爱吃吗/爱吃泡菜 Reference Material

Fudan University / KuroRekishi guys

May 6, 2015

目录

1 Data Structure

1.1 Partition Tree

```
int val[19][100100], lsize[19][100100], sorted[100100]; // sorted: [1,N], sorted needed

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

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

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

1.2 BIT Kth

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

1.3 Another Splay

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

struct Splay {
  int root;
  void update(int t) {
    pt[t].s = pt[pt[t].ch[0]].s + pt[pt[t].ch[1]].s + abs(pt[t].v);
    pt[t].nr = max(0,pt[pt[t].ch[0]].nr + pt[t].v - pt[pt[t].ch[1]].nl) + pt[pt[t].ch[1]].nr;
    pt[t].nl = max(0,pt[pt[t].ch[1]].nl - pt[t].v - pt[pt[t].ch[0]].nr) + pt[pt[t].ch[0]].nl;
    if (pt[t].v > 0) { // node of boy
      pt[t].ans = pt[pt[t].ch[0]].ans + pt[pt[t].ch[1]].ans + min(pt[pt[t].ch[0]].nr + pt[t].v,pt[pt[t].ch[1]].nl);
    } else { // otherwise
      pt[t].ans = pt[pt[t].ch[0]].ans + pt[pt[t].ch[1]].ans + min(pt[pt[t].ch[0]].nr,pt[pt[t].ch[1]].nl - pt[t].v);
    }
  }
  void zig(int x,bool w) {
    int y = pt[x].f; if (root == y) root = x;
    pt[y].ch[!w] = pt[x].ch[w]; if (pt[x].ch[w]) pt[pt[x].ch[w]].f = y;
    pt[x].f = pt[y].f; if (root != x) pt[pt[y].f].ch[y == pt[pt[y].f].ch[1]] = x;
    pt[x].ch[w] = y; pt[y].f = x; update(y);
  }
  void splay(int x) {
    while (x != root) {
      if (pt[x].f == root) zig(x,x == pt[pt[x].f].ch[0]);
      else {
        int y = pt[x].f, z = pt[y].f;
        if (y == pt[z].ch[0]) if (x == pt[y].ch[0]) { zig(y,1); zig(x,1); } else { zig(x,0); zig(x,1); }
        else if (x == pt[y].ch[0]) { zig(x,1); zig(x,0); } else { zig(y,0); zig(x,0); }
      }
    } update(x);
  }
  void splay(int x,int f) {
    while (pt[x].f != f) {
      if (pt[pt[x].f].f == f) zig(x,x == pt[pt[x].f].ch[0]);
      else {
        int y = pt[x].f, z = pt[y].f;
        if (y == pt[z].ch[0]) if (x == pt[y].ch[0]) { zig(y,1); zig(x,1); } else { zig(x,0); zig(x,1); }
        else if (x == pt[y].ch[0]) { zig(x,1); zig(x,0); } else { zig(y,0); zig(x,0); }
      }
    } update(x);
  }
  int selFlag;
  int sel(int Key) {
    int t = root;
    while (1) {
      int ls = pt[pt[t].ch[0]].s;
      if (ls < Key && ls + abs(pt[t].v) >= Key) {
        selFlag = Key - ls;
        return t;
      }
      if (Key <= ls) t = pt[t].ch[0]; else {
        Key -= ls + abs(pt[t].v);
        t = pt[t].ch[1];
      }
    } return t;
  }
  void Del(int t) {
    while (pt[t].ch[0] + pt[t].ch[1]) if (pt[t].ch[0]) zig(pt[t].ch[0],1); else zig(pt[t].ch[1],0);
    if (root == t) {
      root = 0; return ;
    }
    pt[pt[t].f].ch[t == pt[pt[t].f].ch[1]] = 0; splay(pt[t].f);
  }
  int bound(int x,bool w) {
    splay(x);
    int ret = pt[x].ch[w];
    while (pt[ret].ch[!w]) ret = pt[ret].ch[!w];
    return ret;
  }
  PII Split(int t,int pos) {// break node t at postion pos
    int L = bound(t,0), R = bound(t,1); Del(t);
    splay(L,0); splay(R,L);
    int s = abs(pt[t].v); int c = (pt[t].v > 0) ? 1 : -1;
    if (pos >= 1) {
      pt[++now].Init(c * (pos),R); pt[R].ch[0] = now;
      splay(now); L = now; splay(R,L);
    }
    if (pos < abs(pt[t].v)) {
      pt[++now].Init(c * (abs(pt[t].v) - pos),R); pt[R].ch[0] = now;
      splay(now); R = now;
    }
    return MP(L,R);
  }
```

```
}Tab;
```

1.4 KD Tree

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

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

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

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

inline KDNODE* AllocNode() { memset(NPTop,0,sizeof(KDNODE)); return NPTop++; }
inline ll PDist(const POINT& a,const POINT& b) { return sqr((ll)(a.x-b.x))+sqr((ll)(a.y-b.y)); }

POINT pnt[111111];
KDNODE* KDTree_Build(int l,int r,int depth=0) {
  if(l >= r) return NULL;

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

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

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

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;
}
```

1.5 LCT_Path

```
// Link-Cut-Tree模板，支持路径查询、路径修改、link、cut、不支持子树操作。
//MaxNode 节点数
//用的时候在外面对输入树的每个节点i调用一下pt[i].Init()即可。
struct node {
  int v, f, ch[2], s;
  node() {}
  void Init(int _v, int _f) {
    v = _v; f = _f;
    ch[0] = ch[1] = 0;
    s = 1;
  }
}pt[MaxNode];

struct LCT {
  struct node{
    int f,ch[2],sum,v,s; bool rev,isroot;
    void Make_Rev() {
      rev = !rev; swap(ch[0],ch[1]);
    }
    void Init(int _v, int _f) {
      f = _f; ch[0] = ch[1] = 0; isroot = true; rev = false;
      sum = v = _v;
      s = 1;
    }
  }pt[MaxNode];
  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 = false;
    }
  }
  void updata(int t) {
    pt[t].s = pt[pt[t].ch[0]].s + pt[pt[t].ch[1]].s + 1;
    pt[t].sum = pt[pt[t].ch[0]].sum + pt[pt[t].ch[1]].sum + pt[t].v;
  }
  void zig(int x,bool w) {
    int y = pt[x].f; 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; updata(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); }
      }
    } updata(x);
```

```
  }
  int  access(int x,int cas) { // -1 for reverse
    int p = 0, q = x, ret = 0;
    while (q) {
      splay(q); push(q);
      if (pt[q].f == 0) {
        ret = pt[pt[q].ch[1]].sum + pt[p].sum;
        if (pt[q].ch[1]) {
          int t = pt[q].ch[1];
          if (!p) {
            while (pt[t].ch[0]) t = pt[t].ch[0];
            ret -= pt[t].v;
          }
        }
        if (p) {
          int t = p;
          if (!pt[q].ch[1]) {
            while (pt[t].ch[0]) t = pt[t].ch[0];
            ret -= pt[t].v;
          }
        }
      }
      pt[pt[q].ch[1]].isroot = true; pt[p].isroot = false;
      pt[q].ch[1] = p; updata(q); p = q; q = pt[q].f;
    }
    splay(p);
    if (cas == 0) ret = pt[p].s;
    if (cas == -1) pt[p].Make_Rev();
    return ret;
  }
  int root(int x) {
    access(x,0);
    splay(x);
    while (pt[x].ch[0]) x = pt[x].ch[0];
    return x;
  }
  void Link(int x,int y) {
    access(x, -1); splay(x); pt[x].f = y;
  }
  bool Son(int x,int y) {
    access(x,0); splay(y);
    return pt[y].ch[1] == x;
  }
  void Cut(int x,int y) {
    if (Son(x,y)) swap(x,y);
    access(y,0); splay(x); pt[y].isroot = true; pt[x].ch
    [1] = 0; pt[y].f = 0;
  }
}Tree;
```

1.6 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;
```

1.7 Light-Heavy Decomposition

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

```
int BlockRoot[MAXN], NodeID[MAXN], IndexToNode[MAXN],
TreeSize[MAXN], Depth[MAXN], fa[MAXN];
int NodeID_Out[MAXN]; // 离开节点的时候的dfs序
int HeavyChild[MAXN]; // 0 if not set
int idx = 0;
int dfs_size(int x) {
  TreeSize[x] = 1;
  for(EDGE* e = E[x];e;e = e->Next) {
    int y = e->y;
```

```
    if(y == fa[x]) continue;

    fa[y] = x; Depth[y] = Depth[x]+1;
    dfs_size(y);
    TreeSize[x] += TreeSize[y];
    if(TreeSize[HeavyChild[x]] < TreeSize[y]) HeavyChild[x] = y;
  }
  return 0;
}
int dfs_lh(int x,int block) {
  BlockRoot[x] = block; NodeID[x] = ++idx; IndexToNode[idx] = x;
  if(HeavyChild[x]) dfs_lh(HeavyChild[x],block);
  for(EDGE* e = E[x];e;e = e->Next) {
    int y = e->y;
    if(y == fa[x] || y == HeavyChild[x]) continue;
    dfs_lh(y,y);
  }
  NodeID_Out[x] = idx;
  return 0;
}
int Decomposition(int s,int N) {
  idx = 0; fa[s] = 0;
  memset(HeavyChild,0,sizeof(HeavyChild[0])*(N+10));
  dfs_size(s); dfs_lh(s,s);
  return 0;
}

// 如果需要非递归的，一点提示，bfs都会写，后面的：
for(int i = qend-1;i >= 0;i--) {
  int x = Queue[i];
  if(x == HeavyChild[fa[x]]) continue;
  int t = x;
  while(t) {
    BlockRoot[t] = x;
    NodeID[t] = ++idx;
    t = HeavyChild[t];
  }
}

// 参考用爬树过程
int ColorNode(int x,int y,int nc) {
  while(1) {
    if(Depth[BlockRoot[x]] > Depth[BlockRoot[y]]) swap(x,y);

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

1.8 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;
}
```

1.9 Persistent Interval Tree

As elegant as possible.

```
class SEGNODE {
public:
  SEGNODE() { memset(this,0,sizeof(SEGNODE)); maxval = -23333; }
  SEGNODE(SEGNODE* _orig) { if(_orig) memcpy(this,_orig,sizeof(SEGNODE)); else { memset(this,0,sizeof(SEGNODE)); maxval = -23333; } }

  SEGNODE *left, *right;
  int maxval;

  SEGNODE* dup() { return new SEGNODE(this); }

  SEGNODE* modify(int x,int val,int tl,int tr) {
    SEGNODE* self = dup();
    if(x == tl && tl == tr) {
      self->maxval = max(self->maxval,val);
      return self;
    }

    int mid = (tl+tr)/2;
    if(x <= mid) self->left = self->left->modify(x,val,tl,mid);
    else if(x > mid) self->right = self->right->modify(x,val,mid+1,tr);

    /* UPDATE */
    self->maxval = -23333;
```

```
    if(self->left) self->maxval = max(self->maxval,self->left->maxval);
    if(self->right) self->maxval = max(self->maxval,self->right->maxval);
    ////////////
    return self;
  }

  int query(int l,int r,int tl,int tr) {
    if(!this) return -23333;
    if(l <= tl && tr <= r) return maxval;

    int mid = (tl+tr)/2;
    int ans = -23333;
    if(l <= mid) ans = max(ans,left->query(l,r,tl,mid));
    if(r > mid) ans = max(ans,right->query(l,r,mid+1,tr));
    return ans;
  }
};
```

2 Graph

2.1 Bridge

无向图求桥，支持重边。直接拆掉桥就是边 BCC。

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

2.2 Cut Point

求割点/点 BCC，同样支持重边。BCCId 为某条边在哪个 BCC 内。

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

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

2.3 MMC (Karp)

$O(nm+n^2)$ 最大平均权值环需要存边但是不需要边表。

```
int d[677][677] = {0};
double Karp(int n,int m) {
  memset(d,0,sizeof(d));
  // init all d[0][i] with 0 if no memset or reversing

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

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

2.4 LCA (Tarjan)

$O(n)$ 仅在需要顺手维护点别的东西的时候用。

```
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];
}
```

2.5 LCA (sqr)

倍增 LCA $O(n\log n)$ 只要维护的是树，可以动态添加

```
int fa[MAXN][LOGMAXN]; int depth[MAXN];
int lca(int x,int y) {
  if(depth[x] < depth[y]) swap(x,y);
  int delta = depth[x]-depth[y];
  for(int i = 0;i < LOGMAXN;i++) if(delta&(1<<i)) x = fa[x][i];
  for(int i = LOGMAXN-1;i >= 0;i--) if(fa[x][i] != fa[y][i]) { x = fa[x][i]; y = fa[y][i]; }
  if(x != y) x = fa[x][0];
  return x;
}
```

2.6 Stable Marriage

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

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

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

```
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                                    ;}}}}}
```

2.7 Arborescence

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

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

      if(i == root) continue;
      if(minIn[i] == MAXW) return false; // does not exist
      ans += minIn[i];
    }
    int SCC[MAXN] = {0};
    int vis[MAXN] = {0};
    prev[root] = root;
    int sccidx = 0; int vidx = 0;
    for(int i = 0;i < n;i++) {
      if(vis[i]) continue;
      int x = i; vidx++;
      while(!vis[x]) {
        vis[x] = vidx;
        SCC[x] = sccidx++;
        x = prev[x];
      }
      if(vis[x] == vidx) { // circle
        int ori = x;
        sccidx = SCC[x]+1;
        do {
          SCC[x] = SCC[ori];
          x = prev[x];
        } while(x != ori);
      }
    }
    if(sccidx == n) break; // found
    // rebuild
    EDGE* TE[MAXN] = {0};
    for(int i = 0;i < n;i++)
      for(EDGE* e = E[i];e;e = e->Next)
        if(SCC[i] != SCC[e->y]) insert_edge(SCC[i],SCC[e->y],e->w,TE);
    memcpy(E,TE,sizeof(E));

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

2.8 Stoer_Wagner

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

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

2.9 MaxFlow (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。

@Yuege: 请，一定，要，用，dfs，找割集。

```cpp
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;
  }
};
```

2.10 KM

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

bool DFS(int x) {
  visx[x] = 1;
  for (int y = 1;y <= ny;y++) {
    if (visy[y]) continue;
    int t = lx[x] + ly[y] - w[x][y];
    if (t == 0) {
      visy[y] = 1;
      if (link[y] == -1 || DFS(link[y])) {
        link[y] = x;
```

```
        return true;
      }
    }
    else if (slack[y] > t) slack[y] = t;
  }
  return false;
}
void KM() {
  int i,j;
  memset (link,-1,sizeof(link));
  memset (ly,0,sizeof(ly));
  for (i = 1;i <= nx;i ++) for (j = 1,lx[i] = -INF;j <= ny;j ++)
    if (w[i][j] > lx[i]) lx[i] = w[i][j];

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

2.11 Hopcroft-Karp

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

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

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

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

2.12 Edmonds matching algorithm

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

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

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

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

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

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

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

2.13 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();
    }
  }
};
```

2.14 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;
}
```

2.15 Planar Gragh

2.15.1 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。

2.15.2 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。

2.15.3 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。

2.16 Prufer Code

2.16.1 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。

2.16.2 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。

2.16.3 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 $n_1^{n_2-1} * n_2^{n_1-1}$。

2.17 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;
}

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);
```

2.18 Spanning Tree Count

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

2.19 Maximum Clique

最大团，复杂度上限大概是 $O(3^{\frac{n}{3}})$ 左右。G 是邻接矩阵，dp_i 表示由 i 到 $n-1$ 的子图构成的最大团点数，剪枝用。adj 中存放的是与 i 邻接且标号比 i 大的顶点。顺便可以保证方案的字典序最小。另外对 adj 进行压位有非常良好的效果。

```
const int MAXN = 55;
int G[MAXN][MAXN];
int ans = 0; int plan[MAXN]; int cur[MAXN]; int dp[MAXN];

bool dfs(int* adj,int adjCnt,int curClique) {
  if(!adjCnt && curClique > ans) { ans = curClique; return true; } // memcpy(plan,cur,sizeof(plan[0])*ans); if plan needed
  if(!adjCnt) return false;
  int nextAdj[MAXN];
  for(int i = 0;i < adjCnt;i++) {
    if(curClique + adjCnt - i <= ans) return false;
    if(curClique + dp[adj[i]] <= ans) return false;
    //cur[curClique] = adj[i];
    int nextAdjCnt = 0;
    for(int j = i+1;j < adjCnt;j++) if(G[adj[i]][adj[j]]) nextAdj[nextAdjCnt++] = adj[j];
    if(dfs(nextAdj,nextAdjCnt,curClique+1)) return true;
  }
  return false;
}

int maximumClique(int n) {
  ans = 0; memset(dp,0,sizeof(dp));

  int adj[MAXN];
  for(int i = n-1;i >= 0;i--) {
    cur[0] = i;
    int adjCnt = 0;
    for(int j = i+1;j < n;j++) if(G[i][j]) adj[adjCnt++] = j;
    dfs(adj,adjCnt,1);
    dp[i] = ans;
  }
  return ans;
```

```
}
```

2.20 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);
```

2.21 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;
}
```

2.22 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;
}
```

2.23 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;
}
```

2.24 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;

  for(int k = 0;k < n;k++) for(int i = 0;i < n;i++) for(
  int j = 0;j < n;j++)
    G[i][j] = min(G[i][j],G[i][k] + G[k][j]);

  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]];
}
```

2.25 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());
}
```

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 Aho-Corasick

fail 指向与当前串的后缀相等的前缀最长的节点

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

3.5 Manacher

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

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

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

3.6 Suffix Automaton

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

```
      nfa->len = x->len+1;
      bro->fa = t->fa = nfa;

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

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

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

3.7 Suffix Array

```
int aa[222222], 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]);
}
```

3.8 Palindrome Tree

神奇的东西，所谓的 Palindrome Tree 其实是每个点表示了一个回文子串，而边则是表示在两侧同时添加上这个字母可以得到的新回文子串。从点 u 到点 w 的 suffix link 表示 w 是 u 的所有不是 u 本身的后缀中最长的回文子串。这个所谓的“Tree”实际上有两个根。一个表示 -1 长度的串，用于表示只有一个字母的新回文串的产生，一个表示空串。两个根的 suffix link 都指向 -1 根。有编号大的点就是拓扑序小的点这个性质。

一些应用：

1. 统计加入一个字母的时候增加了多少个新的（不同的）回文串：看看加入字母的时候多出来几个点就行了。答案只可能是 0 或 1。
2. 计算回文子串个数：这个 Manacher 可以做，但是用 Palindrome Tree 的话更有趣。注意到 Suffix Link 关系是棵树（两个根两棵树），对每个点维护它到根的连接数，然后对于新加入的点加上它的连接数即可（考虑 Suffix Link 关系的意义，显然）。
3. 计算每个不同的子回文串的出现次数：基本同上，注意到对于新加入的点它是对本身和 Suffix Link 上的所有点贡献了 1 的答案，于是上推一遍即可。

$O(n \log \sum)$，可以假装自己是线性的，像 SAM 一样。

```
template<int MAXN, int SIGMA> class PalindromeTree {
public:
  struct TNODE {
    int len; /* Extra fields */ // int cnt;
    TNODE* suffLink, *child[SIGMA];
  };

  char text[MAXN]; int textLen;
  TNODE NPool[MAXN+2];
  TNODE* NPTop, *Last, *Root_Guard, *Root_Empty;

  PalindromeTree() { reset(); }
  int reset() {
    NPTop = NPool;
    Root_Guard = NPTop++; Root_Empty = NPTop++;
    memset(Root_Guard,0,sizeof(TNODE));
    memset(Root_Empty,0,sizeof(TNODE));
    Root_Guard->len = -1; Root_Empty->len = 0;
    Root_Guard->suffLink = Root_Empty->suffLink = Root_Guard;
    Last = Root_Empty;
    return 0;
  }

  // return 0 or 1, indicating new (distinct) palindrome substring count.
  int feed(int ch) {
```

```
    text[textLen++] = ch;

    TNODE* cur = Last;
    while(textLen-1-1-cur->len < 0 || text[textLen-1-1-cur->len] != ch) cur = cur->suffLink;
    if(cur->child[ch]) { Last = cur->child[ch]; return 0; }

    TNODE* x = NPTop++;
    memset(x,0,sizeof(TNODE));
    Last = x;
    x->len = cur->len + 2;
    cur->child[ch] = x;
    if(x->len == 1) { // from -1 root
      x->suffLink = Root_Empty;
    } else {
      cur = cur->suffLink;
      while(textLen-1-1-cur->len < 0 || text[textLen-1-1-cur->len] != ch)
        cur = cur->suffLink;
      x->suffLink = cur->child[ch];
    }
    // maintain additional field here.
    return 1;
  }
};
```

3.9 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);
  }
};
```

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 三角形外接圆半径

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

4.1.6 Pick's theorem

顶点全为整点的简单多边形的面积定为 S，边上的整点数为 B，内部的整点数为 I，则:

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

4.1.7 超球坐标系

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

4.1.8 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

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

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

4.1.9 立体角公式

ϕ

二面角

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

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

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

4.1.10 四面体体积公式

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

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

其中

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

4.2 2D-Geometry

```
struct Point {
  double x, y;
  Point (){}
  Point(double x, double y) : x(x), y(y) {}
  Point operator - (const Point &b) { return Point(x - b.x , y - b.y); }
  Point operator + (const Point &b) { return Point(x + b.x , y + b.y); }
  Point operator * (const double &b) { return Point(x * b, y * b); }
  Point operator / (const double &b) { return Point(x / b, y / b); }
  Point rot90(int t) { return Point(-y, x) * t; }
  Point rot(double ang) { return Point(x * cos(ang) - y * sin(ang), x * sin(ang) + y * cos(ang)); }
  double ang() { double res = atan2(y, x); if (dcmp(res) < 0) res += pi * 2; return res; }
  double operator * (const Point &b) { return x * b.y - y * b.x; }
  double operator % (const Point &b) { return x * b.x + y * b.y; }
  double len2() { return x * x + y * y; }
  double len() { return sqrt(x * x + y * y); }
};
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).len2()));
}
// 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) % (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).len2();
  double d = x * x - y * ((p1 - c).len2() - r * r);
  if (d < -eps) return res;
  if (d < 0) d = 0;
  Point q1 = p1 - ((p2 - p1) * (x / y));
  Point q2 = (p2 - p1) * (sqrt(d) / y);
  res.push_back(q1 - q2);
  res.push_back(q1 + q2);
  return res;
}
// 圆与圆的交点
vector<Point> getCC(Point c1, double r1, Point c2, double r2) {
  vector<Point> res;
  double x = (c1 - c2).len2();
  double y = ((r1 * r1 - r2 * r2) / x + 1) / 2;
  double d = r1 * r1 / x - y * y;
  if (d < -eps) return res;
  if (d < 0) d = 0;
  Point q1 = c1 + (c2 - c1) * y;
  Point q2 = ((c2 - c1) * sqrt(d)).rot90();
  res.push_back(q1 - q2);
  res.push_back(q1 + q2);
  return res;
}
// 两圆公共面积.
double areaCC(Point c1, double r1, Point c2, double r2) {
  double d = (c1 - c2).len();
  if (r1 + r2 < d + eps) return 0;
  if (d < fabs(r1 - r2) + eps) {
    double r = min(r1, r2);
    return r * r * pi;
  }
  double x = (d * d + r1 * r1 - r2 * r2) / (2 * d);
  double t1 = acos(x / r1);
  double t2 = acos((d - x) / r2);
  return r1 * r1 * t1 + r2 * r2 * t2 - d * r1 * sin(t1);
}
// ccenter 返回{p1, p2, p3} 的外接圆圆心, formula
// 四点在同一圆周
bool onCir(Point p1, Point p2, Point p3, Point p4) {
  if (fabs((p2 - p1) * (p3 - p1)) < eps) return true;
  Point c = ccenter(p1, p2, p3);
  return fabs((c - p1).len2() - (c - p4).len2()) < eps;
}
//两圆公切线, 先返回内公切线, 后面是外公切线
vector<Line> getLineCC(Point c1, double r1, Point c2, double r2) {
  vector<Line> res;
  double d = (c1 - c2).len();
  if (fabs(d - r1 - r2) < eps) {
    Point o = (c1 + c2) * 0.5;
    res.push_back(Line(o, o + (c1 - c2).rot90()));
    res.push_back(res[res.size() - 1]);
  } else {
    double ang = acos((r1 + r2) / d);
    res.push_back(Line(c1 + ((c2 - c1) * (r1 / d)).rot(ang), c2 + ((c1 - c2) * (r2 / d)).rot(ang)));
    ang = -ang;
    res.push_back(Line(c1 + ((c2 - c1) * (r1 / d)).rot(ang), c2 + ((c1 - c2) * (r2 / d)).rot(ang)));
  }
  double ang = acos((r2 - r1) / d);
  res.push_back(Line(c1 + ((c1 - c2) * (r1 / d)).rot(ang), c2 + ((c1 - c2).rot(ang) * (r2 / d))));
  ang = -ang;
  res.push_back(Line(c1 + ((c1 - c2) * (r1 / d)).rot(ang), c2 + ((c1 - c2).rot(ang) * (r2 / d))));
  return res;
}
// 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

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

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

4.4 Convex Hull

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

4.5 Euclid Nearest

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

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

4.6 Minimal Circle Cover

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

POINT pt[100010] = {0};

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

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

4.7 3D Convex Hull

```
struct Hull3D {
  struct Plane {
    int a, b, c;
    bool ok;
    Plane(){}
    Plane(int a, int b, int c, bool ok)
      : a(a), b(b), c(c), ok(ok) {}
  };
  int n, tricnt;        //初始点数
  int vis[MaxN][MaxN];  //点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;
```

4.8 Rotate Carbin

返回凸包上最远点对距离

```
double RC(int N) {
  double ans = 0.0;
  Hull[N] = Hull[0];
  int to = 1;
  for(int i = 0;i < N;i++) {
    while((Hull[i+1]-Hull[i])*(Hull[to]-Hull[i]) < (Hull[i+1]-Hull[i])*(Hull[to+1]-Hull[i])) to = (to+1)%N;
    ans = max(ans,(Hull[i]-Hull[to]).len2());
    ans = max(ans,(Hull[i+1]-Hull[to]).len2());
  }
  return sqrt(ans);
}
```

4.9 Halfplane

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

```
struct Segment {
  Point s, e;
  double angle;
  Segment(){}
  Segment(Point s, Point e)
    : s(s), e(e) {
      angle = atan2(e.y - s.y, e.x - s.x);
    }
} ;
Point get_intersect(Segment s1, Segment s2) {
  double u = xmul(s1.s, s1.e, s2.s);
  double v = xmul(s1.e, s1.s, s2.e);
  Point t;
  t.x = (s2.s.x * v + s2.e.x * u) / (u + v);
  t.y = (s2.s.y * v + s2.e.y * u) / (u + v);
  return t;
}
bool cmp(Segment a, Segment b) {
  if (dcmp(a.angle - b.angle) == 0) return dcmp(xmul(a.s, a.e, b.s)) < 0;
  return dcmp(a.angle - b.angle) < 0;
  return 0;
}
bool IsParallel(Segment P, Segment Q) {
  return dcmp((P.e - P.s) * (Q.e - Q.s)) == 0;
}
Segment deq[MaxN];
int HalfPlaneIntersect(Segment seg[], int n, Point hull[]) {
  sort(seg, seg + n, cmp);
  int tmp = 1;
  for (int i = 1; i < n; ++i) {
    if (dcmp(seg[i].angle - seg[tmp - 1].angle) != 0) {
      seg[tmp++] = seg[i];
    }
  }
  n = tmp;
  deq[0] = seg[0]; deq[1] = seg[1];
  int front = 0, tail = 1;
  for (int i = 2; i < n; ++i) {
    if(IsParallel(deq[tail], deq[tail-1]) || IsParallel(deq[front], deq[front+1])) return 0;
    while (front < tail && dcmp(xmul(seg[i].s, seg[i].e, get_intersect(deq[tail], deq[tail - 1]))) < 0) --tail;
    while (front < tail && dcmp(xmul(seg[i].s, seg[i].e, get_intersect(deq[front], deq[front+1]))) < 0) ++front;
    deq[++tail] = seg[i];
  }
  while(front < tail && xmul(deq[front].s, deq[front].e, get_intersect(deq[tail], deq[tail-1])) < -eps) tail--;
  while(front < tail && xmul(deq[tail].s, deq[tail].e, get_intersect(deq[front], deq[front+1])) < -eps) front++;
  int cnt = 0;
  deq[++tail] = deq[front];
  for (int i = front; i < tail; ++i) hull[cnt++] = get_intersect(deq[i], deq[i+1]);
  return cnt;
}
```

4.10 Simpson

如果怀疑被畸形数据了并且时间很多，试试

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

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

4.11 Circle Union

圆的面积并. 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;
}
```

4.12 Polygon Union

多边形面积并 $O(n^2logn)$

```
struct Polygon {
  int M;
  vector<Point> p;
  void init() {
    p.resize(M);
    for (int i = 0; i < M; ++i) p[i].init();
    if (dcmp((p[1]-p[0]) * (p[2]-p[1])) < 0) reverse(p.begin(), p.end());
    p.push_back(p[0]);
  }
  Point& operator [](const int &i) { return p[i]; }
} poly[MaxN];
double xmul(Point a, Point b, Point c) { return (b-a)*(c-a); }
pair<double, int> keys[MaxN];
double get(Point a, Point b, Point c) {
  double t;
  if (fabs(a.x-b.x) > eps) t = (c.x-a.x)/(b.x-a.x);
  else t = (c.y-a.y)/(b.y-a.y);
  t = max(min(t,1.0),0.0);
  return t;
}
double solve(int n) {
  double res = 0;
  for (int i = 0; i < n; ++i)
  for (int x = 0; x < poly[i].M; ++x) {
    int keysize = 0;
    for (int k = 0; k < n; ++k) if (k != i)
    for (int y = 0; y < poly[k].M; ++y) {
      int t1 = dcmp(xmul(poly[i][x], poly[i][x+1], poly[k][y]));
      int t2 = dcmp(xmul(poly[i][x], poly[i][x+1], poly[k][y+1]));
      if (!t1 && !t2) {
        if (k < i && dcmp((poly[k][y+1]-poly[k][y])%(poly[i][x+1]-poly[i][x])) >= 0) {
          double d1 = get(poly[i][x], poly[i][x+1], poly[k][y]);
          double d2 = get(poly[i][x], poly[i][x+1], poly[k][y+1]);
          keys[keysize++] = make_pair(d1, 1);
          keys[keysize++] = make_pair(d2, -1);
        }
      } else if ((t1 >= 0 && t2 < 0) || (t1 < 0 && t2 >= 0)) {
        double d1 = xmul(poly[k][y], poly[k][y+1], poly[i][x]);
        double d2 = xmul(poly[k][y], poly[k][y+1], poly[i][x+1]);
        int t = 1; if (t2 >= 0) t = -1;
        keys[keysize++] = make_pair(max(min(d1/(d1-d2),1.),0.), t);
      }
    }
    sort(keys, keys + keysize);
    int cnt = 0;
    double s = 0, tmp = 0;
    bool f = 1;
    for (int j = 0; j < keysize; ++j) {
      cnt += keys[j].second;
      if (!cnt && !f) tmp = keys[j].first, f = 1;
      if (cnt && f) s += keys[j].first - tmp, f = 0;
    }
    s += 1. - tmp;
    res += (poly[i][x] * poly[i][x+1]) * s;
  }
  return res * 0.5;
}
```

5 Math

5.1 Formula

5.1.1 Catalan number

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

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

5.1.2 勾股数

n, m 互质且 m, n 至少有一个偶数则是苏勾股数

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

$$b = 2mn$$

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

5.1.3 容斥原理

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

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

5.1.4 Bell Number

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

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

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

5.1.5 第一类 Stirling 数

n 个元素的项目分作 k 个环排列的方法数目

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

5.1.6 第二类 Stirling 数

第二类 Stirling 数是 n 个元素的集定义 k 个等价类的方法数目

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

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

5.1.7 判断是否为二次剩余

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

5.1.8 级数展开式

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

5.1.9 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

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

5.1.10 概率论

条件概率公式

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

贝叶斯定理

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

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

5.1.11 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。

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

5.1.12 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。

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

5.1.13 威尔逊定理

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

5.2 Factorial-Mod

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

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

5.3 NTT

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

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

  int N;
  int a[MaxN], b[MaxN], c[MaxN];
  LL mul(LL x, LL y, LL z) {
    return (x * y - (LL) (x / (long double) z * y + 1e-3)
    * z + z) % z;
  }
  int crt(int x1, int x2) {
    return (mul(M1, x1, MM) + mul(M2, x2, MM)) % MM % P;
  }
  void NTT(int *A, int pm, int g) {
    if (g < 0) g = fpow(-g, pm - 2, pm);
    int pw = fpow(g, (pm - 1) / N, pm);
    for (int m = N, h; h = m / 2, m >= 2; pw = (LL) pw *
    pw % pm, m = h) {
      for (int i = 0, w = 1; i < h; ++i, w = (LL) w * pw %
       pm)
      for (int j = i; j < N; j += m) {
        int k = j + h, x = (A[j] - A[k] + pm) % pm;
        A[j] += A[k]; A[j] %= pm;
        A[k] = (LL) w * x % pm;
      }
    }
    for (int i = 0, j = 1; j < N - 1; ++j) {
      for (int k = N / 2; k > (i^=k); k /= 2);
      if (j < i) swap(A[i], A[j]);
    }
  }
  void solve(int *A, int *B, int *C, int n) {

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

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

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

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

5.4 DFT

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

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

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

5.5 Baby-Step-Giant-Step

```
namespace BSGS {
  #define MaxNode 1200007
  #define HMD 1000007
  struct Thash{
    PII has[MaxNode];int next[MaxNode],h[HMD],tot;
    void clr(){
      memset(h, 0, sizeof h);
    }
    void ins(int p,int pos){
      int vex = p % HMD;
      for(int z = h[vex]; z; z = next[z]) if(has[z].fi ==
      p) return;
      has[++tot] = MP(p,pos); next[tot] = h[vex]; h[vex] =
       tot;
    }
    int find(int p){
      if (p == 0) return -1;
```

```
      for(int z = h[p % HMD]; z; z = next[z]) if(has[z].fi
       == p) return has[z].se;
      return -1;
    }
  }Hash;
  void build(int y, int p){
    int m = 700000, now = 1;
    for (int i = 0; i <= m; ++i) {
      Hash.ins(now,i);
      now = (LL) now * y % p;
    }
  }
  int find(int y, int z, int p) {
    int D = fpow(y, m, p), now = 1;
    D = fpow(D, p - 2, p);
    for (int i = 0; i <= p / m + 1; ++i) {
      int t = Hash.find((long long)z * now % p);
      if (t + 1) return i * m + t;
      now=(long long) now * D % p;
    }
    return -1;
  }
};
```

5.6 CRT

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

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

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

5.7 Find-Square-Root

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

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

5.8 Integer-Partition

整数划分。五边形数 $\frac{3j^2-j}{2}$

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

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

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

5.9 Lattice Count

计算

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

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

```
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);
}
```

5.10 Linear Dependency

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];
    }
  }
}
```

5.11 Linear Eratosthenes Sieve

```
int MinDivi[MAXNUM], Prime[MAXNUM/10], 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]));
    }
  }
}
```

5.12 Miller-Rabin

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

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

const int LPrimes[] = {2,3,5,7,11,13,17,19,23};
int isPrime(ull n) {
  int result = 1;
  for(int i = 0;i < sizeof(LPrimes)/sizeof(int);i++) {
```

```
    if(LPrimes[i] >= n) break;
    result &= MillerRabin(LPrimes[i],n);
    if(!result) return result;
  }
  return result;
}
```

5.13 Modular Factorial

$n!\mod mod$ where $mod = p^k$. $O(p \log n)$

```
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;
}
```

5.14 Pollard-Rho

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

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

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

// call sort if sorted results needed.
void fullFactorize(ull n, vector<ull>& result) {
  if(n < 2) return;
  if(n%2 == 0) {
    result.push_back(2);
    while(n%2 == 0) n /= 2;
  }
  while(n != 1 && !isPrime(n)) {
    ull t = factorize(n);
    int cdvc = result.size();
    if(!isPrime(t)) fullFactorize(t, result);
    else result.push_back(t);
    for(int i = cdvc;i < result.size();i++) while(n % dv[i
    ] == 0) n /= dv[i];
  }
  if(n != 1) result.push_back(n);
}
```

5.15 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 $O(n \log n \log m)$

```
// 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;
}
```

5.16 Simplex

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

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

```
    p = 1e100;
    for (int i=0; i<m; i++) if (a[i][c]>eps) {
      v = a[i][n]/a[i][c];
      if (v<p) p=v, r=i;
    }
    if (p > 1e90) return -1;
    pivot(m, n, a, B, N, r, c);
  }
}
```

6 Others

6.1 DLX

```
namespace dlx_with_overlapping{
  //dlx可重覆盖模板，行和列的下标从0开始，一开始调用init(R,C)初始化，R为行数，C为列数
  //在x行y列添加节点直接调用add(x,y)
  //如果行带权的话存val数组，下表同样从0开始，慢的话看情况加最优化剪枝
  const int N = 55;
  const int R = N;
  const int C = 55;
  const int MaxNode = R * C;

  struct node {
      int x, y;
      node *l, *r, *u, *d;
  };

  node nodes[MaxNode], *nxt, *root, *row[R], *col[C];
  int size[C];
  bool mark[C];

  void init(int r, int c) {
      nxt = nodes;
      memset(row, 0, sizeof(row));
      memset(size, 0, sizeof(size));
      root = nxt++;
      root->l = root->r = root;
      for (int y = 0; y < c; ++y) {
          node *p = nxt++;
          p->x = -1; p->y = y;
          p->r = root;
          p->l = root->l;
          p->r->l = p->l->r = p;
          col[y] = p->u = p->d = p;
      }
  }

  node *add(int x, int y) {
      node *p = nxt++;
      p->x = x; p->y = y;
      size[y]++;
      if (!row[x]) {
          row[x] = p->l = p->r = p;
      } else {
          p->r = row[x];
          p->l = row[x]->l;
          p->r->l = p->l->r = p;
      }
      p->d = col[y];
      p->u = col[y]->u;
      p->u->d = p->d->u = p;
      return p;
  }

  void cover(node *x) {
      for (node *y = x->d; y != x; y = y->d) {
          y->l->r = y->r;
          y->r->l = y->l;
          size[x->y]--;
      }
  }

  void uncover(node *x) {
      for (node *y = x->u; y != x; y = y->u) {
          y->l->r = y->r->l = y;
          size[x->y]++;
      }
  }

  int h() {
      int res = 0;
      node *x, *y, *z;
      memset(mark, 0, sizeof(mark));
      for (x = root->l; x != root; x = x->l) if (!mark[x->y]) {
          mark[x->y] = 1;
          res++;
          for (y = x->u; y != x; y = y->u)
              for (z = y->r; z != y; z = z->r)
                  mark[z->y] = 1;
      }
      return res;
  } //贪心确定上界

  int ans;
  int val[R];
  void dfs(int dep, int curval)  {
    if (curval >= ans) return ;
    node *x, *y, *z = NULL;
    if (dep < h()) return ;
    for (x = root->r; x != root; x = x->r) {
      if (!z || size[x->y] < size[z->y]) z = x;
    }
    if (!z) {
      ans = min(ans, curval);
      return ;
    }
    if (!dep) return ;
    for (x = z->u; x != z; x = x->u) {
      cover(x);
      for (y = x->r; y != x; y = y->r) cover(y);
      dfs(dep - 1, curval + val[x->x]);
      for (y = x->l; y != x; y = y->l) uncover(y);
      uncover(x);
    }
  }

  int solve(int MAX, int *_val)  {
    ans = INF;
    memcpy(val, _val, sizeof(val));
    dfs(MAX, 0);
    return ans;
  }
};
```

6.2 Java References

```
// DecimalFormat if you want to output double with specific precision
// FileInputStream if file is needed. FileReader, FileWriter?
// new PrintWrite(new BufferedWriter(  ...

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

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

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

6.3 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);
}
```

6.4 nCk Iterator

升序枚举 C(n,k) 的所有组合。

```
for(int comb = (1 << k) - 1;comb < (1 << n);) {
  // do something here...

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

6.5 信仰

6.5.1 信仰

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

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

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

7 外挂

7.1 Evaluate

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

7.2 mulmod

```
/* return x*y%mod. no overflow if x,y < mod
 * remove 'i' in "idiv"/"imul" -> unsigned */
inline long mulmod(long x,long y,long mod)
{
  long ans = 0;
  __asm__ (
    "movq %1,%%rax\n imulq %2\n idivq %3\n"
    :"=d"(ans):"m"(x),"m"(y),"m"(mod):"%rax"
  );
  return ans;
}
```

7.3 pb_ds

use a pair to emulate multiset, rank value is 0-based.

pairing_heap::join $O(1)$, thin_heap::join $O(n)$, join with tree_order_statistics $= O(n)$, take care.

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

using namespace __gnu_pbds;

// .order_of_key() = find rank for a given value, .
find_by_order() = find value for a given rank
template<typename T> using rset=tree<T, null_type, less<T
>, rb_tree_tag, tree_order_statistics_node_update>;
template<typename T, typename U> rmap=tree<T, U, less<T>,
rb_tree_tag, tree_order_statistics_node_update>;

// 合并，需要保证st中所有元素的key比another_st中的小。
拆分是严格大于key的。
st.join(another_st); st.split(key, output_st);

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

7.4 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;
}
```

8 Lookup Tables

8.1 Integration Table

8.1.1 含有 $ax+b$ 的积分 $(a \neq 0)$

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

8.1.2 含有 $\sqrt{ax+b}$ 的积分

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

8.1.3 含有 $x^2 \pm a^2$ 的积分

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

8.1.4 含有 $ax^2+b(a>0)$ 的积分

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

8.1.5 含有 $ax^2+bx+c(a>0)$ 的积分

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

8.1.6 含有 $\sqrt{x^2+a^2}(a>0)$ 的积分

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

8.1.7 含有 $\sqrt{x^2-a^2}(a>0)$ 的积分

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

8.1.8 含有 $\sqrt{a^2-x^2}(a>0)$ 的积分

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

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

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

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

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

8.1.11 含有三角函数的积分

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

8.1.12 含有反三角函数的积分 (其中 $a>0$)

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

8.1.13 含有指数函数的积分

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

8.1.14 含有对数函数的积分

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

8.2 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					