-1)^k x^{k(3k-1)/2} = \sum_{k=0}^{\infty}(-1)^k x^{k(3k\pm1)/2}$

```
void parition(int n) {
    dp[0] = 1;
    for (int i = 1; i <= n; ++i) {
        for (int j = 1, r = 1; i - (3*j*j-j) / 2 >= 0; ++j, r *= -1) {
            dp[i] = (dp[i] + dp[i - (3*j*j-j) / 2] * r) % MOD;
            if (i - (3*j*j + j) / 2 >= 0)
                dp[i] = (dp[i] + dp[i - (3*j*j+j) / 2] * r) % MOD;
        }
    }
}
```

5.9 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;
}
```

5.10 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;
}
```

5.11 Pollard-Rho

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

    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;
}
```

5.12 Simplex

returns 1 if feasible, 0 if not feasible, -1 if unbounded
solutions in b
returns $max(cx|Ax <= b)$
$c = A_{m,i}$
$b = A_{i,n}$
$A_{m,n} = 0$

```
void pivot(int m, int n, double a[][MaxN], int B[], int N[], int r, int c) {
  swap(N[c], B[r]);
  a[r][c]=1/a[r][c];
  for (int j=0; j<=n; j++)if (j!=c) a[r][j]*=a[r][c];
  for (int i=0; i<=m; i++)if (i!=r) {
    for (int j=0; j<=n; j++)if (j!=c)
      a[i][j]-=a[i][c]*a[r][j];
```

```
    a[i][c] = -a[i][c]*a[r][c];
  }
}
int feasible(int m, int n, double a[][MaxN], int B[], int N[]) {
  int r, c; double v;
  while (1) {
    double p = 1e100;
    for (int i=0; i<m; i++) if (a[i][n]<p) p=a[r=i][n];
    if (p>-eps) return 1;
    p = 0;
    for (int i=0; i<n; i++) if (a[r][i]<p) p=a[r][c=i];
    if (p>-eps) return 0;
    p = a[r][n]/a[r][c];
    for (int i=r+1; i<m; i++) if (a[i][c]>eps) {
      v = a[i][n]/a[i][c];
      if (v<p) r=i, p=v;
    }
    pivot(m, n, a, B, N, r, c);
  }
}
int B[10], N[MaxN];
int simplex(int m, int n, double a[][MaxN], double b[], double& ret) {
  int r, c; double v;
  for (int i=0; i<n; i++) N[i]=i;
  for (int i=0; i<m; i++) B[i]=n+i;
  if (!feasible(m, n, a, B, N)) return 0;
  while (1) {
    double p = 0;
    for (int i=0; i<n; i++) if (a[m][i]>p)
      p=a[m][c=i];
    if (p<eps) {
      for (int i=0; i<n; i++) if (N[i]<n)
        b[N[i]]=0;
      for (int i=0; i<m; i++) if (B[i]<n)
        b[B[i]]=a[i][n];
      ret = -a[m][n];
      return 1;
    }
    p = 1e100;
    for (int i=0; i<m; i++) if (a[i][c]>eps) {
      v = a[i][n]/a[i][c];
      if (v<p) p=v, r=i;
    }
    if (p > 1e90) return -1;
    pivot(m, n, a, B, N, r, c);
  }
}
```

6 Others

6.1 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->D->U = j;
```

```
      j->C->S++;
    }
  }

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

bool dfs(int k)
{
  if(H.R == &H)
  {
    // found! add custom handler here.
    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; // useless
  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.R = H.U = H.D = &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->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;
}
```

6.2 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. .....
}
```

6.3 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。

6.4 Point-Related Tree DC

树分治的大体框架，没整理成模板，起个 Hint 作用吧。

```
bool disabled[222222];

// init mintree with MAXINT plz...
int mintree = 0;
int cog = -1;
int allSize = 0;
int TreeSize[222222];
int findcog(int x,int fa)
{
  TreeSize[x] = 1;

  int cur = 0;
  for(EDGE* e = E[x];e;e = e->Next)
  {
    int y = e->y;
    if(y == fa || disabled[y]) continue;
    findcog(y,x);
    TreeSize[x] += TreeSize[y];
    cur = max(cur,TreeSize[y]);
  }
  cur = max(cur,allSize-TreeSize[x]);
  if(cur < mintree)
  {
    mintree = cur;
    cog = x;
  }
  return 0;
}

int FuckTree(int root,int size)
{
  mintree = 0x7FFFFFFF; cog = -1;
  allSize = size; findcog(root,-1);
  root = cog;

  pcnt = 0; // deal subtree ops here
  for(EDGE* e = E[root];e;e = e->Next)
  {
    int y = e->y;
    int w = e->w;
    if(disabled[y]) continue;

    length[y] = w;
    depth[y] = 1;
    dfs(y,y,root);
  }

  disabled[root] = true;
  for(EDGE* e = E[root];e;e = e->Next)
  {
    int y = e->y;
```

```
    if(disabled[y]) continue;

    FuckTree(y,TreeSize[y]);
  }
  return 0;
}
```

7 外挂

7.1 Evaluate

```
import javax.script.ScriptEngineManager;
import javax.script.ScriptEngine;
public class Main {
  public static void main(String[] args) throws Exception{
    ScriptEngineManager mgr = new ScriptEngineManager();
    ScriptEngine engine = mgr.getEngineByName("JavaScript");
    String foo = "3+4";
    System.out.println(engine.eval(foo));
    }
}
```

7.2 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;
}
```

7.3 pb_ds

使用时注意，对于 G++ 4.7.0 以下版本自带的 libstdc++，需要用 null_mapped_type，以上使用 null_type，注意不支持 multi 系，需要自行塞个 pair。所有 rank 值都是 0 开始的。

复杂度方面，pairing_heap 的合并是 $O(1)$，其他方面 thin_heap 要好一点，注意 thin_heap 合并是 $O(n)$，谨防被坑。

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/priority_queue.hpp>

using namespace __gnu_pbds;

typedef tree<int, null_mapped_type, less<int>,rb_tree_tag,
  tree_order_statistics_node_update> superset;
typedef tree<int, int, less<int>,rb_tree_tag,
  tree_order_statistics_node_update> supermap;
// .order_of_key() 返回某个值的排名，是的版。lower_boundrank
// .find_by_order() 返回一个表示某个特定排名的值。iterator

// 合并，需要保证中所有元素的比中的小。stkeyanother_st
st.join(another_st);
// 拆分严格大于的东西到里。keyoutput_st
st.split(key,output_st);

// 还有tagpairing_heap_tag, binomal_heap_tag, rc_binomal_heap_tag.
typedef priority_queue<int, greater<int>, thin_heap_tag> hyperheap;
// .是修改值的操作，第一个参数是一个。modifyiterator
// 也支持join
```

7.4 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;
}
```

8 临时应对策略

8.1 Circular LCS

```
int n, a[N << 1], b[N << 1];

bool has(int i, int j) {
    return a[(i − 1) % n] == b[(j − 1) % n];
}

const int DELTA[3][2] = {{0, −1}, {−1, −1}, {−1, 0}};

int from[N][N];

int solve() {
    memset(from, 0, sizeof(from));
    int ret = 0;
    for (int i = 1; i <= 2 * n; ++ i) {
        from[i][0] = 2;
        int left = 0, up = 0;
        for (int j = 1; j <= n; ++ j) {
            int upleft = up + 1 + !!from[i − 1][j];
            if (!has(i, j)) {
                upleft = INT_MIN;
            }
            int max = std::max(left, std::max(upleft, up));
            if (left == max) {
                from[i][j] = 0;
            } else if (upleft == max) {
                from[i][j] = 1;
            } else {
                from[i][j] = 2;
            }
            left = max;
        }
        if (i >= n) {
            int count = 0;
            for (int x = i, y = n; y;) {
                int t = from[x][y];
                count += t == 1;
                x += DELTA[t][0];
                y += DELTA[t][1];
            }
            ret = std::max(ret, count);
            int x = i − n + 1;
            from[x][0] = 0;
            int y = 0;
            while (y <= n && from[x][y] == 0) {
                y++;
            }
            for (; x <= i; ++ x) {
                from[x][y] = 0;
                if (x == i) {
                    break;
                }
                for (; y <= n; ++ y) {
                    if (from[x + 1][y] == 2) {
                        break;
                    }
                    if (y + 1 <= n && from[x + 1][y + 1] == 1) {
                        y ++;
                        break;
                    }
                }
            }
        }
    }
    return ret;
}
```

8.2 Lattice Count

计算

$$\sum_{0\le i<n} \lfloor \frac{a+b\cdot i}{m} \rfloor$$

$(n, m > 0, a, b \ge 0)$

```
typedef long long LL;

LL count(LL n, LL a, LL b, LL m) {
    if (b == 0) {
        return n * (a / m);
    }
    if (a >= m) {
        return n * (a / m) + count(n, a % m, b, m);
    }
    if (b >= m) {
        return (n − 1) * n / 2 * (b / m) + count(n, a, b % m, m);
    }
    return count((a + b * n) / m, (a + b * n) % m, m, b);
}
```

8.3 Planer Dual

DYF 的完整代码，临时应对用，需要看一会儿。

```
#include <cmath>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <algorithm>
#include <vector>
#include <set>
```

```
using namespace std;

const double mi = 1e+20;
const double eps = 1e-08;
const double pi = 3.14159265358979;
inline int fi (double a)
{
    if (a > eps) return 1;
    else if (a >= -eps) return 0;
    else return -1;
}
const int mvert = 100010;
const int marea = 300010;
const int medge = 1000010;
double tx[mvert], ty[mvert];
int s, e;
struct edge_rec
{
    int in, out; double angle;
    edge_rec (void) {}
    edge_rec (int in0, int out0, double a0) : in(in0), out(out0), angle(a0) {}
    bool operator < (const edge_rec& a) const { return fi(angle - a.angle) == -1; }
};
double capacity[mvert];
vector<edge_rec> vertex[mvert];
inline int edge1 (int x) { return x << 1; }
inline int edge2 (int x) { return (x << 1) | 1; }
int next_rec[medge << 1], belong[medge << 1]; bool sch[medge << 1];
int areamr;
void find_ring (int x, int ori)
{
    if (next_rec[x] != ori) find_ring(next_rec[x], ori);
    belong[x] = areamr, sch[x] = true;
}
struct { int to, next; double len; } graph[medge]; int grapmr, mv;
void ginit (int m)
{
    grapmr = mv = m;
    for (int i = 0; i < m; i++) graph[i].next = -1;
}
void glink (int a, int b, double len)
{
    int p = grapmr++;
    graph[p].to = b, graph[p].len = len, graph[p].next = graph[a].next;
    graph[a].next = p;
}
struct dij_pair
{
    int vertex; double dist;
    dij_pair (void) {}
    dij_pair (int v0, double d0) : vertex(v0), dist(d0) {}
    bool operator < (const dij_pair& a) const
    {
        int tt = fi(dist - a.dist);
        if (tt) return tt == -1;
        else return vertex < a.vertex;
    }
};
set<dij_pair> heap;
double dist[marea]; bool proc[marea];
set<dij_pair>::iterator mh[marea];
double dijkstra (int source, int terminate)
{
    for (int i = 0; i < mv; i++) dist[i] = mi;
    dist[source] = 0;
    heap.clear();
    for (int i = 0; i < mv; i++) mh[i] = heap.insert(dij_pair(i, dist[i])).first, proc[i] = false;
    proc[source] = true;
    while (!heap.empty())
    {
        int cur = heap.begin()->vertex; heap.erase(heap.begin());
        proc[cur] = true;
        if (cur == terminate) break;
        for (int p = graph[cur].next; p != -1; p = graph[p].next)
        {
            int tar = graph[p].to;
            if (!proc[tar] && fi(dist[tar] - (dist[cur] + graph[p].len)) == 1)
            {
                dist[tar] = dist[cur] + graph[p].len;
                heap.erase(mh[tar]);
                mh[tar] = heap.insert(dij_pair(tar, dist[tar])).first;
            }
        }
    } return dist[terminate];
}
int main ()
{
    int n, m; scanf("%d %d", &n, &m);
    s = e = 0;
    for (int i = 0; i < n; i++)
    {
        double x, y; scanf("%lf %lf", &x, &y);
        tx[i] = x, ty[i] = y;
        if (fi(tx[s] - tx[i]) == 1) s = i;
```

```
        if (fi(tx[e] - tx[i]) == -1) e = i;
    }
    for (int i = 0; i < m; i++)
    {
        int a, b; double x; scanf("%d %d %lf", &a, &b, &x);
        if (a == b) { --m, --i; continue; }
        int eg1 = edge1(i), eg2 = edge2(i); capacity[i] = x;
        double agab = atan2(ty[b] - ty[a], tx[b] - tx[a]);
        double agba = atan2(ty[a] - ty[b], tx[a] - tx[b]);
        vertex[a].push_back(edge_rec(eg1, eg2, agab));
        vertex[b].push_back(edge_rec(eg2, eg1, agba));
    }
    int adder1 = edge1(m), adder2 = edge2(m);
    vertex[s].push_back(edge_rec(adder1, adder2, -pi));
    vertex[e].push_back(edge_rec(adder2, adder1, 0));
    for (int i = 0; i < n; i++)
    {
        sort(vertex[i].begin(), vertex[i].end());
        int ms = vertex[i].size();
        for (int j = 0; j < ms - 1; j++) next_rec[vertex[i][j].in] = vertex[i
          ][j + 1].out;
        next_rec[vertex[i][ms - 1].in] = vertex[i][0].out;
    }
    areamr = 0; memset(sch, false, sizeof sch);
    for (int i = 0; i <= adder2; i++) if (!sch[i]) find_ring(i, i), ++areamr;
    ginit(areamr);
    for (int i = 0; i < m; i++)
    {
        int eg1 = edge1(i), eg2 = edge2(i);
        glink(belong[eg1], belong[eg2], capacity[i]);
        glink(belong[eg2], belong[eg1], capacity[i]);
    }
    printf("%.4f\n", dijkstra(belong[adder1], belong[adder2]));
    return 0;
}
```

8.4 Point Location

不掉精度的题可以试试，现在不太好用。自带持久化，改成非持久化小心写错。

```
#include <stdio.h>
#include <string.h>
#include <algorithm>
#include <map>
#include <vector>
#include <cmath>

using namespace std;

typedef long long ll;
typedef pair<int,int> pii;

template<typename T,typename HT>
class POINT2D_META
{
public:
  T x;
  T y;
  POINT2D_META() { x = y = 0; }
  POINT2D_META(T _x,T _y):x(_x),y(_y) { }
  inline POINT2D_META operator+(const POINT2D_META& b) const { return
    POINT2D_META(x+b.x,y+b.y); }
  inline POINT2D_META operator-(const POINT2D_META& b) const { return
    POINT2D_META(x-b.x,y-b.y); }
  inline HT operator*(const POINT2D_META& b) const { return (HT)x*b.y-(HT)b.x
    *y; }
  inline HT dot(const POINT2D_META& b) const { return (HT)x*b.x+(HT)y*b.y; }
  inline HT hypot2() const { return (HT)x*x+(HT)y*y; }
  inline T hypot() const { return ::hypot(x,y); }
  inline int read() { return scanf("%d %d",&x,&y); }
  // h-v order
  inline bool operator<(const POINT2D_META& b) const { return x == b.x ? y <
    b.y : x < b.x; }
  inline bool operator==(const POINT2D_META& b) const { return x == b.x && y
    == b.y; }
};
typedef POINT2D_META<int,ll> POINT;

class SEGMENT
{
public:
  static long double x;

  POINT l; POINT r;
  int which;
  int side;
  SEGMENT() {}
  SEGMENT(const POINT& _l,const POINT& _r,int which,int side):l(_l),r(_r),
    which(which),side(side) {if(l.x > r.x) swap(l,r); }
  long double evaluate() const { return (x-l.x)/((long double)r.x-l.x)*((long
     double)r.y-l.y)+l.y; }
  inline bool operator==(const SEGMENT& b) const { return side == b.side &&
    which == b.which && l == b.l && r == b.r; }
}; long double SEGMENT::x = 0.0; // dirty implementation

class TNODE
{
private:
  inline bool determineSide(int a,int b) { return rand() % (a+b) < a; }
```

```
public:
  SEGMENT val;
  int size,refCount;
  TNODE* left;
  TNODE* right;

  TNODE(const SEGMENT& val,TNODE* left,TNODE* right):val(val),left(left),
    right(right)
  {
    update();
    refCount = 0;
    if(left) left->refCount++;
    if(right) right->refCount++;
  }
  ~TNODE()
  {
    if(left && --left->refCount <= 0) delete left;
    if(right && --right->refCount <= 0) delete right;
  }

  inline TNODE* ref() { if(this) refCount++; return this; }
  inline TNODE* deref() { if(this && --refCount <= 0) { delete this; return
    NULL; } return this; }

  inline TNODE* update()
  {
    size = (left ? left->size : 0) + 1 + (right ? right->size : 0);
    return this;
  }

  TNODE* merge(TNODE* q)
  {
    TNODE* p = this;
    if(!p) return q;
    if(!q) return p;
    if(determineSide(p->size,q->size)) return new TNODE(p->val,p->left,p->
      right->merge(q)); // as p
    return new TNODE(q->val,p->merge(q->left),q->right); // as q
  }

  typedef pair<TNODE*,TNODE*> ptt;
  ptt split(const SEGMENT& base) // split right after base, and erase base if
     found
  {
    if(!this) return ptt(NULL,NULL);
    if(base == val) return ptt(left, right); // erase base
    long double va = base.evaluate();
    long double vb = val.evaluate();
    if(va < vb)
    {
      ptt st = left->split(base);
      return ptt(st.first, new TNODE(val,st.second,right)); // as self
    }
    else
    {
      ptt st = right->split(base);
      return ptt(new TNODE(val,left,st.first), st.second); // as self
    }
  }

  TNODE* lower_bound(long double va)
  {
    if(!this) return NULL;
    long double vb = val.evaluate();
    if(fabs(vb-va) < 1e-8) return this; // no dup entry, no issue
    if(va > vb) return right ? right->lower_bound(va) : NULL;

    TNODE* ans = NULL;
    if(left) ans = left->lower_bound(va);
    if(!ans) ans = this;
    return ans;
  }

  TNODE* abit_smaller(long double va)
  {
    if(!this) return NULL;
    long double vb = val.evaluate();
    if(va <= vb+1e-8) return left ? left->abit_smaller(va) : NULL;

    TNODE* ans = NULL;
    if(right) ans = right->abit_smaller(va);
    if(!ans) ans = this;
    return ans;
  }
};typedef pair<TNODE*,TNODE*> ptt;

vector<POINT> polygon[111111];

struct EVENT
{
  int key,type;
  SEGMENT seg;
  EVENT() {}
  EVENT(int key,int type,const SEGMENT& seg):key(key),type(type),seg(seg) {}
  // warning: non-stable, do not use on set or similar things.
  inline bool operator<(const EVENT& b) const { return key != b.key ? key < b
    .key : type < b.type; }
};
```

```
EVENT events[666666];
int xpos[333333];
int xpcnt = 0;
TNODE* rootNode[333333];
int ShootingStar(int N)
{
  int evcnt = 0;
  for(int i = 0;i < N;i++)
  {
    int cnt = polygon[i].size();
    ll area = 0;
    for(int j = 0;j < cnt;j++) area += polygon[i][j] * polygon[i][(j+1)%cnt];
    if(area < 0) reverse(polygon[i].begin(),polygon[i].end());

    for(int j = 0;j < cnt;j++)
    {
      POINT now = polygon[i][j];
      POINT next = polygon[i][(j+1)%cnt];
      if(now.x == next.x) continue;

      int side = 1;
      if(now.x > next.x) { swap(now,next); side ^= 1; }
      events[evcnt++] = EVENT(now.x,1,SEGMENT(now,next,i,side));
      events[evcnt++] = EVENT(next.x,-1,SEGMENT(now,next,i,side));
    }
  }
  sort(events,events+evcnt);

  int xi = 0;
  for(int i = 0;i < evcnt;i++) if(!i || events[i].key != events[i-1].key)
    xpos[xpcnt++] = events[i].key;
  for(int i = 0;i < evcnt;)
  {
    int p = 0;
    for(p = i;p < evcnt && events[p].key == xpos[xi];p++)
    {
      if(events[p].type == -1)
      {
        SEGMENT::x = ((long double)xpos[xi-1]+xpos[xi])/2.0;
        TNODE* orig = rootNode[xi];
        ptt part = rootNode[xi]->split(events[p].seg); part.first->ref();
          part.second->ref();
        rootNode[xi] = part.first->merge(part.second)->ref();
        part.first->deref(); part.second->deref(); orig->deref();
      }
      else if(events[p].type == 1)
      {
        SEGMENT::x = ((long double)xpos[xi+1]+xpos[xi])/2.0;
        TNODE* orig = rootNode[xi];
        ptt part = rootNode[xi]->split(events[p].seg); part.first->ref();
          part.second->ref();
        TNODE* temp = part.first->merge(new TNODE(events[p].seg,NULL,NULL))->
          ref();
        rootNode[xi] = temp->merge(part.second)->ref();
        part.first->deref(); part.second->deref(); temp->deref(); orig->deref
          ();
      }
    }
    rootNode[xi+1] = rootNode[xi]->ref();
    i = p; xi++;
  }
  return 0;
}

map<pii, int> vertexBelong;
int Memo(TNODE* root,const POINT& target)
{
  SEGMENT::x = target.x;
  TNODE* l = root->abit_smaller(target.y);
  TNODE* r = root->lower_bound(target.y);
  if(r && ((l && l->val.which == r->val.which && l->val.side == 1 && r->val.
    side == 0) ||
    fabs(r->val.evaluate()-target.y) < 1e-8)) return r->val.which+1;
  return -1;
}
int Marisa(const POINT& target)
{
  if(target.x > xpos[xpcnt-1] || target.x < xpos[0]) return -1;
  auto it = vertexBelong.find(pii(target.x,target.y));
  if(it != vertexBelong.end()) return it->second+1;

  int tx = max(0,lower_bound(xpos,xpos+xpcnt,target.x)-xpos-1);
  int ans = Memo(rootNode[tx],target);
  if(ans == -1 && tx+1 < xpcnt && xpos[tx+1] == target.x) ans = Memo(rootNode
    [tx+1],target);
  return ans;
}

int main(void)
{
  int N = 0;
  scanf("%d",&N);

  for(int i = 0;i < N;i++)
  {
    int cnt = 0;
    scanf("%d",&cnt);
    for(int j = 0;j < cnt;j++)
```

```
    {
      POINT cur; cur.read();
      polygon[i].push_back(cur);
      vertexBelong.insert(make_pair(pii(cur.x,cur.y),i));
    }
  }

  ShootingStar(N);

  int Q = 0;
  scanf("%d",&Q);
  while(Q--)
  {
    POINT target; target.read();
    printf("%d\n",Marisa(target));
    fflush(stdout);
  }
  return 0;
}
```

8.5 不知道是啥总之混脸熟

8.5.1 四面体体积公式

U,V,W,u,v,w 是四面体的 6 条棱，U,V,W 构成三角形，$(U,u),(V,v),(W,w)$ 互为对棱，则

$$V=\frac{\sqrt{(s-2a)(s-2b)(s-2c)(s-2d)}}{192uvw}$$

其中

$$\begin{cases} a &= \sqrt{xYZ}, \\ b &= \sqrt{yZX}, \\ c &= \sqrt{zXY}, \\ d &= \sqrt{xyz}, \\ s &= a+b+c+d, \\ X &= (w-U+v)(U+v+w), \\ x &= (U-v+w)(v-w+U), \\ Y &= (u-V+w)(V+w+u), \\ y &= (V-w+u)(w-u+V), \\ Z &= (v-W+u)(W+u+v), \\ z &= (W-u+v)(u-v+W) \end{cases}$$

8.5.2 牛顿恒等式

设

$$\prod_{i=1}^{n}(x-x_i)=a_n+a_{n-1}x+\cdots+a_1x^{n-1}+a_0x^n$$

$$p_k=\sum_{i=1}^{n}x_i^k$$

则

$$a_0p_k+a_1p_{k-1}+\cdots+a_{k-1}p_1+ka_k=0$$

特别地，对于

$$|\mathbf{A}-\lambda\mathbf{E}|=(-1)^n(a_n+a_{n-1}\lambda+\cdots+a_1\lambda^{n-1}+a_0\lambda^n)$$

有

$$p_k=\mathrm{Tr}(\mathbf{A}^k)$$

8.5.3 信仰

A006265 Shapes of height-balanced AVL trees with n nodes.

1, 1, 2, 1, 4, 6, 4, 17, 32, 44, 60, 70, 184, 476, 872, 1553, 2720, 4288, 6312, 9004, 16088, 36900, 82984, 174374, 346048, 653096, 1199384, 2160732, 3812464, 6617304, 11307920, 18978577, 31327104, 51931296, 90400704, 170054336, 341729616, 711634072, 1491256624

G.f.: $A(x)=B(x,0)$ where $B(x,y)$ satisfies $B(x,y)=x+B(x^2+2xy,x)$.

9 积分表

9.1 Integration

9.1.1 含有 $ax+b$ 的积分 ($a\neq 0$)

1. $\int\frac{\mathrm{d}x}{ax+b}=\frac{1}{a}\ln|ax+b|+C$
2. $\int(ax+b)^\mu\mathrm{d}x=\frac{1}{a(\mu+1)}(ax+b)^{\mu+1}+C(\mu\neq 1)$
3. $\int\frac{x}{ax+b}\mathrm{d}x=\frac{1}{a^2}(ax+b-b\ln|ax+b|)+C$
4. $\int\frac{x^2}{ax+b}\mathrm{d}x=\frac{1}{a^3}\left(\frac{1}{2}(ax+b)^2-2b(ax+b)+b^2\ln|ax+b|\right)+C$
5. $\int\frac{\mathrm{d}x}{x(ax+b)}=-\frac{1}{b}\ln\left|\frac{ax+b}{x}\right|+C$
6. $\int\frac{\mathrm{d}x}{x^2(ax+b)}=-\frac{1}{bx}+\frac{a}{b^2}\ln\left|\frac{ax+b}{x}\right|+C$
7. $\int\frac{x}{(ax+b)^2}\mathrm{d}x=\frac{1}{a^2}\left(\ln|ax+b|+\frac{b}{ax+b}\right)+C$
8. $\int\frac{x^2}{(ax+b)^2}\mathrm{d}x=\frac{1}{a^3}\left(ax+b-2b\ln|ax+b|-\frac{b^2}{ax+b}\right)+C$
9. $\int\frac{\mathrm{d}x}{x(ax+b)^2}=\frac{1}{b(ax+b)}-\frac{1}{b^2}\ln\left|\frac{ax+b}{x}\right|+C$

9.1.2 含有 $\sqrt{ax+b}$ 的积分

1. $\int \sqrt{ax+b}\mathrm{d}x = \frac{2}{3a}\sqrt{(ax+b)^3}+C$
2. $\int x\sqrt{ax+b}\mathrm{d}x = \frac{2}{15a^2}(3ax-2b)\sqrt{(ax+b)^3}+C$
3. $\int x^2\sqrt{ax+b}\mathrm{d}x = \frac{2}{105a^3}(15a^2x^2-12abx+8b^2)\sqrt{(ax+b)^3}+C$
4. $\int \frac{x}{\sqrt{ax+b}}\mathrm{d}x = \frac{2}{3a^2}(ax-2b)\sqrt{ax+b}+C$
5. $\int \frac{x^2}{\sqrt{ax+b}}\mathrm{d}x = \frac{2}{15a^3}(3a^2x^2-4abx+8b^2)\sqrt{ax+b}+C$
6. $\int \frac{\mathrm{d}x}{x\sqrt{ax+b}} = \begin{cases} \frac{1}{\sqrt{b}}\ln\left|\frac{\sqrt{ax+b}-\sqrt{b}}{\sqrt{ax+b}+\sqrt{b}}\right|+C & (b>0) \\ \frac{2}{\sqrt{-b}}\arctan\sqrt{\frac{ax+b}{-b}}+C & (b<0) \end{cases}$
7. $\int \frac{\mathrm{d}x}{x^2\sqrt{ax+b}} = -\frac{\sqrt{ax+b}}{bx}-\frac{a}{2b}\int\frac{\mathrm{d}x}{x\sqrt{ax+b}}$
8. $\int \frac{\sqrt{ax+b}}{x}\mathrm{d}x = 2\sqrt{ax+b}+b\int\frac{\mathrm{d}x}{x\sqrt{ax+b}}$
9. $\int \frac{\sqrt{ax+b}}{x^2}\mathrm{d}x = -\frac{\sqrt{ax+b}}{x}+\frac{a}{2}\int\frac{\mathrm{d}x}{x\sqrt{ax+b}}$

9.1.3 含有 $x^2\pm a^2$ 的积分

1. $\int \frac{\mathrm{d}x}{x^2+a^2} = \frac{1}{a}\arctan\frac{x}{a}+C$
2. $\int \frac{\mathrm{d}x}{(x^2+a^2)^n} = \frac{x}{2(n-1)a^2(x^2+a^2)^{n-1}}+\frac{2n-3}{2(n-1)a^2}\int\frac{\mathrm{d}x}{(x^2+a^2)^{n-1}}$
3. $\int \frac{\mathrm{d}x}{x^2-a^2} = \frac{1}{2a}\ln\left|\frac{x-a}{x+a}\right|+C$

9.1.4 含有 $ax^2+b(a>0)$ 的积分

1. $\int \frac{\mathrm{d}x}{ax^2+b} = \begin{cases} \frac{1}{\sqrt{ab}}\arctan\sqrt{\frac{a}{b}}x+C & (b>0) \\ \frac{1}{2\sqrt{-ab}}\ln\left|\frac{\sqrt{a}x-\sqrt{-b}}{\sqrt{a}x+\sqrt{-b}}\right|+C & (b<0) \end{cases}$
2. $\int \frac{x}{ax^2+b}\mathrm{d}x = \frac{1}{2a}\ln\left|ax^2+b\right|+C$
3. $\int \frac{x^2}{ax^2+b}\mathrm{d}x = \frac{x}{a}-\frac{b}{a}\int\frac{\mathrm{d}x}{ax^2+b}$
4. $\int \frac{\mathrm{d}x}{x(ax^2+b)} = \frac{1}{2b}\ln\frac{x^2}{|ax^2+b|}+C$
5. $\int \frac{\mathrm{d}x}{x^2(ax^2+b)} = -\frac{1}{bx}-\frac{a}{b}\int\frac{\mathrm{d}x}{ax^2+b}$
6. $\int \frac{\mathrm{d}x}{x^3(ax^2+b)} = \frac{a}{2b^2}\ln\frac{|ax^2+b|}{x^2}-\frac{1}{2bx^2}+C$
7. $\int \frac{\mathrm{d}x}{(ax^2+b)^2} = \frac{x}{2b(ax^2+b)}+\frac{1}{2b}\int\frac{\mathrm{d}x}{ax^2+b}$

9.1.5 含有 $ax^2+bx+c(a>0)$ 的积分

1. $\frac{\mathrm{d}x}{ax^2+bx+c} = \begin{cases} \frac{2}{\sqrt{4ac-b^2}}\arctan\frac{2ax+b}{\sqrt{4ac-b^2}}+C & (b^2<4ac) \\ \frac{1}{\sqrt{b^2-4ac}}\ln\left|\frac{2ax+b-\sqrt{b^2-4ac}}{2ax+b+\sqrt{b^2-4ac}}\right|+C & (b^2>4ac) \end{cases}$
2. $\int \frac{x}{ax^2+bx+c}\mathrm{d}x = \frac{1}{2a}\ln|ax^2+bx+c|-\frac{b}{2a}\int\frac{\mathrm{d}x}{ax^2+bx+c}$

9.1.6 含有 $\sqrt{x^2+a^2}(a>0)$ 的积分

1. $\int \frac{\mathrm{d}x}{\sqrt{x^2+a^2}} = \operatorname{arsh}\frac{x}{a}+C_1 = \ln(x+\sqrt{x^2+a^2})+C$
2. $\int \frac{\mathrm{d}x}{\sqrt{(x^2+a^2)^3}} = \frac{x}{a^2\sqrt{x^2+a^2}}+C$
3. $\int \frac{x}{\sqrt{x^2+a^2}}\mathrm{d}x = \sqrt{x^2+a^2}+C$
4. $\int \frac{x}{\sqrt{(x^2+a^2)^3}}\mathrm{d}x = -\frac{1}{\sqrt{x^2+a^2}}+C$
5. $\int \frac{x^2}{\sqrt{x^2+a^2}}\mathrm{d}x = \frac{x}{2}\sqrt{x^2+a^2}-\frac{a^2}{2}\ln(x+\sqrt{x^2+a^2})+C$
6. $\int \frac{x^2}{\sqrt{(x^2+a^2)^3}}\mathrm{d}x = -\frac{x}{\sqrt{x^2+a^2}}+\ln(x+\sqrt{x^2+a^2})+C$
7. $\int \frac{\mathrm{d}x}{x\sqrt{x^2+a^2}} = \frac{1}{a}\ln\frac{\sqrt{x^2+a^2}-a}{|x|}+C$
8. $\int \frac{\mathrm{d}x}{x^2\sqrt{x^2+a^2}} = -\frac{\sqrt{x^2+a^2}}{a^2x}+C$
9. $\int \sqrt{x^2+a^2}\mathrm{d}x = \frac{x}{2}\sqrt{x^2+a^2}+\frac{a^2}{2}\ln(x+\sqrt{x^2+a^2})+C$
10. $\int \sqrt{(x^2+a^2)^3}\mathrm{d}x = \frac{x}{8}(2x^2+5a^2)\sqrt{x^2+a^2}+\frac{3}{8}a^4\ln(x+\sqrt{x^2+a^2})+C$
11. $\int x\sqrt{x^2+a^2}\mathrm{d}x = \frac{1}{3}\sqrt{(x^2+a^2)^3}+C$
12. $\int x^2\sqrt{x^2+a^2}\mathrm{d}x = \frac{x}{8}(2x^2+a^2)\sqrt{x^2+a^2}-\frac{a^4}{8}\ln(x+\sqrt{x^2+a^2})+C$
13. $\int \frac{\sqrt{x^2+a^2}}{x}\mathrm{d}x = \sqrt{x^2+a^2}+a\ln\frac{\sqrt{x^2+a^2}-a}{|x|}+C$
14. $\int \frac{\sqrt{x^2+a^2}}{x^2}\mathrm{d}x = -\frac{\sqrt{x^2+a^2}}{x}+\ln(x+\sqrt{x^2+a^2})+C$

9.1.7 含有 $\sqrt{x^2-a^2}(a>0)$ 的积分

1. $\int \frac{\mathrm{d}x}{\sqrt{x^2-a^2}} = \frac{x}{|x|}\operatorname{arch}\frac{|x|}{a} + C_1 = \ln\left|x+\sqrt{x^2-a^2}\right| + C$
2. $\int \frac{\mathrm{d}x}{\sqrt{(x^2-a^2)^3}} = -\frac{x}{a^2\sqrt{x^2-a^2}} + C$
3. $\int \frac{x}{\sqrt{x^2-a^2}}\mathrm{d}x = \sqrt{x^2-a^2} + C$
4. $\int \frac{x}{\sqrt{(x^2-a^2)^3}}\mathrm{d}x = -\frac{1}{\sqrt{x^2-a^2}} + C$
5. $\int \frac{x^2}{\sqrt{x^2-a^2}}\mathrm{d}x = \frac{x}{2}\sqrt{x^2-a^2} + \frac{a^2}{2}\ln\left|x+\sqrt{x^2-a^2}\right| + C$
6. $\int \frac{x^2}{\sqrt{(x^2-a^2)^3}}\mathrm{d}x = -\frac{x}{\sqrt{x^2-a^2}} + \ln\left|x+\sqrt{x^2-a^2}\right| + C$
7. $\int \frac{\mathrm{d}x}{x\sqrt{x^2-a^2}} = \frac{1}{a}\arccos\frac{a}{|x|} + C$
8. $\int \frac{\mathrm{d}x}{x^2\sqrt{x^2-a^2}} = \frac{\sqrt{x^2-a^2}}{a^2x} + C$
9. $\int \sqrt{x^2-a^2}\mathrm{d}x = \frac{x}{2}\sqrt{x^2-a^2} - \frac{a^2}{2}\ln\left|x+\sqrt{x^2-a^2}\right| + C$
10. $\int \sqrt{(x^2-a^2)^3}\mathrm{d}x = \frac{x}{8}(2x^2-5a^2)\sqrt{x^2-a^2} + \frac{3}{8}a^4\ln\left|x+\sqrt{x^2-a^2}\right| + C$
11. $\int x\sqrt{x^2-a^2}\mathrm{d}x = \frac{1}{3}\sqrt{(x^2-a^2)^3} + C$
12. $\int x^2\sqrt{x^2-a^2}\mathrm{d}x = \frac{x}{8}(2x^2-a^2)\sqrt{x^2-a^2} - \frac{a^4}{8}\ln\left|x+\sqrt{x^2-a^2}\right| + C$
13. $\int \frac{\sqrt{x^2-a^2}}{x}\mathrm{d}x = \sqrt{x^2-a^2} - a\arccos\frac{a}{|x|} + C$
14. $\int \frac{\sqrt{x^2-a^2}}{x^2}\mathrm{d}x = -\frac{\sqrt{x^2-a^2}}{x} + \ln\left|x+\sqrt{x^2-a^2}\right| + C$

9.1.8 含有 $\sqrt{a^2-x^2}(a>0)$ 的积分

1. $\int \frac{\mathrm{d}x}{\sqrt{a^2-x^2}} = \arcsin\frac{x}{a} + C$
2. $\frac{\mathrm{d}x}{\sqrt{(a^2-x^2)^3}} = \frac{x}{a^2\sqrt{a^2-x^2}} + C$
3. $\int \frac{x}{\sqrt{a^2-x^2}}\mathrm{d}x = -\sqrt{a^2-x^2} + C$
4. $\int \frac{x}{\sqrt{(a^2-x^2)^3}}\mathrm{d}x = \frac{1}{\sqrt{a^2-x^2}} + C$
5. $\int \frac{x^2}{\sqrt{a^2-x^2}}\mathrm{d}x = -\frac{x}{2}\sqrt{a^2-x^2} + \frac{a^2}{2}\arcsin\frac{x}{a} + C$
6. $\int \frac{x^2}{\sqrt{(a^2-x^2)^3}}\mathrm{d}x = \frac{x}{\sqrt{a^2-x^2}} - \arcsin\frac{x}{a} + C$
7. $\int \frac{\mathrm{d}x}{x\sqrt{a^2-x^2}} = \frac{1}{a}\ln\frac{a-\sqrt{a^2-x^2}}{|x|} + C$
8. $\int \frac{\mathrm{d}x}{x^2\sqrt{a^2-x^2}} = -\frac{\sqrt{a^2-x^2}}{a^2x} + C$
9. $\int \sqrt{a^2-x^2}\mathrm{d}x = \frac{x}{2}\sqrt{a^2-x^2} + \frac{a^2}{2}\arcsin\frac{x}{a} + C$
10. $\int \sqrt{(a^2-x^2)^3}\mathrm{d}x = \frac{x}{8}(5a^2-2x^2)\sqrt{a^2-x^2} + \frac{3}{8}a^4\arcsin\frac{x}{a} + C$
11. $\int x\sqrt{a^2-x^2}\mathrm{d}x = -\frac{1}{3}\sqrt{(a^2-x^2)^3} + C$
12. $\int x^2\sqrt{a^2-x^2}\mathrm{d}x = \frac{x}{8}(2x^2-a^2)\sqrt{a^2-x^2} + \frac{a^4}{8}\arcsin\frac{x}{a} + C$
13. $\int \frac{\sqrt{a^2-x^2}}{x}\mathrm{d}x = \sqrt{a^2-x^2} + a\ln\frac{a-\sqrt{a^2-x^2}}{|x|} + C$
14. $\int \frac{\sqrt{a^2-x^2}}{x^2}\mathrm{d}x = -\frac{\sqrt{a^2-x^2}}{x} - \arcsin\frac{x}{a} + C$

9.1.9 含有 $\sqrt{\pm ax^2+bx+c}(a>0)$ 的积分

1. $\int \frac{\mathrm{d}x}{\sqrt{ax^2+bx+c}} = \frac{1}{\sqrt{a}}\ln\left|2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}\right| + C$
2. $\int \sqrt{ax^2+bx+c}\mathrm{d}x = \frac{2ax+b}{4a}\sqrt{ax^2+bx+c} + \frac{4ac-b^2}{8\sqrt{a^3}}\ln\left|2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}\right| + C$
3. $\int \frac{x}{\sqrt{ax^2+bx+c}}\mathrm{d}x = \frac{1}{a}\sqrt{ax^2+bx+c} - \frac{b}{2\sqrt{a^3}}\ln\left|2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}\right| + C$
4. $\int \frac{\mathrm{d}x}{\sqrt{c+bx-ax^2}} = -\frac{1}{\sqrt{a}}\arcsin\frac{2ax-b}{\sqrt{b^2+4ac}} + C$
5. $\int \sqrt{c+bx-ax^2}\mathrm{d}x = \frac{2ax-b}{4a}\sqrt{c+bx-ax^2} + \frac{b^2+4ac}{8\sqrt{a^3}}\arcsin\frac{2ax-b}{\sqrt{b^2+4ac}} + C$
6. $\int \frac{x}{\sqrt{c+bx-ax^2}}\mathrm{d}x = -\frac{1}{a}\sqrt{c+bx-ax^2} + \frac{b}{2\sqrt{a^3}}\arcsin\frac{2ax-b}{\sqrt{b^2+4ac}} + C$

9.1.10 含有 $\sqrt{\pm\frac{x-a}{x-b}}$ 或 $\sqrt{(x-a)(x-b)}$ 的积分

1. $\int \sqrt{\frac{x-a}{x-b}}\mathrm{d}x = (x-b)\sqrt{\frac{x-a}{x-b}} + (b-a)\ln(\sqrt{|x-a|} + \sqrt{|x-b|}) + C$
2. $\int \sqrt{\frac{x-a}{b-x}}\mathrm{d}x = (x-b)\sqrt{\frac{x-a}{b-x}} + (b-a)\arcsin\sqrt{\frac{x-a}{b-x}} + C$
3. $\int \frac{\mathrm{d}x}{\sqrt{(x-a)(b-x)}} = 2\arcsin\sqrt{\frac{x-a}{b-x}} + C\ (a<b)$
4. $$\int \sqrt{(x-a)(b-x)}\mathrm{d}x = \frac{2x-a-b}{4}\sqrt{(x-a)(b-x)} + \frac{(b-a)^2}{4}\arcsin\sqrt{\frac{x-a}{b-x}} + C, (a<b) \quad (1)$$

9.1.11 含有三角函数的积分

1. $\int \sin x\mathrm{d}x = -\cos x + C$
2. $\int \cos x\mathrm{d}x = \sin x + C$
3. $\int \tan x\mathrm{d}x = -\ln|\cos x| + C$
4. $\int \cot x\mathrm{d}x = \ln|\sin x| + C$
5. $\int \sec x\mathrm{d}x = \ln\left|\tan\left(\frac{\pi}{4}+\frac{x}{2}\right)\right| + C = \ln|\sec x + \tan x| + C$
6. $\int \csc x\mathrm{d}x = \ln\left|\tan\frac{x}{2}\right| + C = \ln|\csc x - \cot x| + C$
7. $\int \sec^2 x\mathrm{d}x = \tan x + C$
8. $\int \csc^2 x\mathrm{d}x = -\cot x + C$
9. $\int \sec x\tan x\mathrm{d}x = \sec x + C$
10. $\int \csc x\cot x\mathrm{d}x = -\csc x + C$
11. $\int \sin^2 x\mathrm{d}x = \frac{x}{2} - \frac{1}{4}\sin 2x + C$
12. $\int \cos^2 x\mathrm{d}x = \frac{x}{2} + \frac{1}{4}\sin 2x + C$
13. $\int \sin^n x\mathrm{d}x = -\frac{1}{n}\sin^{n-1}x\cos x + \frac{n-1}{n}\int \sin^{n-2}x\mathrm{d}x$
14. $\int \cos^n x\mathrm{d}x = \frac{1}{n}\cos^{n-1}x\sin x + \frac{n-1}{n}\int \cos^{n-2}x\mathrm{d}x$
15. $\frac{\mathrm{d}x}{\sin^n x} = -\frac{1}{n-1}\frac{\cos x}{\sin^{n-1}x} + \frac{n-2}{n-1}\int \frac{\mathrm{d}x}{\sin^{n-2}x}$
16. $\frac{\mathrm{d}x}{\cos^n x} = \frac{1}{n-1}\frac{\sin x}{\cos^{n-1}x} + \frac{n-2}{n-1}\int \frac{\mathrm{d}x}{\cos^{n-2}x}$
17. $$\begin{aligned}&\int \cos^m x\sin^n x\mathrm{d}x\\ =&\frac{1}{m+n}\cos^{m-1}x\sin^{n+1}x + \frac{m-1}{m+n}\int \cos^{m-2}x\sin^n x\mathrm{d}x\\ =&-\frac{1}{m+n}\cos^{m+1}x\sin^{n-1}x + \frac{n-1}{m+1}\int \cos^m x\sin^{n-2}x\mathrm{d}x\end{aligned}$$
18. $\int \sin ax\cos bx\mathrm{d}x = -\frac{1}{2(a+b)}\cos(a+b)x - \frac{1}{2(a-b)}\cos(a-b)x + C$
19. $\int \sin ax\sin bx\mathrm{d}x = -\frac{1}{2(a+b)}\sin(a+b)x + \frac{1}{2(a-b)}\sin(a-b)x + C$
20. $\int \cos ax\cos bx\mathrm{d}x = \frac{1}{2(a+b)}\sin(a+b)x + \frac{1}{2(a-b)}\sin(a-b)x + C$
21. $\int \frac{\mathrm{d}x}{a+b\sin x} = \begin{cases}\frac{2}{\sqrt{a^2-b^2}}\arctan\frac{a\tan\frac{x}{2}+b}{\sqrt{a^2-b^2}} + C & (a^2>b^2)\\ \frac{1}{\sqrt{b^2-a^2}}\ln\left|\frac{a\tan\frac{x}{2}+b-\sqrt{b^2-a^2}}{a\tan\frac{x}{2}+b+\sqrt{b^2-a^2}}\right| + C & (a^2<b^2)\end{cases}$
22. $\int \frac{\mathrm{d}x}{a+b\cos x} = \begin{cases}\frac{2}{a+b}\sqrt{\frac{a+b}{a-b}}\arctan\left(\sqrt{\frac{a-b}{a+b}}\tan\frac{x}{2}\right) + C & (a^2>b^2)\\ \frac{1}{a+b}\sqrt{\frac{a+b}{a-b}}\ln\left|\frac{\tan\frac{x}{2}+\sqrt{\frac{a+b}{b-a}}}{\tan\frac{x}{2}-\sqrt{\frac{a+b}{b-a}}}\right| + C & (a^2<b^2)\end{cases}$
23. $\int \frac{\mathrm{d}x}{a^2\cos^2 x + b^2\sin^2 x} = \frac{1}{ab}\arctan\left(\frac{b}{a}\tan x\right) + C$
24. $\int \frac{\mathrm{d}x}{a^2\cos^2 x - b^2\sin^2 x} = \frac{1}{2ab}\ln\left|\frac{b\tan x + a}{b\tan x - a}\right| + C$
25. $\int x\sin ax\mathrm{d}x = \frac{1}{a^2}\sin ax - \frac{1}{a}x\cos ax + C$
26. $\int x^2\sin ax\mathrm{d}x = -\frac{1}{a}x^2\cos ax + \frac{2}{a^2}x\sin ax + \frac{2}{a^3}\cos ax + C$
27. $\int x\cos ax\mathrm{d}x = \frac{1}{a^2}\cos ax + \frac{1}{a}x\sin ax + C$
28. $\int x^2\cos ax\mathrm{d}x = \frac{1}{a}x^2\sin ax + \frac{2}{a^2}x\cos ax - \frac{2}{a^3}\sin ax + C$

9.1.12 含有反三角函数的积分 (其中 $a>0$)

1. $\int \arcsin\frac{x}{a}\mathrm{d}x = x\arcsin\frac{x}{a} + \sqrt{a^2-x^2} + C$
2. $\int x\arcsin\frac{x}{a}\mathrm{d}x = (\frac{x^2}{2}-\frac{a^2}{4})\arcsin\frac{x}{a} + \frac{x}{4}\sqrt{x^2-x^2} + C$
3. $\int x^2\arcsin\frac{x}{a}\mathrm{d}x = \frac{x^3}{3}\arcsin\frac{x}{a} + \frac{1}{9}(x^2+2a^2)\sqrt{a^2-x^2} + C$
4. $\int \arccos\frac{x}{a}\mathrm{d}x = x\ arccos\frac{x}{a} - \sqrt{a^2-x^2} + C$
5. $\int x\arccos\frac{x}{a}\mathrm{d}x = (\frac{x^2}{2}-\frac{a^2}{4})\arccos\frac{x}{a} - \frac{x}{4}\sqrt{a^2-x^2} + C$
6. $\int x^2\arccos\frac{x}{a}\mathrm{d}x = \frac{x^3}{3}\arccos\frac{x}{a} - \frac{1}{9}(x^2+2a^2)\sqrt{a^2-x^2} + C$
7. $\int \arctan\frac{x}{a}\mathrm{d}x = x\arctan\frac{x}{a} - \frac{a}{2}\ln(a^2+x^2) + C$
8. $\int x\arctan\frac{x}{a}\mathrm{d}x = \frac{1}{2}(a^2+x^2)\arctan\frac{x}{a} - \frac{a}{2}x + C$
9. $\int x^2\arctan\frac{x}{a}\mathrm{d}x = \frac{x^3}{3}\arctan\frac{x}{a} - \frac{a}{6}x^2 + \frac{a^3}{6}\ln(a^2+x^2) + C$

9.1.13 含有指数函数的积分

1. $\int a^x\mathrm{d}x = \frac{1}{\ln a}a^x + C$
2. $\int \mathrm{e}^{ax}\mathrm{d}x = \frac{1}{a}a^{ax} + C$
3. $\int x\mathrm{e}^{ax}\mathrm{d}x = \frac{1}{a^2}(ax-1)a^{ax} + C$
4. $\int x^n\mathrm{e}^{ax}\mathrm{d}x = \frac{1}{a}x^n\mathrm{e}^{ax} - \frac{n}{a}\int x^{n-1}\mathrm{e}^{ax}\mathrm{d}x$
5. $\int xa^x\mathrm{d}x = \frac{x}{\ln a}a^x - \frac{1}{(\ln a)^2}a^x + C$
6. $\int x^n a^x\mathrm{d}x = \frac{1}{\ln a}x^n a^x - \frac{n}{\ln a}\int x^{n-1}a^x\mathrm{d}x$
7. $\int \mathrm{e}^{ax}\sin bx\mathrm{d}x = \frac{1}{a^2+b^2}\mathrm{e}^{ax}(a\sin bx - b\cos bx) + C$

8. $\int \mathrm{e}^{ax}\cos bx\mathrm{d}x = \frac{1}{a^2+b^2}\mathrm{e}^{ax}(b\sin bx + a\cos bx) + C$
9. $\int \mathrm{e}^{ax}\sin^n bx\mathrm{d}x = \frac{1}{a^2+b^2n^2}\mathrm{e}^{ax}\sin^{n-1} bx(a\sin bx - nb\cos bx) + \frac{n(n-1)b^2}{a^2+b^2n^2}\int \mathrm{e}^{ax}\sin^{n-2} bx\mathrm{d}x$
10. $\int \mathrm{e}^{ax}\cos^n bx\mathrm{d}x = \frac{1}{a^2+b^2n^2}\mathrm{e}^{ax}\cos^{n-1} bx(a\cos bx + nb\sin bx) + \frac{n(n-1)b^2}{a^2+b^2n^2}\int \mathrm{e}^{ax}\cos^{n-2} bx\mathrm{d}x$

9.1.14 含有对数函数的积分

1. $\int \ln x\mathrm{d}x = x\ln x - x + C$
2. $\int \frac{\mathrm{d}x}{x\ln x} = \ln\left|\ln x\right| + C$
3. $\int x^n \ln x\mathrm{d}x = \frac{1}{n+1}x^{n+1}(\ln x - \frac{1}{n+1}) + C$
4. $\int (\ln x)^n\mathrm{d}x = x(\ln x)^n - n\int (\ln x)^{n-1}\mathrm{d}x$
5. $\int x^m(\ln x)^n\mathrm{d}x = \frac{1}{m+1}x^{m+1}(\ln x)^n - \frac{n}{m+1}\int x^m(\ln x)^{n-1}\mathrm{d}x$