目录

1 Data Structure

1.1 BIT Kth

```
int Kth(int k) {
  int cnt = 0, 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;
}
```

1.2 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], *fa;
};
KDNODE NPool[111111];
KDNODE *NPTop = NPool, *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;
  int mid = (l+r)/2;
  if(depth&1) nth_element(pnt+l,pnt+mid,pnt+r,cmp_y);
    //sort(pnt+l,pnt+r,cmp_y);
  else nth_element(pnt+l,pnt+mid,pnt+r,cmp_x);
    //sort(pnt+l,pnt+r,cmp_x);
  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;
}
void 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++;
    }
  }
}
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;
}
```

1.3 LCT_Path

```
namespace tree {
    struct node{int ch[2],r,p,w,isum,imax;bool rev;}t[NN];
    void pushup(int o) {
        if(o==0)return;
        t[o].isum=t[o].imax=t[o].w;
        int lc=t[o].ch[0],rc=t[o].ch[1];
        if(t[lc].imax>t[o].imax)t[o].imax=t[lc].imax;
        if(t[rc].imax>t[o].imax)t[o].imax=t[rc].imax;
        t[o].isum+=t[lc].isum;
        t[o].isum+=t[rc].isum;
    }
    void pushdown(int o) {
        if(t[o].rev) {
            swap(t[o].ch[0],t[o].ch[1]);
            t[t[o].ch[0]].r^=1;
            t[t[o].ch[1]].r^=1;
            t[t[o].ch[0]].rev^=1;
            t[t[o].ch[1]].rev^=1;
            t[o].rev=false;
        }
    }
    bool isroot(int x)
    {return x==0||(t[t[x].p].ch[0]!=x&&t[t[x].p].ch[1]!=x)
    ;}
    void setc(int c,int p,int sty)
    {t[p].ch[sty]=c;t[c].p=p;t[c].r=sty;pushup(p);}
    void rotate(int o,int sty) {
        int x=t[o].ch[sty];
        if(isroot(o))t[x].p=t[o].p;
        else setc(x,t[o].p,t[o].r);
        setc(t[x].ch[1^sty],o,sty);
        setc(o,x,1^sty);
        pushup(x);
        if(!isroot(x))pushup(t[x].p);
    }
    int stk[NN],top;
    void splay(int o) {
        int f=o,g;
        stk[top=1]=o;
        while(!isroot(f)){stk[++top]=t[f].p;f=t[f].p;}
        while(top)pushdown(stk[top--]);
        while(!isroot(o)) {
            f=t[o].p;g=t[f].p;
            if(isroot(f))rotate(f,t[o].r);
            else {
                if(t[o].r==t[f].r)rotate(g,t[f].r);
                else rotate(f,t[o].r);
                rotate(t[o].f , t[o].r);
            }
        }
    }
    void access(int o) {
        int pre=0;
        while(o) {
            splay(o);
            setc(pre,o,1);
            pre=o;
            o=t[o].p;
        }
    }
    void makeroot(int o)
    {access(o);splay(o);t[o].rev^=1;}
    void link(int a,int b)
    {makeroot(b);t[b].p=a;}
    void modify(int o,int v)
    {splay(o);t[o].w=v;pushup(o);}
}
```

1.4 LCT_Subtree

```
// MaxN 节点数
// 支持link cut 修改点权，查询所在联通块最大值
// 简单版的维护子树lct，不要随便改奇怪的地方。
struct node {
  int f,ch[2],Max; bool isroot,rev;
  multiset<int> Heap;
  void Init(int _v,int _f) {
    isroot = true; rev = false ;
    ch[0] = ch[1] = 0; f = _f;
    Heap.clear(); Heap.insert(_v); Max = _v;
  }
  void Make_Rev() {
    swap(ch[0],ch[1]); rev ^= 1;
  }
}pt[MaxN];
struct LCT {
  void push(int t) {
    if (pt[t].rev) {
      if (pt[t].ch[0]) pt[pt[t].ch[0]].Make_Rev();
      if (pt[t].ch[1]) pt[pt[t].ch[1]].Make_Rev();
      pt[t].rev = 0;
    }
  }
  void update(int t) {
    if (!t) while(1);
    pt[t].Max = max(*pt[t].Heap.rbegin(),max(pt[pt[t].ch
     [0]].Max,pt[pt[t].ch[1]].Max));
  }
  void zig(int x,bool w){
    int y = pt[x].f; push(y); push(x);
    if(pt[y].isroot)
      pt[y].isroot = false, pt[x].isroot = true;
    push(y); push(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(!pt[x].isroot)
      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(!pt[x].isroot){
      push(x);
      if(pt[pt[x].f].isroot)
        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 access(int x){//1 询问 2 Reverse 0 没事
    int p = 0,q = x; PII ret = MP(0,0);
    while (q) {
      splay(q); push(q);
      if (pt[q].ch[1]) {
        pt[pt[q].ch[1]].isroot = true;
        pt[q].Heap.insert(pt[pt[q].ch[1]].Max);
      }
      if (p) {
        if (pt[q].Heap.find(pt[p].Max) == pt[q].Heap.end()
         ) {
          int z = q;
        }
        pt[p].isroot = false; pt[q].Heap.erase(pt[q].Heap.
        find(pt[p].Max));
      }
      pt[q].ch[1] = p; update(q);
      p = q; q = pt[q].f;
    }
    return 0;
  }
  int Find(int p){
    access(p);splay(p);
    int t = p; while(pt[t].ch[0]) t = pt[t].ch[0];
    return t;
  }
  void Link(int x,int y){//x儿子 y父亲
    if(x==y)return;
    access(x); access(y);
    pt[x].f = y; pt[y].Heap.insert(pt[x].Max);
    update(y);
  }
  bool Son(int x,int y){//返回x是y的儿子
    access(y); splay(x); return (pt[x].f == y);
  }
  void Cut(int x,int y){
    if(!Son(x,y)) swap(x,y);
    if(x==y)return;
    //pt[y].Heap.erase(pt[y].Heap.find(pt[x].Max));
    access(x); splay(y);
    pt[y].ch[1] = 0; update(y);
    pt[x].f = 0; pt[x].isroot = true;
  }
}DTree;
```

1.5 Merge-Split Treap

需要改成持久化的话每次修改的时候新建节点。必要的情况下在 newNode 里面加上 GC。

```
// for persistence: random() when merge() -> rand()%(a->
size+b->size)<a->size
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;
  for(;x->fa;x = x->fa)
    if(x == x->fa->right) ans += (x->fa->left ? x->fa->
    left->size : 0) + 1;
  return ans;
}
```

1.6 Mo Algorithm

1.6.1 树上莫队

记树的 dfs 序为 S，则路径 u—v 在 dfs 序中即为 [dfn1[u] + 1 , dfn1[v]] 加上 lca(u , v)，其中出现两次的点记为不出现。

1.7 Query On Subtree

```
namespace Algo {
  #define NN 100001
  int ver[NN],next[NN<<1],to[NN<<1],tot;
  void addedge(int from,int t)
  {next[++tot]=ver[from];to[tot]=t;ver[from]=tot;}
  int st[NN],ed[NN],seq[NN],vt;
  int sz[NN];
  void size(int x,int f) {
    sz[x]=1;
    for(int i=ver[x];i;i=next[i])
      if(to[i]!=f) {
        size(to[i],x);
        sz[x]+=sz[to[i]];
      }
  }
  void solve(int x, int f) {
      st[x]=++vt;
    seq[vt]=col[x];
      int mx=0,mf=0;
      for (int i=ver[x];i;i=next[i])
      if(to[i]!=f&&sz[to[i]]>mx)
              mx=sz[to[i]],mf=to[i];
    for(int i=ver[x];i;i=next[i])
      if(to[i]!=f&&to[i]!=mf)
        solve(to[i],x),del(to[i]);
    if(mf)solve(mf,x);
    for(int i=ver[x];i;i=next[i])
      if(to[i]!=f&&to[i]!=mf)
        ins(to[i]);
    //Add x,...
    //Answer Query,...
  }
}
```

1.8 Scapegoat tree

```
namespace stree {
  const double alpha = 0.618;
  const int SIZE = 15;
  typedef double ld;
  pair<ld , ld> subseg(pair<ld , ld> seg , int r) {
    if(r == 0)
      return make_pair(seg.first , (seg.first + seg.second
      ) / 2.0);
    return make_pair((seg.first + seg.second) / 2.0 , seg.
    second);
  }
struct node{int ch[2] , size , p;pair<ld , ld> seg;}t
[300000];
int tot = 0 , root = 0;
void init() {
  tot = root = 0;
  t[0].seg = make_pair(-1e2, 1e2);
}
int newnode() {
  int now = ++tot;
  t[now].ch[0] = t[now].ch[1] = t[now].p = 0;
  t[now].size = 1;
  return now;
}
void pushup(int o)
{if(!o)return;t[o].size = t[t[o].ch[0]].size + t[t[o].ch
[1]].size + 1;}
void setc(int c , int p , int r) {
  t[c].p = p;
  t[p].ch[r] = c;
  if(p == 0)root = c;
  pushup(p);
  t[c].seg = subseg(t[p].seg , r);
}
int lst[300000] , len;
void go(int o) {
  if(t[o].ch[0])go(t[o].ch[0]);
  lst[++len] = o;
  if(t[o].ch[1])go(t[o].ch[1]);
}
int build(int l , int r , pair<ld , ld> seg) {
  if(l > r)return 0;
  int mid = (l + r) >> 1;
  int now = lst[mid];
  t[now].seg = seg;
  setc(build(l , mid - 1 , subseg(seg , 0)) , now , 0);
  setc(build(mid + 1 , r , subseg(seg , 1)) , now , 1);
  return now;
}
void rebuild(int o) {
  len = 0;go(o);
  setc(build(1 , len , t[o].seg) , t[o].p , o == t[t[o].
  p].ch[1]);
}
void print() {
  len = 0;
  go(root);
  for(int i = 1 ; i <= len ; i++)
    printf("%d " , lst[i]);
  printf("\n");
}
int goat;
void insert(int o , int k) {
  int r = t[t[o].ch[0]].size < (k - 1);
  if(!t[o].ch[r])
    setc(newnode() , o , r);
  else {
    if(r)k -= t[t[o].ch[0]].size + 1;
    insert(t[o].ch[r] , k);
  }
  pushup(o);
  if(t[o].size > SIZE)
  if(max(t[t[o].ch[0]].size , t[t[o].ch[1]].size) > t[o
  ].size * alpha)
    goat = o;
}
int insert(int k) {
  if(root == 0)
    setc(newnode() , 0 , 0);
  else {
    goat = -1;
    insert(root , k);
    if(goat != -1)rebuild(goat);
  }
  return tot;
}
int getrank(int o) {
  int rank = 1 + t[t[o].ch[0]].size;
  while(1) {
    if(t[o].p == 0)break;
    if(t[o].r == 1)
      rank += t[t[t[o].p].ch[0]].size + 1;
```

```cpp
      o = t[o].p;
    }
    return rank;
  }
}
```

1.9 Splay

```cpp
const int MAX_POOL_SIZE = 1000000;
struct node {
  node *f, *ch[2];
  int val, size, idx;
  node () {}
  void init(int _val, int _idx, node *_f) {
    ch[0] = ch[1] = NULL;
    f = _f;
    val = _val; idx = _idx; size = 1;
  }
}Pool[MAX_POOL_SIZE], *Top = Pool, *null;
// remember to make Mr.null point to an instance with zero value.
struct Splay {
  node *root;
  inline bool cid(node *x) { // 0 for left, 1 for left.
    assert(x != root);
    return x == x->f->ch[1];
  }
  inline void push(node *x);
  inline void update(node *x);
  void zig(node *x) {
    node *y = x->f; push(y); push(x);
    if (root == y) root = x;
    bool w = cid(x);
    y->ch[!w] = x->ch[w]; if (x->ch[w] != null) x->ch[w]->f = y;
    x->f = y->f; if (y->f != null) y->f->ch[cid(y)] = x;
    x->ch[w] = y; y->f = x;
    update(y);
  }
  void splay(node *x) {
    while (x != root) {
      if (x->f == root) zig(x);
      else {
        if (cid(x) != cid(x->f)) zig(x); else zig(x->f);
        zig(x);
      }
    } update(x);
  }
  void splay(node *x, node *f) {
    while (x->f != f) {
      if (x->f->f == f) zig(x);
      else {
        if (cid(x) != cid(x->f)) zig(x); else zig(x->f);
        zig(x);
      }
    } update(x);
  }
  //类似set<T>的一些操作
  node *go(node *x, bool dir) {
    // 0 for pred(x), 1 for succ(x)
    splay(x);
    node *t = x->ch[dir];
    while (t->ch[!dir] != null) t = t->ch[!dir];
    return t;
  }
  node *recl;
  node *lbound(node *t, int idx) {
    if (t == null) return null;
    if (t->idx == idx) return t;
    if (t->idx > idx) {
      node *res = lbound(t->ch[0], idx);
      if (recl == null) recl = t;
      return res;
    } else {
      node *res = lbound(t->ch[1], idx);
      if (recl == null) recl = t;
      if (res != null) return res;
      return t;
    }
  }
  node *lower_bound(int idx) {
    recl = null;
    node *res = lbound(root, idx);
    if (recl != null) splay(recl);
    return res;
  }
  // 维护数列相关操作
  node *pick_interval(int idx1, int idx2) {
    //不是多棵Splay的话建议加上极左极右两个假点，方便很多
    node *l = lower_bound(idx1);
    l = go(l, 0);
    node *r = lower_bound(idx2);
    if (r != null && r->idx == idx2) r = go(r, 1);
    if (l != null) splay(l);
    if (r != null) {
      splay(r, l);
      return r->ch[0];
    }
    if (l != null) return l->ch[1]; else return root;
  }
};
```

1.10 Virtual Tree

```cpp
int n, head[100001] , nxt[200001] , to[200001] , tot, stk[100001] , top;
bool used[100001];
vector<int> que, e[100001];
void addedge(int from , int t) {
  nxt[++tot] = head[from];to[tot] = t;head[from] = tot;
}
int in[100001] , out[100001] , clk;
int fa[100001][20] , dep[100001];
void dfs(int o) {
  ++clk;in[o] = clk;
  for(int i = head[o] ; i ; i = nxt[i]) {
    if(!in[to[i]]) {
      dep[to[i]] = dep[o] + 1;
      fa[to[i]][0] = o;
      for(int j = 1 ; j < 20 ; j++)
        fa[to[i]][j] = fa[fa[to[i]][j - 1]][j - 1];
      dfs(to[i]);
    }
  }
  out[o] = clk;
}
int lca(int u , int v) {
  if(dep[u] > dep[v])return lca(v , u);
  int x = dep[v] - dep[u];
  for(int i = 0; i < 20 ; i++)
    if(x >> i & 1)v = fa[v][i];
  if(u == v)return u;
  for(int i = 19 ; i >= 0 ; i--)
    if(fa[u][i] != fa[v][i])
      u = fa[u][i] , v = fa[v][i];
  return fa[u][0];
}
bool cmp_by_rank(int a , int b)
{return in[a] < in[b];}
void build() {
  int len = que.size();
  for(int i = 0 ; i < len ; i++)
    used[que[i]] = true;
  sort(que.begin() , que.end() , cmp_by_rank);
  for(int i = 0 ; i < len - 1 ; i++) {
    int u = lca(que[i] , que[i + 1]);
    if(!used[u])
      que.push_back(u);
  }
  sort(que.begin() , que.end() , cmp_by_rank);
  for(int i = 0 ; i < que.size() ; i++) {
    while(top > 1 && lca(que[i] , stk[top]) != stk[top]) top--;
    if(top) {
      e[stk[top]].pb(que[i]);
      e[que[i]].pb(stk[top]);
    }
    stk[++top] = que[i];
  }
  for(int i = 0 ; i < que.size() ; i++)
    used[que[i]] = false;
  for(int i = 0 ; i < que.size() ; i++)
```

```
    e[que[i]].clear();
  top = 0;
  que.clear();
}
```

1.11 rope

```
#include <ext/rope>
using namespace __gnu_cxx;

rope<int> *s[1000];
s[0] = new rope<int>();
s[i] = new rope<int>(*s[i - 1]);
s[i]->push_back(x);
s[i]->length());
s[i]->at(j);
```

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, 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 LCA (Tarjan)

$O(n)$ 仅在需要顺手维护点别的东西的时候用。

```
void tarjan_lca(int x) {
  ufs[x] = x; vis[x] = true;
  for(QLINK* i = QLink[x];i != NULL;i = i->Next) {
    int qx = i->q->x; int qy = i->q->y;
    if(qx == x && vis[qy]) i->q->lca = ufs_find(qy);
    if(qy == x && vis[qx]) i->q->lca = ufs_find(qx);
  }
  for(EDGE* e = E[x];e;e = e->Next) if(e->y != fa[x]) tarjan_lca(e->y);
  ufs[x] = fa[x];
}
```

2.4 Edmonds matching algorithm

```
int n, match[maxn], pre[maxn], base[maxn]; // maximum matching on graphs
vector<int> edge[maxn];
bool inQ[maxn], inB[maxn], inP[maxn];
queue<int> Q;
int LCA(int u, int v) {
  for (int i = 1; i <= n; ++i) inP[i] = false;
  while (true) {
    u = base[u];
    inP[u] = true;
    if (match[u] == -1) break;
    u = pre[match[u]];
  }
  while (true) {
    v = base[v];
    if (inP[v]) return v;
    v = pre[match[v]];
  }
}
void reset(int u, int a) {
  while (u != a) {
    int v = match[u];
    inB[base[u]] = inB[base[v]] = true;
    v = pre[v];
    if (base[v] != a) pre[v] = match[u];
    u = v;
  }
}
void contract(int u, int v) {
  int a = LCA(u, v);
  for (int i = 1; i <= n; ++i) inB[i] = false;
  reset(u, a), reset(v, a);
  if (base[u] != a) pre[u] = v;
  if (base[v] != a) pre[v] = u;
  for (int i = 1; i <= n; ++i) {
    if (!inB[base[i]]) continue;
    base[i] = a;
    if (!inQ[i]) Q.push(i), inQ[i] = true;
  }
}
bool dfs(int s) {
  for (int i = 1; i <= n; ++i)
    pre[i] = -1, inQ[i] = false, base[i] = i;
  while (!Q.empty()) Q.pop();
  Q.push(s), inQ[s] = true;
  while (!Q.empty()) {
    int u = Q.front();
    Q.pop();
    for (vector<int>::iterator it = edge[u].begin(); it != edge[u].end(); ++it) {
      int v = *it;
      if (base[u] == base[v] || match[u] == v) continue;
      if (v == s || (match[v] != -1 && pre[match[v]] != -1)) contract(u, v);
      else if (pre[v] == -1) {
        pre[v] = u;
        if (match[v] != -1) {
          Q.push(match[v]), inQ[match[v]] = true;
        } 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 blossom() {
 int ans = 0;
 for (int i = 1; i <= n; ++i) match[i] = -1;
 for (int i = 1; i <= n; ++i) {
  if (match[i] == -1 && dfs(i)) ++ans;
 }
 return ans;
}
```

2.5 Hopcroft_Karp

```
const int N = 65536, inf = 0x3fffffff;
int n,m,x,y,match[N*2],dist[N*2], p;
vector<int> e[N];
queue<int> q;
bool bfs() {
  dist[0]=inf;
  for (int i=1;i<=n;i++)
    if (match[i]==0) dist[i]=0,q.push(i);
    else dist[i]=inf;
  while (!q.empty()) {
    int now=q.front();
    q.pop();
    for (auto i: e[now])
      if (dist[match[i]]==inf)
        dist[match[i]]=dist[now]+1,q.push(match[i]);
  }
  return dist[0]!=inf;
}
bool dfs(int x) {
  if (x==0) return true;
  for (auto i: e[x])
    if (dist[x]+1==dist[match[i]]) {
      if (dfs(match[i])) {
        match[x]=i;
        match[i]=x;
        return true;
      }
      else dist[match[i]]=inf;
    }
  return false;
}
int Hopcroft_Karp() {
  int ans=0;
  memset(match,0,sizeof(match));
  while (bfs()) {
    for (int i=1;i<=n;i++)
      if (dist[i]==0)
        if (dfs(i)) ans++;
  }
  return ans;
}
int main() {
  scanf("%d%d%d",&n,&m,&p);
  for (int i=1;i<=p;i++) {
    scanf("%d%d",&x,&y);
    e[x].push_back(y+n);
  }
  printf("%d\n",Hopcroft_Karp());
}
```

2.6 KM

```
// maxn: Left Size; maxm: Right Size.
int n, m, g[maxn][maxm], lx[maxn], ly[maxm], slack[maxm],
match[maxm];
bool vx[maxn], vy[maxm];

bool find(int x) {
  vx[x] = true;
  for (int y = 1; y <= m; ++y) {
    if (!vy[y]) {
      int delta = lx[x] + ly[y] - g[x][y];
      if (delta == 0) {
        vy[y] = true;
        if (match[y] == 0 || find(match[y])) {
          match[y] = x;
          return true;
        }
      } else slack[y] = min(slack[y], delta);
    }
  }
  return false;
}

int km() { // #define sm(p, f) memset((p), f, sizeof(p))
  // maximum weight, if minimum, negate all g then restore
   at the end.
  sm(lx, 0x80), sm(ly, 0), sm(match, 0);
  for (int i = 1; i <= n; ++i)
    for (int j = 1; j <= m; ++j)
      lx[i] = max(lx[i], g[i][j]);
  for (int k = 1; k <= n; ++k) {
    sm(slack, 0x7f);
    while (true) {
      sm(vx, 0), sm(vy, 0);
      if (find(k)) break;
      else {
        int delta = maxint;
        for (int i = 1; i <= m; ++i)
          if (!vy[i]) delta = min(delta, slack[i]);
        for (int i = 1; i <= n; ++i)
          if (vx[i]) lx[i] -= delta;
        for (int i = 1; i <= m; ++i) {
          if (vy[i]) ly[i] += delta;
          if (!vy[i]) slack[i] -= delta;
        }
      }
    }
  }
  int result = 0;
  for (int i = 1; i <= n; ++i) result += lx[i];
  for (int i = 1; i <= m; ++i) result += ly[i];
  return result;
}
```

2.7 MCMF (Cycle Cancling)

```
//调用init初始化
//addEdge加边，注意点的标号从0开始。
struct NetWork {
  static const int NVET_MAX = 60;
  static const int NEDGE_MAX = NVET_MAX * NVET_MAX * 2;
  int head[NVET_MAX], nvet, nedge;
  int dest[NEDGE_MAX], next[NEDGE_MAX], cost[NEDGE_MAX],
  cap[NEDGE_MAX];
  int originCap[NEDGE_MAX];

  void init() {
    memset(head, -1, sizeof head);
    nedge = 0;
    nvet = 0;
  }
  int addVertex() { return nvet++; }
  void makeEdge(int s, int t, int c, int f) { //source,
  dest,cost,flow
    next[nedge] = head[s];
    dest[nedge] = t;
    cost[nedge] = c;
    originCap[nedge] = cap[nedge] = f;
    head[s] = nedge++;
  }
  void addEdge(int s, int t, int c, int f) {
    makeEdge(s, t, c, f);
    makeEdge(t, s, -c, 0);
  }
};

#define FOREDGE(e,G,u) for(int e=G.head[u];e!=-1;e=G.next[
e])
struct MaxCostFlow {
  static const int NVET_MAX = 60;
  int maxCost;
  int n;
  NetWork&network;
  MaxCostFlow(NetWork&_network): network(_network) {
    n = network.nvet;
    calcMaxCostFlow();
  }
  int dist[NVET_MAX];
  int prev[NVET_MAX]; // edge-id
  bool inStack[NVET_MAX];
  int start;
  bool dfsSpfa(int u) {
    inStack[u] = true;
    FOREDGE(e,network,u)
    if (network.cap[e]) {
      int v = network.dest[e];
      int nc = dist[u] + network.cost[e];
      if (nc > dist[v]) {
        dist[v] = nc;
        prev[v] = e;
        if (inStack[v]) {
          start = v;
          return true;
        } else if (dfsSpfa(v)) {
          return true;
        }
      }
    }
    inStack[u] = false;
    return false;
  }
  bool findPositiveCycle() {
    memset(dist, 0, sizeof(int) * n);
    memset(prev, -1, sizeof(int) * n);
    memset(inStack, 0, sizeof(bool) * n);
    for (int u = 0; u < n; u++)
      if (dfsSpfa(u))
        return true;
      return false;
    }
  #define prevNode(it) (it==-1?it:network.dest[prev[it]^1])
  void cancelCycle() {
  //cout<<"Start Cancel:"<<endl;
    int at = start;
    while (true) {
      int edge = prev[at];
      network.cap[edge]--;
      network.cap[edge ^ 1]++;
      maxCost += network.cost[edge];
      at = prevNode(at);
      if (at == start)
        break;
    }
  }
  void calcMaxCostFlow() {
    maxCost = 0;
    while (findPositiveCycle()) {
      cancelCycle();
    }
  }
};
```

2.8 MaxFlow (Dinic)

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。

请，一定，要，用，dfs，找割集。

```
class DinicMaxFlow {
public:
  void reset() {memset(Arc,0,sizeof(Arc)); APTop = APool;}
  DinicMaxFlow() { reset(); }
  int level[MAXN]; int s,t,n;
  bool bfs_level() {
    memset(level,-1,sizeof(level[0]) * (n+3));
    int front(0), end(0);
    static int queue[MAXN];
    queue[end++] = s; level[s] = 0;
    while(front < end) {
      int x = queue[front++];
      for(ARC* ar = Arc[x];ar;ar = ar->Next)
        if(ar->c > 0 && level[ar->y] == -1) {
          level[ar->y] = level[x] + 1;
          queue[end++] = ar->y;
        }
    }
    return level[t] != -1;
  }
  int dfs_augment(int x,int available) {
    if(x == t) return available;
    int used = 0;
    for(ARC* ar = Arc[x];ar && used < available;ar = ar->Next)
      if(ar->c > 0 && level[ar->y] == level[x] + 1) {
        int tflow = dfs_augment(ar->y, min(ar->c, available-used));
        used += tflow;
        ar->c -= tflow; ar->R->c += tflow;
      }
    if(!used) level[x] = -1;
    return used;
  }
  int dinic(int s,int t,int n) {
    this->s = s; this->t = t; this->n = n;
    int maxflow(0), tflow(0);
    while(bfs_level()) while((tflow = dfs_augment(s,INF))) maxflow += tflow;
    return maxflow;
  }
};
```

2.9 SAP

```
namespace nf
{
  #define NN 100000
  #define MM 500000
  struct Edge
  {int to;long long cap;Edge* rev;Edge(to , cap):to(to), cap(cap){}};
  vector<Edge> edges[NN];
  int source , sink , n , tot = 1;
  void clear(){tot=1;for(int i = 1 ; i <= n ; i++)edges[i].clear();}
  void addedge(int from,int des,long long cap) {
    Edge e1 = Edge(des , cap);
    Edge e2 = Edge(from , 0);
    e1.rev = &e2;e2.rev = &e1;
    edges[from].push_back(e1);
    edges[des].push_back(e2);
  }
  int h[NN] , vh[NN];
  long long isap (int x,long long f) {
    if(poi==sink)return f;
    int minh=n-1;long long uf=f;
    for(auto x : edges[poi]) {
      if(x.cap > 0) {
        if(h[poi] == h[x.to] +  1) {
          long long tmp = isap(x.to , min(c[i] , f));
          f -= tmp;
          x.cap -= tmp;
          x.rev->cap += tmp;
          if(f == 0 || h[source] >= n)
            return uf - f;
        }
        if(h[x.to] < minh)minh = h[x.to];
      }
    }
    if(uf == f) {
      if(--vh[h[poi]] == 0)h[source] = n;
      h[poi] = minh + 1;
      vh[h[poi]]++;
    }
    return uf - f;
  }
  long long solve() {
    memset(vh , 0 , sizeof vh);vh[0]=n;
    memset(h , 0 , sizeof h);
    long long flow = 0;
    while(h[source] < n)
      flow += isap(source , 2100000000);
    return flow;
  }
  #undef NN
  #undef MM
  /*
  令G'为残量网络G在缩强连通分量之后的图
  一定在最小割方案中的边：<u,v>满流，且在G'中u=S，v=T
  可能在最小割方案中的边：<u,v>满流，且在G'中u!=v
  对二分图而言
  在最大流方案中一定满流的边：<u,v>满流，且在G'中u!=v
  在最大流方案中可能满流的边：<u,v>满流，或（<u,v>流量为0，且在G'中u=v）
  */
}
```

2.10 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;
}
```

2.11 ZKW Min Cost Max Flow

调用前使用 init 清空, 需要指定 src,des 和 n, 对应 S 和 T，n 为总点数

```
struct EDGE {
  int cost, cap, v;
  int next, re;
}edge[MAXM];
int head[MAXN], e;
int vis[MAXN];
int ans, cost, src, des, n;
void init() {
  memset(head, -1, sizeof(head));
  e = ans = cost = 0;
}
void addedge(int u, int v, int cap, int cost) {
  edge[e].v = v; edge[e].cap = cap;
  edge[e].cost = cost; edge[e].re = e + 1;
  edge[e].next = head[u]; head[u] = e++;

  edge[e].v = u; edge[e].cap = 0;
  edge[e].cost = -cost; edge[e].re = e - 1;
  edge[e].next = head[v]; head[v] = e++;
}
int aug(int u, int f) {
  if(u == des) {
    ans += cost * f;
    return f;
  }
  vis[u] = 1;
  int tmp = f;
  for(int i = head[u]; i != -1; i = edge[i].next) {
    if(edge[i].cap && !edge[i].cost && !vis[edge[i].v]) {
      int delta = aug(edge[i].v, tmp < edge[i].cap ? tmp : edge[i].cap);
      edge[i].cap -= delta; edge[edge[i].re].cap += delta;
      tmp -= delta;
      if(!tmp) return f;
    }
  }
  return f - tmp;
}
bool modlabel() {
  int delta = INF;
  for(int u = 0; u < n; u++) if(vis[u])
    for(int i = head[u]; i != -1; i = edge[i].next)
      if(edge[i].cap && !vis[edge[i].v] && edge[i].cost < delta) delta = edge[i].cost;
  if(delta == INF) return false;
  for(int u = 0; u < n; u++) if(vis[u])
    for(int i = head[u]; i != -1; i = edge[i].next)
      edge[i].cost -= delta, edge[edge[i].re].cost += delta;
  cost += delta;
  return true;
}
void costflow() {
  do do memset(vis, 0, sizeof(vis));
  while(aug(src, INF)); while(modlabel());
}
```

2.12 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)
      t = min(t, (double)(d[n][i]-d[j][i])/(n-j));
    u = max(u, t);
  }
  return u;
}
```

2.13 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;
}
```

2.14 Arborescence

```
int n, ec, ID[maxn], pre[maxn], in[maxn], vis[maxn];
struct edge_t {
  int u, v, w;
} edge[maxm];
void add(int u, int v, int w)
{ edge[++ec].u = u, edge[ec].v = v, edge[ec].w = w; }
int arborescence(int n, int root) {
  int res = 0, index;
  while (true) {
    for (int i = 1; i <= n; ++i)
      in[i] = maxint, vis[i] = -1, ID[i] = -1;
    for (int i = 1; i <= ec; ++i) {
      int u = edge[i].u, v = edge[i].v;
      if (u == v || in[v] <= edge[i].w) continue;
      in[v] = edge[i].w, pre[v] = u;
    }
    pre[root] = root, in[root] = 0;
    for (int i = 1; i <= n; ++i) {
      res += in[i];
      if (in[i] == maxint) return -1;
    }
    index = 0;
    for (int i = 1; i <= n; ++i) {
      if (vis[i] != -1) continue;
      int u = i, v;
      while (vis[u] == -1) {
        vis[u] = i;
        u = pre[u];
      }
      if (vis[u] != i || u == root) continue;
      for (v = u, u = pre[u], ++index; u != v; u = pre[u])
        ID[u] = index;
      ID[v] = index;
    }
    if (index == 0) return res;
    for (int i = 1; i <= n; ++i)
      if (ID[i] == -1) ID[i] = ++index;
    for (int i = 1; i <= ec; ++i) {
      int u = edge[i].u, v = edge[i].v;
      edge[i].u = ID[u], edge[i].v = ID[v];
      edge[i].w -= in[v];
    }
    n = index, root = ID[root];
  }
  return res;
}
```

2.15 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], order[10010], seq[10010], color[10010], usable[10010];
int chordal() {
  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(auto y: edges[t]) label[y]++;
  }
  int ans = 0;
  for(int i = N;i > 0;i--) {
    for(auto y: edges[seq[i]]) usable[color[e->y]] = i;
    int c = 1;
    while(usable[c] == i) c++;
    color[seq[i]] = c;
    ans = max(ans, c)
  }
  return ans;
}
```

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];
void 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);
  }
}
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; }
void ufs_internal_find(int x) {
  if(ufs[ufs[x]] == ufs[x]) return;
  ufs_internal_find(ufs[x]);
  if(semi[mnsemi[ufs[x]]] < semi[mnsemi[x]]) mnsemi[x] = mnsemi[ufs[x]];
  ufs[x] = ufs[ufs[x]];
}
int ufs_find(int x) {
  if(ufs[x] == x) return x;
  ufs_internal_find(x);
  return mnsemi[x];
}
void 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;
}
// main
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 Maximal Clique

```
int G[30];
int ans = 0;
void dfs(int P, int X) {
  if (P == 0 && X == 0) {ans++; return;}
  int p = __builtin_ctz(P | X);
  int Q = P & ~G[p];
  while (Q) {
    int i = __builtin_ctz(Q);
    dfs(P & G[i], X & G[i]);
    Q &= ~(1<<i); P &= ~(1<<i); X |= (1<<i);
  }
}
int Bron_Kerbosch() {
  // remove loop
  for (int i=0; i<N; ++i) G[i] &= ~(1<<i);
  ans = 0;
  dfs((1<<N)-1, 0);
  return ans;
}
```

2.18 Maximum Clique (NCTU)

魔法！

```
ULL edge[MXN];
int go(ULL candi, int ans, int szCur) {
  if (candi == 0) return ans;
  if (ans <= szCur) ans = szCur + 1;
  ULL ncandi = candi;
  int quota = __builtin_popcountll(candi) + szCur - ans;
  while (ncandi && --quota >= 0) {
    int i = __builtin_ctzll(ncandi);
    candi ^= (1ULL << i);
    if (ncandi & edge[i]) ans = go(candi & edge[i], ans,
    szCur + 1);
    ncandi ^= (1ULL << i);
    if (candi == ncandi) ncandi &= ~edge[i];
  }
  return ans;
}
ans = go((1ULL << n) - 1, 0, 0);
```

2.19 Prufer Code

2.19.1 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。

2.19.2 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。

2.19.3 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 $n_1^{n_2-1} * n_2^{n_1-1}$。

2.20 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 $O(n^2)$

```
void 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) {/* use queue if needed */
      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;
      }
    }
  }
}
```

2.21 Steiner Tree

求非负无向图上包含 ts 中所有点的最小 Steiner 树。G 是邻接矩阵。dp[S][v] 表示包含 S 中的点和 v 的最小 Steiner 树。$O(3^t n + 2^t n^2 + n^3)$

```
int dp[1 << MAX_M][MAX_N]; // no memset needed
int steiner(int n, vector<int> ts) {
  int m = ts.size(); if (m < 2) return 0;
  floyd(G);
  for(int i = 0;i < m;i++)
    for(int j = 0;j < n;j++)
      dp[1 << i][j] = G[ts[i]][j];
  for(int i = 1;i < (1 << m);i++) if( ((i-1) & i) != 0 ) {
    for(int j = 0;j < n;j++) {
      dp[i][j] = INF; // 0x3F3F3F3F or something like.
      for(int k = (i-1) & i;k > 0;k = (k-1) & i)
        dp[i][j] = min(dp[i][j], dp[k][j] + dp[i^k][j]);
    }
    for(int j = 0;j < n;j++) for(int k = 0;k < n;k++)
      dp[i][j] = min(dp[i][j], dp[i][k] + G[k][j]);
  }
  return dp[(1<<m) - 1][ts[0]];
}
```

2.22 Planar Gragh

2.22.1 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。

2.22.2 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。

2.22.3 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。

2.23 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。

2.24 2-SAT

顺便提一句 2SAT 输出方案如果要字典序最小的话有个非常蠢的暴力做法，顺序考虑每个点的两种状态，然后类似 bfs 一样暴力去确定其前趋后继。正常的么下面这样做就行了，rev_i 是 i 相对的点，$choose_i$ 为 i 点是否要选在方案中，保证一对点恰有一个在方案中。

```
bool twoSAT(int n) {
  for(int i = 0;i < 2*n;i++) if(!DFN[i]) tarjan(i);
  for(int i = 0;i < 2*n;i++) {
    if(SCC[i] == SCC[rev[i]]) return false;
    choose[i] = SCC[i] < SCC[rev[i]];
  }
  return true;
}
```

2.25 Graph Connected

2.25.1 图在删边后联通性判定

随机一个生成树，给非树边随机权值，树边为经过它的非树边的权值的 xor 一个边的集合在删去后使得图不连通 <=> xor = 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[MAXLEN];
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

z_i = lcp(text+i,pattern)

```
int z_pat[MAXLEN] = {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 Suffix Array

```
int aa[222222], 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];
void 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
}
void 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, m = p) {
    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++;
  }
}
int height[222222]; // lcp of suffix_i and suffix_{i-1}
void 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;
    }
  }
}
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]);
}
```

3.5 Manacher

```
int len,ans;
int p[262144];
// e.g. s = "\0#a#b#a#b#a#"
void manacher(char s[],int n) {
    int maxid=0;
    for (int i=1;i<=n;i++) {
        if (maxid+p[maxid]>=i) p[i]=min(p[maxid*2-i],maxid+p[maxid]-i);
        else p[i]=0;
        while (i+p[i]<n && s[i+p[i]+1]==s[i-p[i]-1]) p[i]++;
        if (i+p[i]>maxid+p[maxid]) maxid=i;
        if (p[i]>ans) ans=p[i];
    }
}
```

3.6 Online Manacher

在线 Manacher，支持尾部添加字母操作，这里的奇偶回文是分开维护的，而不是扩充一倍解决。maxPal 是获取最长回文。

```
template<int delta> class ManacherBase {
private:
  static const int MAXN=1e5+1;
  int r[MAXN]; char s[MAXN];
  int mid,n,i;
public:
  ManacherBase():mid(0),i(0),n(1) {
    memset(r,-1,sizeof(int)*MAXN);
    s[0] = '$'; r[0] = 0;
  }
  int get(int pos) {
    pos++;
    if(pos <= mid) return r[pos];
    else return min(r[mid - (pos - mid)], n - pos - 1);
  }
  void addLetter(char c) {
    s[n] = s[n+1] = c;

    while(s[i - r[i] - 1 + delta] != s[i + r[i] + 1]) r[++i] = get(i-1);
    r[mid=i]++; n++;
  }
  int maxPal() { return ( n - mid - 1 ) * 2 + 1 - delta; }
} ;

class Manacher {
private:
  ManacherBase<1> manacherEven;
  ManacherBase<0> manacherOdd;
public:
  void addLetter(char c) {
    manacherEven.addLetter(c);
    manacherOdd.addLetter(c);
  }
  int maxPal() { return max(manacherEven.maxPal(), manacherOdd.maxPal()); }
  int getRad(int type,int pos) {
    if(type) return manacherOdd.get(pos);
    else return manacherEven.get(pos);
  }
};
```

3.7 PAM

```
struct PAM {
  int S[300000],len;
  struct node{int go[26],fail,len;long long times;}t[500000];
  int tot , last;
  int newnode(int x) {
    memset(&t[tot + 1] , 0 , sizeof t[tot + 1]);
    return tot++;
  }
  void init() {
    tot=0;newnode(0);newnode(-1);t[0].fail=1;
    len=0;last=1;S[0]=-1;
  }
  int getfail(int x) {
    while(S[len-t[x].len-1] != S[len])x=t[x].fail;
    return x;
  }
  void buildnew(int c) {
    S[++len]=c;
    int u=getfail(last);
    if(!t[u].go[c]) {
      int v=newnode(t[u].len+2);
      t[v].fail=t[getfail(t[u].fail)].go[c];
      t[u].go[c]=v;
    }
    last=t[u].go[c];
    t[last].times++;
  }
  void push() {
    for(int i = tot - 1 ; i >= 2 ; i--)
      t[t[i].fail].times += t[i].times;
    t[0].times = 0;
    t[1].times = 0;
  }
};
```

3.8 Suffix Automaton

```
struct SAM {
  struct node{int ch[10],p,maxl,right;}t[300000];
  int tot,tail;
  SAM(){tot=1;tail=1;}
  void buildnew(int c,int len) {
    int now=++tot,p=tail;
    t[now].maxl=len;
    t[now].right=0;
    tail=now;
    while(p&&!t[p].ch[c]){t[p].ch[c]=now;p=t[p].p;}
    if(!p)t[now].p=1;
    else {
      int q=t[p].ch[c];
      if(t[q].maxl==t[p].maxl+1)t[now].p=q;
      else {
        int r=++tot;
        t[r]=t[q];
        t[r].maxl=t[p].maxl+1;
        t[q].p=r;
        t[now].p=r;
        while(p&&t[p].ch[c]==q){t[p].ch[c]=r;p=t[p].p;}
      }
    }
  }
  int b[200000],cnt[200000];
  void sort() {
    memset(cnt,0,sizeof(cnt));
    int maxlen=0;
    for(int i=1;i<=tot;i++) {
      if(t[i].maxl>maxlen)maxlen=t[i].maxl;
      cnt[t[i].maxl]++;
    }
    for(int i=1;i<=maxlen;i++)
      cnt[i]+=cnt[i-1];
    for(int i=1;i<=tot;i++)
      b[cnt[t[i].maxl]--]=i;
  }
  void makeright() {
    sort();
    for(int i=tot;i>=1;i--)
      t[t[b[i]].p].right+=t[b[i]].right;
  }
}sam;
```

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

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

4.1.3 三角形垂心

$$\vec{H}=3\vec{G}-2\vec{O}$$

4.1.4 三角形偏心

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

剩余两点的同理。

4.1.5 三角形内接外接圆半径

$$r=\frac{2S}{a+b+c},\ R=\frac{abc}{4S}$$

4.1.6 Pick's Theorem

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

4.1.7 Euler's Formula

For convex polyhedron: $V-E+F=2$.
For planar graph: $|F|=|E|-|V|+n+1$, n denotes the number of connected components. S is the area of lattice polygon, I is the number of lattice interior points, and B is the number of lattice boundary points.

4.1.8 Heron's Formula

$$S=\sqrt{p(p-a)(p-b)(p-c)}$$
$$p=\frac{a+b+c}{2}$$

4.1.9 超球坐标系

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

4.1.10 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

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

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

4.1.11 立体角公式

ϕ：二面角

$\Omega = (\phi_{ab} + \phi_{bc} + \phi_{ac})\ \mathrm{rad} - \pi\ \mathrm{sr}$

$$\tan\left(\frac{1}{2}\Omega/\mathrm{rad}\right) = \frac{\left|\vec{a}\ \vec{b}\ \vec{c}\right|}{abc + \left(\vec{a}\cdot\vec{b}\right)c + \left(\vec{a}\cdot\vec{c}\right)b + \left(\vec{b}\cdot\vec{c}\right)a}$$

$$\theta_s = \frac{\theta_a + \theta_b + \theta_c}{2}$$

4.1.12 常用体积公式

- Pyramid $V = \frac{1}{3}Sh$.
- Sphere $V = \frac{4}{3}\pi R^3$.
- Frustum $V = \frac{1}{3}h(S_1 + \sqrt{S_1S_2} + S_2)$.
- Ellipsoid $V = \frac{4}{3}\pi abc$.

4.1.13 高维球体积

$$\begin{aligned}
&V_2 = \pi R^2,\ S_2 = 2\pi R \\
&V_3 = \frac{4}{3}\pi R^3,\ S_3 = 4\pi R^2 \\
&V_4 = \frac{1}{2}\pi^2 R^4,\ S_4 = 2\pi^2 R^3 \\
&V_5 = \frac{8}{15}\pi^2 R^5,\ S_5 = \frac{8}{3}\pi^2 R^4 \\
&V_6 = \frac{1}{6}\pi^3 R^6,\ S_6 = \pi^3 R^5 \\
&\text{Generally}, V_n = \frac{2\pi}{n}V_{n-2},\ S_{n-1} = \frac{2\pi}{n-2}S_{n-3} \\
&\text{Where}, S_0 = 2,\ V_1 = 2,\ S_1 = 2\pi,\ V_2 = \pi
\end{aligned}$$

4.1.14 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

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

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 lensqr() { return x * x + y * y; }
  double len() { return sqrt(x * x + y * y); }
};
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(Point a, Point b, Point c, Point d) {
  double u = xmul(a, b, c);
  double v = xmul(b, a, d);
  Point t;
  t.x = (c.x * v + d.x * u) / (u + v);
  t.y = (c.y * v + d.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).lensqr()));
}
// 点q 到线段 {p1, p2} 的距离
double dis_p_seg(Point q, Point p1, Point p2) {
  double dis = fabs((p1 - q) * (p2 - q)) / (p1 - p2).len();
  if (dcmp((q - p1) % (p2 - p1)) > 0 && dcmp((q - p2) % (
```

```cpp
    p1 - p2)) > 0) return dis;
  return min((q - p1).len(), (q - p2).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).lensqr();
  double d = x * x - y * ((p1 - c).lensqr() - 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).lensqr();
  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).lensqr() - (c - p4).lensqr()) < 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;
}
// 点和圆的公切线
vector<Line> getLinePC(Point c, double r, Point p) {
  vector<Line> res;
  double d = (p-c).len();
  if (dcmp(d-r) <= 0) return res;
  double ang = asin(r/d);
  Point v = (c-p) * (sqrt(d*d-r*r)/d);
  res.push_back(Line(p, p + v.rot(-ang)));
  res.push_back(Line(p, p + v.rot(ang)));
  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

```cpp
// 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 lensqr() { 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;
  //lensqr means len^2
  double d = u.lensqr() * v.lensqr() - (u % v) * (u % v);
  //if (abs(d) < eps) return NULL;
  double s = ((p % u) * v.lensqr() - (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 % (p2 - 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).lensqr();
  if (dcmp(d - r * r) >= 0) return -1.;
  d = (p1 - v).len() - sqrt(r * r - d);
  return d;
}
```

4.4 Convex Hull

```cpp
// 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 Minkowski Sum

调用之后请再求一遍凸包。

```cpp
int minkowski(Point h[], Point h1[], Point h2[], int n, int m) {
  int cnt = 0;
  int i = 0, j = 0, ni, nj;
  Point cur;
  while (cnt <= n + m) {
    cur = h1[i] + h2[j];
    if (cnt > 0 && cur == h[0]) break;
    h[cnt ++] = cur;
    ni = (i + 1) % n;
    nj = (j + 1) % m;
    if (cross(h1[i] + h2[nj] - cur, h1[ni] + h2[j] - cur) > 0 || n == 1) j = nj;
    else i = ni;
  }
  return cnt;
}
```

4.6 Euclid Nearest

```cpp
/* 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) { return P[a].y==P[b].y?P[a].x<P[b].x:P[a].y<P[b].y; }
int thisY[111111];
// [l,r)
double closest_pair(int l,int r) {
  double ans = 1e100;
  if(r-l <= 6) {
    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;
}
```

4.7 Minimal Circle Cover

```cpp
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);
  return 0;
}
```

4.8 3D Convex Hull

```cpp
struct Hull3D {
  struct Plane {
    int a, b, c;
  bool ok;
  Plane(){}
  Plane(int a, int b, int c, bool ok)
    : a(a), b(b), c(c), ok(ok) {}
};
int n, tricnt;        //初始点数
int vis[MaxN][MaxN];  //点i到点j是属于哪个面
Plane tri[MaxN << 2]; //凸包三角形
Point3D Ply[MaxN];    //初始点
double dist(Point3D a) {
  return sqrt(a.x * a.x + a.y * a.y + a.z * a.z);
}
double area(Point3D a, Point3D b, Point3D c) {
  return dist((b - a) * (c - a));
}
double volume(Point3D a, Point3D b, Point3D c, Point3D d
) {
  return ((b - a) * (c - a)) % (d - a);
}
double PtoPlane(Point3D &P, Plane f) {  // 正 : 面同向{
  Point3D m = Ply[f.b] - Ply[f.a];
  Point3D n = Ply[f.c] - Ply[f.a];
  Point3D t = P - Ply[f.a];
  return (m * n) % t;
}
void deal(int p, int a, int b) {
  int f = vis[a][b];
  Plane add;
  if (tri[f].ok) {
    if ((PtoPlane(Ply[p], tri[f])) > eps) dfs(p, f);
    else {
      add = Plane(b, a, p, 1);
      vis[p][b] = vis[a][p] = vis[b][a] = tricnt;
      tri[tricnt++] = add;
    }
  }
}
void dfs(int p, int cnt) { // 维护凸包，如果点p在凸包外
更新凸包
  tri[cnt].ok = 0;
  deal(p, tri[cnt].b, tri[cnt].a);
  deal(p, tri[cnt].c, tri[cnt].b);
  deal(p, tri[cnt].a, tri[cnt].c);
}
bool same(int s, int e) { //判面是否相同
  Point3D a = Ply[tri[s].a];
  Point3D b = Ply[tri[s].b];
  Point3D c = Ply[tri[s].c];
  return fabs(volume(a, b, c, Ply[tri[e].a])) < eps
    && fabs(volume(a, b, c, Ply[tri[e].b])) < eps
    && fabs(volume(a, b, c, Ply[tri[e].c])) < eps;
}
void construct() { //构造凸包
  tricnt = 0;
  if (n < 4) return;
  bool tmp = 1;
  for (int i = 1; i < n; ++i) { // 两两不共点
    if (dist(Ply[0] - Ply[i]) > eps) {
      swap(Ply[1], Ply[i]);
      tmp = 0;
      break;
    }
  }
  if (tmp) return;
  tmp = 1;
  for (int i = 2; i < n; ++i) { //前三点不共线
    if ((dist((Ply[0] - Ply[1]) * (Ply[1] - Ply[i]))) >
    eps) {
      swap(Ply[2], Ply[i]);
      tmp = 0;
      break;
    }
  }
  if (tmp) return;
  tmp = 1;
  for (int i = 3; i < n; ++i) { //前四点不共面
    if (fabs((Ply[0] - Ply[1]) * (Ply[1] - Ply[2]) % (
    Ply[0] - Ply[i])) > eps) {
      swap(Ply[3], Ply[i]);
      tmp = 0;
      break;
    }
  }
  if (tmp) return;
  Plane add;
  for (int i = 0; i < 4; ++i) { //初始四面体
    add = Plane((i + 1) % 4, (i + 2) % 4, (i + 3) % 4,
    1);
    if (PtoPlane(Ply[i], add) > 0) swap(add.b, add.c);
    vis[add.a][add.b] = vis[add.b][add.c] = vis[add.c][
    add.a] = tricnt;
    tri[tricnt++] = add;
  }
  for (int i = 4; i < n; ++i) { //构建凸包
    for (int j = 0; j < tricnt; ++j) {
      if (tri[j].ok && (PtoPlane(Ply[i], tri[j])) > eps)
       {
        dfs(i, j);
        break;
      }
    }
  }
  int cnt = tricnt; tricnt = 0;
  for (int i = 0; i < cnt; ++i) { //删除无用的面
    if (tri[i].ok) {
      tri[tricnt++] = tri[i];
    }
  }
}
int Planepolygon() { //多少个面
  int res = 0;
  for (int i = 0; i < tricnt; ++i) {
    bool yes = 1;
    for (int j = 0; j < i; ++j) {
      if (same(i, j)) {
        yes = 0;
        break;
      }
    }
    if (yes) ++res;
  }
  return res;
}
// Volume = sigma(volume(p, a, b, c)); i = 0..tricnt -
1;
} Hull;
```

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

4.10 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;
}
```

4.11 Halfplane Short

```
// O(N^2) sol, polygon counterclockwise order
// i.e., left side of vector v1->v2 is the valid half plane
const double maxd = 1e5;
int n, cnt;
point_t p[maxn];
void init() { // order reversed if right side
  cnt = 4;
  p[1].x = -maxd, p[1].y = -maxd;
  p[2].x = maxd, p[2].y = -maxd;
  p[3].x = maxd, p[3].y = maxd;
  p[4].x = -maxd, p[4].y = maxd;
}
void cut(Point p1, Point p2) {
  int tcnt = 0;
  static Point tp[maxn];
  p[cnt + 1] = p[1];
  for (int i = 1; i <= cnt; ++i) {
    double v1 = cross(p2 - p1, p[i] - p1);
    double v2 = cross(p2 - p1, p[i + 1] - p1);
    if (dcmp(v1) >= 0) tp[++tcnt] = p[i]; // <= if right side
    if (dcmp(v1) * dcmp(v2) < 0)
      tp[++tcnt] = get_intersect(p1, p2, p[i], p[i + 1]);
  }
  cnt = tcnt;
  for (int i = 1; i <= cnt; ++i) p[i] = tp[i];
}
```

4.12 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;
  if(fabs(slr - smr - slm) / slr < eps)return slm + smr;
  return rsimpson(slm,fl,fmid,fml,l,mid)+rsimpson(smr,fmid,fr,fmr,mid,r);
}
```

4.13 Circle Union

圆的面积并. 一定要对圆去重. $area_i$ 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。eps 设成 $1e-8$

```
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 = 0;
  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;
}
```

4.14 Polygon Union

多边形面积并 $O(n^2 logn)$

```cpp
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;
}
```

4.15 球面几何 Common

```cpp
// 注意球面半径为1
bool online(Point l, Point r, Point m) {
  Point v1, v2, v;
  v1 = cross(l, m).dir();
  v2 = cross(m, r).dir();
  v = cross(l, r).dir();
  return dot(v1, v) > 0 && dot(v2, v) > 0;
}
vector <Point> intersect(Point s1, Point t1, Point s2, Point t2) {
  vector <Point> ans;
  Point n1 = cross(s1, t1).dir();
  Point n2 = cross(s2, t2).dir();
  if (n1 == n2) return ans;
  Point p1 = cross(n1, n2).dir();
  Point p2 = Point(0, 0, 0) - p1;
  if (online(s1, t1, p1) && online(s2, t2, p1))
    ans.push_back(p1);
  if (online(s1, t1, p2) && online(s2, t2, p2))
    ans.push_back(p2);
  return ans;
}
double dist(Point a, Point b) {
  return acos(dot(a, b));
}
```

4.16 球面几何学

4.16.1 球面剩余 E

$$E = \alpha + \beta + \gamma - \pi$$

4.16.2 球面多边形面积

$$A = R^2 \cdot E$$

4.16.3 球面正弦定理

$$\frac{\sin a}{\sin A} = \frac{\sin b}{\sin B} = \frac{\sin b}{\sin B}$$

4.16.4 球面余弦定理

$$\cos c = \cos a \cos b + \sin a \sin b \cos C$$

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 Burnside's Lemma

$$|X/G| = \frac{1}{|G|}\sum_{g\in G} |X^g|$$

$$\text{Polya} : X^g = t^{c(g)}$$

Let X^g denote the set of elements in X fixed by g.
$c(g)$ is the number of cycles of the group element g as a permutation of X.

5.1.5 Bell Number

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

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

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

5.1.6 第一类 Stirling 数

n 个元素的项目分作 k 个环排列的方法数目

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

5.1.7 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

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

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

5.1.8 Power Reduction

$a^b\%p = a^{(b\%\varphi(p))+\varphi(p)}\%p\ (b \geq \varphi(p))$

5.1.9 错位排列数

The number of permutations of n elements with no fixed points, denotes as D_n.
$D_n = n!(1-\frac{1}{1!}+\frac{1}{2!}-\frac{1}{3!}+\ldots+(-1)^n\frac{1}{n!})$, or $D_n = n*D_{n-1}+(-1)^n$, $D_n = (n-1)*(D_{n-1}+D_{n-2})$, with $D_1 = 1, D_2 = 1$. The first 10 derangement numbers are 0, 1, 2, 9, 44, 265, 1854, 14833, 133496, 41334961, from $n = 1$.

5.1.10 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 $d^{\frac{p-1}{2}} \equiv -1 \mod p$

5.1.11 级数展开式

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

5.1.12 常用求和式

$$\begin{aligned}
1+2+\ldots+n &= \frac{n^2}{2}+\frac{n}{2} \\
1^2+2^2+\ldots+n^2 &= \frac{n^3}{3}+\frac{n^2}{2}+\frac{n}{6} \\
1^3+2^3+\ldots+n^3 &= \frac{n^4}{4}+\frac{n^3}{2}+\frac{n^2}{4} \\
1^4+2^4+\ldots+n^4 &= \frac{n^5}{5}+\frac{n^4}{2}+\frac{n^3}{3}-\frac{n}{30} \\
1^5+2^5+\ldots+n^5 &= \frac{n^6}{6}+\frac{n^5}{2}+\frac{5n^4}{12}-\frac{n^2}{12} \\
1^6+2^6+\ldots+n^6 &= \frac{n^7}{7}+\frac{n^6}{2}+\frac{n^5}{2}-\frac{n^3}{6}+\frac{n}{42} \\
\mathrm{P}(k) &= \frac{(n+1)^{k+1}-\sum_{i=0}^{k-1}\binom{k+1}{i}\mathrm{P}(i)}{k+1}, \mathrm{P}(0) = n\mathbf{+1}
\end{aligned}$$

$$\sum_{k=1}^{n}k(k+1) = \frac{n(n+1)(n+2)}{3}$$

$$\sum_{k=1}^{n}k(k+1)(k+2) = \frac{n(n+1)(n+2)(n+3)}{4}$$

5.1.13 Pow(b, k)

$$n^k = \sum_{i=0}^{k}\binom{n}{i}*i!*stirling2(k,i)$$

5.1.14 二项式系数

$$\mathrm{C}_r^k = \frac{r}{k}\mathrm{C}_{r-1}^{k-1} \qquad \mathrm{C}_r^k = \mathrm{C}_{r-1}^k+\mathrm{C}_{r-1}^{k-1}$$

$$\mathrm{C}_r^m\mathrm{C}_m^k = \mathrm{C}_r^k\mathrm{C}_{r-k}^{m-k} \qquad \sum_{k\leq n}\mathrm{C}_{r+k}^k = \mathrm{C}_{r+n+1}^n$$

$$\sum_{0\leq k\leq n}\mathrm{C}_k^m = \mathrm{C}_{n+1}^{m+1} \qquad \sum_k\mathrm{C}_r^k\mathrm{C}_s^{n-k} = \mathrm{C}_{r+s}^n$$

The number of non-negative solutions to equation $x_1+x_2+x_3+\ldots+x_k = n$ is $\binom{n+k-1}{k-1}$.

5.1.15 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

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

5.1.16 多项式恒等式

Solve the polynomial congruence equation $f(x) \equiv 0 \mod m$, $m = \prod_{i=1}^{k}p_i^{a_i}$.
We just simply consider the equation $f(x) \equiv 0 \mod p^a$, then use the Chinese Remainder theorem to merge the result.
If x is the root of the equation $f(x) \equiv 0 \mod p^a$, then x is also the root of $f(x) \equiv 0 \mod p^{a-1}$.
$f'(x') \equiv 0 \mod p$ **and** $f(x') \equiv 0 \mod p^a \Rightarrow x = x'+dp^{a-1}(d = 0,\ldots,p-1)$
$f'(x') \not\equiv 0 \mod p \Rightarrow x = x'-\frac{f(x')}{f'(x')}$

5.1.17 概率论

条件概率公式

$$P(A|B) = \frac{P(A\cap B)}{P(B)}$$

贝叶斯定理

$$P(A|B) = \frac{P(B|A)\times P(A)}{P(B)}$$

$$\int_{-\infty}^{\infty}p(x=t)tdt = \int_{-\infty}^{\infty}p(x>=t)dt$$

5.1.18 可重集组合

第 i 个物品有 m_i 种，从 n 个里面选 k 个的方案数。

$$x_{i,j} = x_{i-1,j}+x_{i,j-1}-x_{i-1,j-m_i-1}$$

5.1.19 Eulerian number

$1,\ldots,n$ 的排列，其中恰有 k 个位置的元素比前一个大的排列的数量。

$$E_{n,k} = (k+1)E_{n-1,k}+(n-k)E_{n-1,k-1}$$

5.1.20 卢卡斯定理

For non-negative integers m and n and a prime p, holds the equation

$$\binom{m}{n} \equiv \prod_{i=0}^{k}\binom{m_i}{n_i} \mod p$$

where $m = m_kp^k+m_{k-1}p^{k-1}+\ldots+m_1p+m_0$, and $n = n_kp^k+n_{k-1}p^{k-1}+\ldots+n_1p+n_0$, are the base p expansions of m and n respectively.

5.1.21 威尔逊定理

$$(p-1)! \equiv -1 \mod p$$

5.2 Common

```
inline LL mulmod(LL a,LL b,LL m) {
  a %= m, b %= m;
  LL ans = a * b, s = (LL)((LD)a * (LD)b / (LD)m);
  ans = ans – m * s; ans %= m; if (ans < 0) ans += m;
  return ans;
}

/* remove 'i' in "idiv"/"imul" –> unsigned */
inline long mulmod(long x,long y,long mod) {
  long ans = 0;
  x %= mod, y %= mod;
  __asm__ ( "movq %1,%%rax\n imulq %2\n idivq %3\n"
  :"=d"(ans):"m"(x),"m"(y),"m"(mod):"%rax" );
  return ans;
}

LL fpow(LL a, LL p, int mod); //fpow not pow

int exgcd(int x, int y, int &a, int &b) { // extended gcd,
 ax + by = g.
  int a0 = 1, a1 = 0, b0 = 0, b1 = 1;
  while (y != 0) {
    a0 –= x / y * a1; swap(a0, a1);
    b0 -= x / y * b1; swap(b0, b1);
    x %= y; swap(x, y);
  }
  if (x < 0) a0 = -a0, b0 = -b0, x = -x;
  a = a0, b = b0;
  return x;
}

int inverse(int x, int mod) { // multiplicative inverse.
  int a = 0, b = 0;
  if (exgcd(x, mod, a, b) != 1) return -1;
  return (a % mod + mod) % mod; // C1: x & mod are co-prime
  return fpow(x, mod - 2, mod); // C2: mod is prime
}

void init_inverse(int mod) { // O(n), mod is prime
  inv[0] = inv[1] = 1;
  for (int i = 2; i < n; ++i)
    inv[i] = (LL)inv[mod % i] * (mod - mod / i) % mod; // overflows?
}

LL CRT(int cnt, int *p, int *b) { // chinese remainder theorem
  LL N = 1, ans = 0;
  for (int i = 0; i < k; ++i) N *= p[i];
  for (int i = 0; i < k; ++i) {
    LL mult = (inverse(N / p[i], p[i]) * (N / p[i])) % N;
    mult = (mult * b[i]) % N;
    ans += mult; ans %= N;
  }
  if (ans < 0) ans += N;
  return ans;
}

bool miller_rabin(LL n, LL b) { // miller-rabin prime test
  LL m = n - 1, cnt = 0;
  while (m % 2 == 0) m >>= 1, ++cnt;
  LL ret = fpow(b, m, n);
  if (ret == 1 || ret == n - 1) return true;
  --cnt;
  while (cnt >= 0) {
    ret = mult(ret, ret, n);
    if (ret == n - 1) return true;
    --cnt;
  }
  return false;
}

bool prime_test(LL n) {
  if (n < 2) return false;
  if (n < 4) return true;
  if (n == 3215031751LL) return false;
  const int BASIC[12] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
  for (int i = 0; i < 12 && BASIC[i] < n; ++ i)
    if (!miller_rabin(n, BASIC[i])) return false;
    return true;
}

LL pollard_rho(LL n, LL seed) { // pollard-rho divisors factorization
  LL x, y;
  x = y = rand() % (n - 1) + 1;
  LL head = 1, tail = 2;
  while (true) {
    x = mult(x, x, n);
    x = (x + seed) % n;
    if (x == y) return n;
    LL d = gcd(max(x - y, y - x), n);
    if (1 < d && d < n) return d;
    if (++head == tail) y = x, tail <<= 1;
  }
}

void factorize(LL n, vector<LL> &divisor) {
  if (n == 1) return;
  if (prime_test(n)) divisor.push_back(n);
  else {
    LL d = n;
    while (d >= n) d = pollard_rho(n, rand() % (n - 1) + 1);
    factorize(n / d, divisor);
    factorize(d, divisor);
  }
}

// primtive root, finding the number with order p-1
int primtive_root(int p) {
  vector<int> factor;
  int tmp = p - 1;
  for (int i = 2; i * i <= tmp; ++i) {
    if (tmp % i == 0) {
      factor.push_back(i);
      while (tmp % i == 0) tmp /= i;
    }
  }
  if (tmp != 1) factor.push_back(tmp);
  for (int root = 1; ; ++root) {
    bool flag = true;
    for (int i = 0; i < factor.size(); ++i) {
      if (fpow(root, (p - 1) / factor[i], p) == 1) {
        flag = false;
        break;
      }
    }
    if (flag) return root;
  }
}
```

5.3 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;
}
```

5.4 Modular Factorial

$n! \mod mod$ where $mod = p^k$.

```
LL get(int n, int mod, int p) {
  LL ans = 1;
  for (int i = 1; i <= n; ++i) if (i % p != 0)
    ans = ans * i % mod;
  return ans;
}
pii solve(LL n, int mod, int p) {
  LL init = get(mod, mod, p);
  pii ans = pii(1, 0);
  for (LL now = p; now <= n; now *= p) {
    ans.second += n / now;
    if (now > n / p) break;
  }
  while (n > 0) {
    ans.first = (LL) ans.first * fpow(init, n / mod, mod) % mod;
    ans.first = ans.first * get(n % mod, mod, p) % mod;
    n /= p;
  }
  return ans;
}
```

5.5 NTT

最终结果 mod 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);
    }
  }
}
```

5.6 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]);
  }
}
```

5.7 FWT

```
namespace fwt {
  // [l,r)
  // fwt(a , 0 , 1024)
  #define XOR 1
  #define AND 2
  #define OR 3
  int sty;
  void fwt(int x[],int l,int r) {
    if(l + 1 == r)return;
    int mid = (l + r) >> 1;
    fwt(x , l , mid);
    fwt(x , mid , r);
    for(int i = l ; i < mid ; i++) {
      if(sty==AND)x[i] += x[i-l+mid];
      if(sty==OR)
        x[i-l+mid] += x[i];
      if(sty==XOR) {
        int a = x[i] , b = x[i-l+mid];
        x[i] = a + b;
        x[i-l+mid] = a - b;
      }
    }
  }
  void rfwt(int x[],int l,int r) {
    if(l + 1 == r)return;
    int mid = (l + r) >> 1;
    for(int i = l ; i < mid ; i++) {
      if(sty==AND)x[i] -= x[i-l+mid];
      if(sty==OR)x[i-l+mid] -= x[i];
      if(sty==XOR) {
        int a = x[i] , b = x[i-l+mid];
        x[i] = (a+b) / 2;
        x[i-l+mid] = (a-b) / 2;
      }
    }
    rfwt(x , l , mid);
    rfwt(x , mid , r);
  }
}
```

5.8 Euler Sieve

```
int MinDivi[MAXNUM], Prime[MAXNUM], Miu[MAXNUM], Phi[MAXNUM];
int PCnt = 0;

void 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]));
    }
  }
}
```

5.9 Integer-Partition

整数划分。五边形数 $\frac{3j^2-j}{2}$

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

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

```cpp
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;
    }
  }
}
```

5.10 Lattice Count

计算

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

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

```cpp
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);
}
```

5.11 Mobius Inversion

5.11.1 Hint for Mobius Inversion

$$f(i)=\sum_{j|i} g(j)$$

$$g(i)=\sum_{j|i} f(j)*\mu(i/j)$$

另一种形式：

$$f(i)=\sum_{i|j} g(j)$$

$$g(i)=\sum_{i|j} f(i)*\mu(j/i)$$

5.12 Prime Counting

```cpp
bool notp[100001];
int prime[100000],tot;
void griddle(); // Euler's sieve
int cnt[100001];
std::map<int,int> mem[100001];
int f(int n,int m) {
  if(mem[m][n])return mem[m][n];
  int b = sqrt(n);
  int v;
  if(m > b)v = f(n,b);
  else if(m <= 1)v = n - 1;
  else if(notp[m])v = f(n , m - 1);
  else v=f(n , m - 1) - f(n / m , m - 1) + cnt[m - 1];
  mem[m][n] = v;
  return v;
}
int solve(int n) {
  griddle();
  cnt[1] = 0;
  for(int i = 2 ; i <= 100000 ; i++)
    cnt[i] = cnt[i - 1] + (notp[i] == 0);
  return f(n , sqrt(n));
}
```

5.13 Recurrence Expansion

治疗递推式。对于形如 $a_{i+n} = \sum k_j a_{i+j} + d$ 的递推式，求出其写成 $a_m = \sum c_i a_i + c_n d$ 这一形式时的系数 c_i。工作在 double 上，可以比较显然的改成 int 取模。代码中 c 为传出的答案，需要自行 alloc。

$O(n^2 \log m)$ k_i 稀疏的话用 FFT 可以达成 $O(n \log n \log m)$

```cpp
// c at least double c[n+1]
int recFormula(double* k, int n, long long m, double* c) {
  memset(c,0,sizeof(*c)*(n+1));
  if(m < n) c[m] = 1;
  else {
    double* b = new double[n+1];
    recFormula(k, n, m >> 1, b);
    static double a[MAXN*2];
    memset(a,0,sizeof(a[0])*(n*2));
    int s = m & 1;
    for(int i = 0;i < n;i++) {
      for(int j = 0;j < n;j++) a[i + j + s] += b[i] * b[j];
      c[n] += b[i];
    }
    c[n] = (c[n] + 1) * b[n];
    for(int i = n * 2 - 1;i >= n;i--) {
      for(int j = 0;j < n;j++) a[i - n + j] += k[j] * a[i];
      c[n] += a[i];
    }
    memcpy(c,a,sizeof(c[0])*n); // c[n] is c[n], do not copy it.
    delete[] b;
  }
  return 0;
}
```

5.14 Simplex

```cpp
class Simplex {
public:
  static const int SIZE = 500; //略大于约束条件的最大数量
  vector<double> A[SIZE],B; //约束条件：sum(A[k][i]*X[i])<=B[k]
  vector<double> C,X; //目标函数：max(sum(C[i]*X[i]))
  int n, m; // n: 变元个数, m: 约束个数
  void init(){ // 初始化
    for(int i=0;i<=m;i++) A[i].clear();
    B.clear(); X.clear(); C.clear();
  }
  double gao(){
    // 返回目标函数值，如果是nan则无解，是inf则解无穷大
    A[m].resize(n+1);
    for(int i=0;i<m;i++) A[i].push_back(B[i]);
    for(int i=0;i<n;i++) A[m][i]=-C[i];
    vector<int> idx(n+m);
    for(int i=0;i<n+m;i++) idx[i]=i;
    for(int k=0;k<m;k++) while(cmp(A[k][n],0)<0){
      int r=k,c;
      for(c=0;c<n;c++) if(cmp(A[k][c],0)<0) break;
      if(c==n) return 0.0/0.0;
      for(int i=0;i<m;i++)
        if(cmp(A[i][n],0)>=0 && cmp(A[i][c], 0)>0
        && cmp(A[r][n]/A[r][c],A[i][n]/A[i][c])>0) r=i;
      pivot(idx,r,c);
    }
    for(int cnt=64;cnt--;){ //卡时用，不需要的话 for(;;)
      double tmp,low=1;
      int r=-1,c=-1;
      for(int j=0;j<n;++j) if(cmp(A[m][j],0)<0){
        int k=-1;
        if(c<0) c=j;
        for(int i=0;i<m;++i) if(cmp(A[i][j],0)>0)
        if(k==-1 || cmp(A[k][n]/A[k][j],A[i][n]/A[i][j])>0) k=i;
        if(k!=-1 && cmp(low,tmp=A[m][j]*A[k][n]/A[k][j])>0)
          low=tmp,r=k,c=j;
      }
      if(c<0) break;
      if(r<0) return 1.0/0.0;
      pivot(idx,r,c);
    }
    X.resize(n); // 具体的解在数组X中
    for(int i=0;i<n+m;i++) if(idx[i]<n)
      X[idx[i]]=i<n?0:A[i-n][n];
    return A[m][n];
  }
private:
  double cmp(double a, double b){
    return (1+fabs(a))*EPS<fabs(a-b)?a-b:0;
  }
  void pivot(vector<int>& idx, int r, int c){
    int m=B.size()+1,n=A[0].size();
    double div=1/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[r][j]*A[i][c]*div;
    for(int i=0;i<m;i++) A[i][c]*=-div;
    for(int j=0;j<n;j++) A[r][j]*=+div;
    A[r][c]=div;
    swap(idx[c],idx[n-1+r]);
  }
} SIMP;
```

5.15 Xor Linear Base

求线性无关方程组，本质是个消元，不过按照常用的形式进行了压位（这里是 31 位）。可以顺便维护出一组基。

```
for(int i = 0;i < n;i++) {
  for(int j = 31;j >= 0;j--) {
    if(xx[i] & (1LL<<j)) {
      if(!ind[j]) { ind[j] = xx[i]; break; }
      else xx[i] ^= ind[j];
    }
  }
}
```

5.16 extBSGS

```
int bsgs(int A , int B , int C) { //a ^ x == b (mod c)
  int tmp , i;
  for(i = 0 , tmp = 1 % C ; i <= 50 ; i++ , tmp = (long long)tmp * A % C)
    if(tmp == B)return i;
  int d = 0 , D = 1 % C;
  while((tmp = __gcd(A , C)) != 1) {
    if(B % tmp > 0)return -1;
    B /= tmp;
    C /= tmp;
    D = (long long)D * (A / tmp) % C;
    d++;
  }
  int blk = (int)(ceil(sqrt(double(C))) + 1e-5);
  map<int,int> table;
  for(i = 0 , tmp = 1 % C ; i <= blk ; i++ , tmp = (long long)tmp * A % C)
    if(table.find(tmp) == table.end())
      table[tmp] = i;
  tmp = 1 % C;
  for(int i = 1 ; i <= blk ; i++)tmp = (long long)tmp * A % C;
  for(i = 0 ; i <= blk ; i++ , D = (long long)D * tmp % C) {
    int inv = (long long)inverse(D , C) * B % C;
    if(table.find(inv) != table.end())
      return i * blk + table[inv] + d;
  }
  return -1;
}
```

6 Others

6.1 Circular LCS

a、b 是两个串，正常的 0 下标模式。n 是长度。

```
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;
}
```

6.2 DLX

```
const int N = 1048576;
int L[N],R[N],D[N],U[N],X[N],Y[N],S[N],stk[1024],n,m,sp,k,tmp;
bool flag=false;
void print(int dep); //print stk[1]..stk[dep] flag=true
void uncover(int c) {
  if (flag) return;
  for (int i=D[c];i!=c;i=D[i]) {
    S[c]--;
    for (int j=R[i];j!=i;S[Y[j]]--,j=R[j])
      D[U[j]]=D[j], U[D[j]]=U[j];
  }
  L[R[c]]=L[c]; R[L[c]]=R[c];
}
void recover(int c) {
  if (flag) return;
  for (int i=D[c];i!=c;i=D[i]) {
    S[c]++;
    for (int j=R[i];j!=i;S[Y[j]]++,j=R[j])
      D[U[j]]=j, U[D[j]]=j;
  }
  L[R[c]]=c; R[L[c]]=c;
}
void dfs(int dep) {
  if (flag) return;
  int minn=R[0];
  if (minn==0) { print(dep-1); return; }
  for (int i=minn;i!=0;i=R[i]) {
    if (S[i]==0) return;
    if (S[i]<=S[minn]) minn=i;
  }
  uncover(minn);
  for (int i=D[minn];i!=minn;i=D[i]) {
    stk[dep]=X[i];
    for (int j=R[i];j!=i;j=R[j]) uncover(Y[j]);
    dfs(dep+1);
    for (int j=L[i];j!=i;j=L[j]) recover(Y[j]);
  }
  recover(minn);
}
void insert(int x,int y) {
  S[y]++; X[++sp]=x; Y[sp]=y;
  R[sp]=R[x+m], L[sp]=x+m, L[R[x+m]]=sp, R[x+m]=sp;
  D[sp]=D[y], U[sp]=y, U[D[y]]=sp, D[y]=sp;
}
void init() {
  flag=false;
  sp=n+m; R[0]=0; L[0]=0;
  for (int i=1;i<=m;i++)
    R[i]=R[0],L[i]=0,L[R[0]]=i,R[0]=i,S[i]=0,D[i]=i,U[i]=i
    ;
  for (int i=1;i<=n;i++)
    L[m+i]=m+i, R[m+i]=m+i, D[m+i]=0, U[i+m]=0;
  for (int i=1;i<=n;i++) {
    scanf("%d",&k);
    for (int j=1;j<=k;j++) {
      scanf("%d",&tmp);
      insert(i,tmp);
    }
  }
  for (int i=1;i<=n;i++)
    R[L[m+i]]=R[m+i], L[R[m+i]]=L[m+i];
}
int main() {
  scanf("%d%d",&n,&m);
  init();
  dfs(1);
  if (!flag) printf("NO\n");
}
```

6.3 DLX_recoverable

```
const int maxn = 64; //column
const int maxsize = maxn*(maxn+maxn); //number of nodes
int head = 0; //head pointer
struct links{
  int L,R,U,D,S,row,col;
};
links p[maxsize];
int top, line,m,ans;
bool tb[maxn][maxn], ok, h[maxn];
//int num; //step limit
void addUD(int x,int y) { p[x].D = y; p[y].U = x; }
void addLR(int x,int y) { p[x].R = y; p[y].L = x; }
void BuildLinks(int n){//n: number of column
  p[head].L = n;
  for (int i=1; i<=n; ++i){
    addLR(i-1,i);
    p[i].U = i; p[i].D = i; p[i].S = 0;
  }
  p[n].R = head;
  top = n;
  for (int i=1; i<=line; i++){ // add new node
    int first = 0;
    for (int j=1; j<=n; ++j)
      if (tb[i][j]){
        p[++top].col = j;
        p[top].row = i;
        int x = p[j].U;
        addUD(x,top); addUD(top,j);
        p[j].S++;
        if (first) addLR(top-1,top);
          else first = top;
      }
    if (first == 0) continue;
    addLR(top,first);
  }
}
int H(){
  memset(h,false,sizeof(h));
  int rest = 0;
  for (int c = p[head].R; c != head ; c = p[c].R) {
    if(!h[c]){
      rest ++;
      h[c] = true;
      for (int i = p[c].D ; i != c ; i = p[i].D)
        for(int j = p[i].R ; j != i ; j = p[j].R)
          h[p[j].col] = true;
    }
  }
  return rest;
}
void Cover(int k){
  for (int i = p[k].D; i != k; i = p[i].D){
    p[p[i].L].R = p[i].R;
    p[p[i].R].L = p[i].L;
  }
}
void Release(int k){
  for (int i=p[k].U; i != k; i = p[i].U){
    p[p[i].L].R = i;
    p[p[i].R].L = i;
  }
}
void search(int step){
  if (p[head].R == head){
    if (step < ans) ans=step;
    return;
  }
  if (step + H() >= ans) return;
  int c = head,min = maxsize;
  for (int i=p[head].R; i != head; i = p[i].R)
    if (p[i].S < min)
      min = p[i].S; c = i;
  for (int i=p[c].D; i != c; i = p[i].D){
    Cover(i);
    for (int j=p[i].R; j != i; j = p[j].R)
      Cover(j);
    search(step+1);
    for (int j=p[i].L; j != i; j = p[j].L)
      Release(j);
    Release(i);
  }
}
while (scanf("%d%d",&line,&m)!=EOF) { // main()
  memset(p,0,sizeof(p));
  memset(tb,false,sizeof(tb));
  top=0;
  int x,y;
  for (int i=1;i<=line;i++) tb[i][i]=true;
  for (int i=1;i<=m;i++) {
    scanf("%d%d",&x,&y);
    tb[x][y]=tb[y][x]=true;
  }
  BuildLinks(line);
  ans=line;
  search(0);
  printf("%d\n",ans);
}
```

6.4 Fast-Longest-Common-Subsequence

```
<tex>complexity $O(N^{2}/64)$</tex>
typedef unsigned long long LL;
LL cmask[26][1111];
LL f[2][1111], X[1111];
int solve(char *A, char *B, int n, int m) {
  memset(cmask, 0, sizeof cmask);
  for (int i = 0; i < n; ++i) cmask[A[i] - 'a'][i >> 6] |=
    1ULL << (i & 63);
  memset(f, 0, sizeof f);
  int L = (n - 1) / 64 + 1;
  for (int i = 0; i < m; ++i) {
    int id = B[i] - 'a';
    int cur = i & 1, nxt = cur ^ 1;
    for (int j = 0; j < L; ++j) X[j] = f[cur][j] | cmask[
      id][j];
    for (int j = 0, x = 1; j < L; ++j) {
      int y = f[cur][j] >> 63 & 1;
      f[cur][j] <<= 1; f[cur][j] |= x;
      x = y;
    }
    memcpy(f[nxt], X, sizeof X);
    for (int j = 0, x = 0; j < L; ++j) {
      LL t = f[cur][j] + x;
      x = (f[nxt][j] < t);
      f[nxt][j] -= t;
      f[nxt][j] ^= X[j];
      f[nxt][j] &= X[j];
    }
  }
  int ans = 0;
  for (int i = 0; i < L; ++i) ans += __builtin_popcountll(f[m & 1][i]);
  return ans;
}
```

6.5 Java References

```
// DecimalFormat if you want to output double with specific precision
PrintStream fout =
new PrintStream(new FileOutputStream("file.out"));
BufferedInputStream fin =
new BufferedInputStream(new FileInputStream("file.in"));
BufferedReader reader =
new BufferedReader(new InputStreamReader(System.in));
Scanner in=new Scanner(fin);
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.parseDouble .....
}
```

6.6 Josephus

约瑟夫环，避免有人无聊，某种奇怪的实现等等加上。

```
int josephus(int n,int m,int k) {
  int x = -1;
  for(int i = n - k + 1;i <= n;i++) x = (x + m) % i;
  return x;
}
int invJosephus(int n,int m,int x) {
  for(int i = n;;i--) {
    if(x == i) return n-i;
    x = (x - m % i + i) % i;
  } assert(false);
}
```

6.7 Non-empty subsets

```
for (int sub=mask; sub>0; sub=(sub-1) & mask)
```

6.8 YY-MM-DD to date

```
int days(int y, int m, int d) {
  if (m < 3) y--, m += 12;
  return 365*y + y/4 - y/100 + y/400 + (153*m+2)/5 + d;
}
```

6.9 nCk Iterator

升序枚举 C(n,k) 的所有组合。

```
for(int comb = (1 << k) - 1;comb < (1 << n);) {
  { // next
    int x = comb & -comb;
    int y = comb + x;
    comb = ( (unsigned)((comb & ~y) / x) >> 1 ) | y;
  }
}
```

7 Tips

7.1 Integral

- $\int \frac{\mathrm{d}x}{\sqrt{a^2-x^2}} = \arcsin\frac{x}{a} + C$
- $\int \frac{\mathrm{d}x}{\sqrt{x^2\pm a^2}} = \ln|x+\sqrt{x^2\pm a^2}| + C$
- $\int \frac{\mathrm{d}x}{x^2-a^2} = \frac{1}{2a}\ln|\frac{x-a}{x+a}| + C$
- $\int \frac{\mathrm{d}x}{x^2+a^2} = \frac{1}{a}\arctan\frac{x}{a} + C$
- $\int \sqrt{a^2-x^2}\mathrm{d}x = \frac{1}{2}x\sqrt{a^2-x^2} + \frac{a^2}{2}\arcsin\frac{x}{a} + C$
- $\int \sqrt{x^2\pm a^2}\mathrm{d}x = \frac{1}{2}(x\sqrt{x^2\pm a^2} \pm a^2\ln|x+\sqrt{x^2\pm a^2}|) + C$

7.2 Primes

100003, 200003, 300007, 400009, 500009, 600011, 700001, 800011, 900001, 1000003, 2000003, 3000017, 4100011, 5000011, 6000011, 7000003, 8000009, 9000011, 10000019, 20000003, 50000017, 50100007, 100000007, 100200011, 200100007, 200100031, 250000019, 1000000007, 10000000019, 100000000003, 1000000000039, 10000000000031, 100000000000037, 1000000000000061, 10000000000000003, 100000000000000003, 1000000000000000009, 1000000000000000031

7.3 Constants Table

n	$n!$	$\binom{n}{n/2}$	$LCM(1\ldots n)$	P_n	B_n
2	2	2	2	2	2
3	6	3	6	3	5
4	24	6	12	5	15
5	120	10	60	7	52
6	720	20	60	11	203
7	5040	35	420	15	877
8	40320	70	840	22	4140
9	362880	126	2520	30	21147
10	3628800	252	2520	42	115975
11	39916800	462	27720	56	678570
12	479001600	924	27720	77	4213597
13		1716	360360	101	27644437
14		3432	360360	135	190899322
15		6435	360360	176	1382958545
16		12870	720720	231	
17		24310	12252240	297	
18		48620	12252240	385	
19		92378	232792560	490	
20		184756	232792560	627	
25		5200300		1958	
30		155117520		5604	
40				37338	
50					
70					

7.4 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));
  }
}
```

7.5 stack

```
register char *_sp __asm__("rsp"); // esp / sp
int main(void) {
  const int size = 64*1024*1024;
  static char *sys, *mine(new char[size]+size-4096);
  sys = _sp; _sp = mine; mmain(); _sp = sys;
  return 0;
}
```