KUET_Anarosh Team Notebook

Kazi Rifat Al Muin, Md. Arifur Rahman, Hassan Mohammad Naquibul Hoque

Contents

1 Data Structure

1.1 Dsu With Rollback [89 lines] - 4519b900

```
struct dsu_save {
  int v, rnkv, u, rnku;
  dsu_save() {}
  dsu_save(int _v, int _rnkv, int _u, int _rnku)
    : v(_v), rnkv(_rnkv), u(_u), rnku(_rnku) { }
};
struct dsu_with_rollbacks {
  vector<int> p, rnk;
  int comps;
  stack<dsu_save> op;
  dsu_with_rollbacks() {}
  dsu_with_rollbacks(int n) {
    p.resize(n);
    rnk.resize(n);
    for (int i = 0; i < n; i++) {
      p[i] = i;
      rnk[i] = 0;
    }
    comps = n;
  }
  int find_set(int v) { return (v == p[v]) ? v :
      find_set(p[v]); }
  bool unite(int v, int u) {
    v = find_set(v);
    u = find_set(u);
    if (v == u) return false;
    comps--;
    if (rnk[v] > rnk[u]) swap(v, u);
    op.push(dsu_save(v, rnk[v], u, rnk[u]));
    p[v] = u;
    if (rnk[u] == rnk[v]) rnk[u]++;
    return true;
  }
  void rollback() {
    if (op.empty()) return;
    dsu_save x = op.top();
    op.pop();
    comps++;
    p[x.v] = x.v;
    rnk[x.v] = x.rnkv;
    p[x.u] = x.u;
    rnk[x.u] = x.rnku;
  }
};
struct query {
  int v, u;
  bool united;
  query(int _v, int _u) : v(_v), u(_u) {}
};
struct QueryTree {
  vector<vector<query>> t;
  dsu_with_rollbacks dsu;
  int T;
  QueryTree() {}
  QueryTree(int _T, int n) : T(_T) {
    dsu = dsu_with_rollbacks(n);
    t.resize(4 * T + 4);
  }
  void add_to_tree(int v, int l, int r, int ul, int ur,
      query& q) {
    if (ul > ur) return;
    if (l == ul && r == ur) {
      t[v].push_back(q);
      return;
    }
    int mid = (l + r) / 2;
    add_to_tree(2 * v, l, mid, ul, min(ur, mid), q);
    add_to_tree(2 * v + 1, mid + 1, r, max(ul, mid +
        1), ur, q);
  }
  void add_query(query q, int l, int r) {
      add_to_tree(1, 0, T - 1, l, r, q); }
  void dfs(int v, int l, int r, vector<int>& ans) {
    for (query& q : t[v]) {
      q.united = dsu.unite(q.v, q.u);
    }
    if (l == r)
      ans[l] = dsu.comps;
    else {
      int mid = (l + r) / 2;
      dfs(2 * v, l, mid, ans);
      dfs(2 * v + 1, mid + 1, r, ans);
    }
    for (query q : t[v]) {
      if (q.united) dsu.rollback();
    }
  }
  vector<int> solve() {
    vector<int> ans(T);
    dfs(1, 0, T - 1, ans);
    return ans;
  }
};
```

1.2 MO with Update [43 lines] - a0826346

```
//1 indexed
//Complexity:$O(S \times Q + Q \times \frac{N^2}{S^2})$
//S = (2*n^2)^(1/3)
const int block_size = 2720;  // 4310 for 2e5
const int mx = 1e5 + 5;
struct Query {
  int L, R, T, id;
  Query() {}
  Query(int _L, int _R, int _T, int _id) : L(_L),
      R(_R), T(_T), id(_id) {}
  bool operator<(const Query &x) const {
    if (L / block_size == x.L / block_size) {
      if (R / block_size == x.R / block_size) return T <
          x.T;
      return R / block_size < x.R / block_size;
    }
    return L / block_size < x.L / block_size;
  }
} Q[mx];
struct Update {
  int pos;
  int old, cur;
  Update(){};
  Update(int _p, int _o, int _c) : pos(_p), old(_o),
      cur(_c){};
} U[mx];
int ans[mx];
inline void add(int id) {}
inline void remove(int id) {}
inline void update(int id, int L, int R) {}
inline void undo(int id, int L, int R) {}
inline int get() {}
void MO(int nq, int nu) {
  sort(Q + 1, Q + nq + 1);
  int L = 1, R = 0, T = nu;
  for (int i = 1; i <= nq; i++) {
    Query q = Q[i];
    while (T < q.T) update(++T, L, R);
    while (T > q.T) undo(T--, L, R);
    while (L > q.L) add(--L);
    while (R < q.R) add(++R);
    while (L < q.L) remove(L++);
    while (R > q.R) remove(R--);
    ans[q.id] = get();
  }
}
```

1.3 MO [28 lines] - ec9fc177

```
const int N = 2e5 + 5;
const int Q = 2e5 + 5;
const int SZ = sqrt(N) + 1;
struct qry {
  int l, r, id, blk;
  bool operator<(const qry& p) const {
    return blk == p.blk ? r < p.r : blk < p.blk;
  }
};
qry query[Q];
ll ans[Q];
void add(int id) {}
void remove(int id) {}
ll get() {}
int n, q;
void MO() {
  sort(query, query + q);
  int cur_l = 0, cur_r = -1;
  for (int i = 0; i < q; i++) {
    qry q = query[i];
    while (cur_l > q.l) add(--cur_l);
    while (cur_r < q.r) add(++cur_r);
    while (cur_l < q.l) remove(cur_l++);
    while (cur_r > q.r) remove(cur_r--);
    ans[q.id] = get();
  }
}
/* 0 indexed. */
```

1.4 Persistent Segment Tree [64 lines] - b61e98f7

```
const int mxn = 4e5+5;
int root[mxn], leftchild[25*mxn], rightchild[25*mxn],
    value[25*mxn], a[mxn];
int now = 0, n, sz = 1;
int l, r;

int build(int L, int R){
  int node = ++now;
  if(L == R){
    //initialize
    //value[node] = a[L];
    return node;
  }
  int mid = (L+R)>>1;
  leftchild[node] = build(L, mid);
  rightchild[node] = build(mid+1, R);
  //combine
  //value[node] = value[leftchild[node]] +
      value[rightchild[node]];
  return node;
}

int update(int nownode, int L, int R, int ind, int val){
  int node = ++now;
  if(L == R){
    //value[node] = value[nownode]+val;
    //update value[node]
    return node;
  }
  int mid = (L+R)>>1;
  leftchild[node] = leftchild[nownode];
  rightchild[node] = rightchild[nownode];
  if(mid >= ind){//change condition as required
    leftchild[node] = update(leftchild[nownode], L,
        mid, ind, val);
  }
  else{
    rightchild[node] = update(rightchild[nownode],
        mid+1, R, ind, val);
  }
  //value[node] = value[leftchild[node]] +
      value[rightchild[node]];
  //combine value[node]
  return node;
}


int query(int nownode, int L, int R){
  if(l > R || r < L) return 0;

  if(L>=l && r >= R){
    return value[nownode];
  }
  int mid = (L+R)>>1;
  //change as required
  return query(leftchild[nownode], L, mid) +
      query(rightchild[nownode], mid+1, R);
}

void persistant(){
  root[0] = build(1, n);
  while(m--){
    if(ck == 2){
      cout << query(root[idx], 1, n) << "\n";
    }
    else{
        root[sz++] = update(root[idx], 1, n, ind,
            val);
    }
  }
}
```

1.5 SQRT Decomposition [96 lines] - 80a3d1e6

```
struct sqrtDecomposition {
  static const int sz = 320;//sz = sqrt(N);
  int numberofblocks;

  struct node {
    int L, R;
    bool islazy = false;
    ll lazyval=0;
    //extra data needed for different problems
    void ini(int l, int r) {
      for(int i=l; i<=r; i++) {
        //...initialize as need
      }
      L=l, R=r;
    }
    void semiupdate(int l, int r, ll val) {
```

```
    if(l>r) return;
    if(islazy){
      for(int i=L; i<=R; i++){
        //...distribute lazy to everyone
      }
      islazy = 0;
      lazyval = 0;
    }
    for(int i=l; i<=r; i++){
      //...do it manually
    }
  }
  void fullupdate(ll val){
    if(islazy){
      //...only update lazyval
    }
    else{
      for(int i=L; i<=R; i++){
        //...everyone are not equal, make them equal
      }
      islazy = 1;
      //update lazyval
    }
  }
  void update(int l, int r, ll val){
    if(l<=L && r>=R) fullupdate(val);
    else semiupdate(max(l, L), min(r, R), val);
  }
  ll semiquery(int l, int r){
    if(l>r) return 0;
    if(islazy){
      for(int i=L; i<=R; i++){
        //...distribute lazy to everyone
      }
      islazy = 0;
      lazyval = 0;
    }
    ll ret = 0;
    for(int i=l; i<=r; i++){
      //...take one by one
    }
    return ret;
  }
  ll fullquery(){
    //return stored value;
  }
  ll query(int l, int r){
    if(l<=L && r>=R) return fullquery();
    else return semiquery(max(l, L), min(r, R));
  }
};
vector<node> blocks;
void init(int n){
  numberofblocks = (n+sz-1)/sz;
  int curL = 1, curR = sz;
  blocks.resize(numberofblocks+5);
  for(int i=1; i<=numberofblocks; i++){
    curR = min(n, curR);
    blocks[i].ini(curL, curR);
    curL += sz;
    curR += sz;
  }
}
void update(int l, int r, ll val){
    int left = (l-1)/sz+1;
    int right = (r-1)/sz+1;
    for(int i=left; i<=right; i++){
      blocks[i].update(l, r, val);
    }
  }
  ll query(int l, int r){
    int left = (l-1)/sz+1;
    int right = (r-1)/sz+1;
    ll ret = 0;
    for(int i=left; i<=right; i++){
      ret += blocks[i].query(l, r);
    }
    return ret;
  }
};
```

1.6 Segment Tree [73 lines] - 1b64fde6

```
/*edit:data,combine,build check datatype*/
template<typename T>
struct SegmentTree {
#define lc (C << 1)
#define rc (C << 1 | 1)
#define M ((L+R)>>1)
  struct data {
    T sum;
    data() :sum(0) {};
  };
  vector<data>st;
  vector<bool>isLazy;
  vector<T>lazy;
  int N;
  SegmentTree(int _N) :N(_N) {
    st.resize(4 * N);
    isLazy.resize(4 * N);
    lazy.resize(4 * N);
  }
  void combine(data& cur, data& l, data& r) {
    cur.sum = l.sum + r.sum;
  }
  void push(int C, int L, int R) {
    if (!isLazy[C]) return;
    if (L != R) {
      isLazy[lc] = 1;
      isLazy[rc] = 1;
      lazy[lc] += lazy[C];
      lazy[rc] += lazy[C];
    }
    st[C].sum = (R - L + 1) * lazy[C];
    lazy[C] = 0;
    isLazy[C] = false;
  }
  void build(int C, int L, int R) {
    if (L == R) {
      st[C].sum = 0;
      return;
    }
    build(lc, L, M);
    build(rc, M + 1, R);
    combine(st[C], st[lc], st[rc]);
  }
  data Query(int i, int j, int C, int L, int R) {
    push(C, L, R);
    if (j < L || i > R || L > R) return data();  //
        default val 0/INF
    if (i <= L && R <= j) return st[C];
    data ret;
    data d1 = Query(i, j, lc, L, M);
    data d2 = Query(i, j, rc, M + 1, R);
    combine(ret, d1, d2);
    return ret;
  }
  void Update(int i, int j, T val, int C, int L, int R)
      {
    push(C, L, R);
    if (j < L || i > R || L > R) return;
    if (i <= L && R <= j) {
      isLazy[C] = 1;
      lazy[C] = val;
      push(C, L, R);
      return;
    }
    Update(i, j, val, lc, L, M);
    Update(i, j, val, rc, M + 1, R);
    combine(st[C], st[lc], st[rc]);
  }
  void Update(int i, int j, T val) {
    Update(i, j, val, 1, 1, N);
  }
  T Query(int i, int j) {
    return Query(i, j, 1, 1, N).sum;
  }
};
```

1.7 Sqrt Tricks [8 lines] - 6b5387c8

```
1. Size of the block is not always Sqrt, adjust it as
    necessary. if o(n/b+b) then take n/b = b and
    calculate b.
2. MO's Algorithm
   *it is possible to solve a Mo problem without any
       remove operation. For L in one block R only
       increases, for every range we can start L from
       the last of that block
3. Sqrt Decomposition by time of queries.
   *keep overall solution and sqrt(n) updates in a
       vector and for a query iterate over all of them,
       when the vector size exceeds sqrt(n) you can add
       these updates with overall solution using o(n)
4. If sum of N positive numbers are S, there are at most
    sqrt(S) distinct values.
5. Randomization
6. Baby step, gaint step
```

1.8 Treap [166 lines] - b63d91d7

```
struct Treap {
  struct Node {
    int val, priority, cnt;  // value, priority, subtree
        size
    Node* l, * r;             // left child,right child
        pointer
    Node() {} //rng from template
    Node(int key) : val(key), priority(rng()),
        l(nullptr), r(nullptr) {}
  };
  typedef Node* node;
  node root;
  Treap() : root(0) {}
  int cnt(node t) { return t ? t->cnt : 0; }  // return
      subtree size
void updateCnt(node t) {
  if (t) t->cnt = 1 + cnt(t->l) + cnt(t->r);  //
      update subtree size
}
void push(node cur) {
  ;  // Lazy Propagation
}

void combine(node& cur, node l, node r) {
  if (!l) {
    cur = r;
    return;
  }
  if (!r) {
    cur = l;
    return;
  }
  // Merge Operations like in segment tree
}

void reset(node& cur) {
  if (!cur) return;  // To reset other fields of cur
      except value and cnt
}

void operation(node& cur) {
  if (!cur) return;
  reset(cur);
  combine(cur, cur->l, cur);
  combine(cur, cur, cur->r);
}
// Split(T,key): split the tree in two tree. Left
    pointer contains all value
// less than or equal to key.Right pointer contains
    the rest.
void split(node t, node& l, node& r, int key) {
  if (!t)
    return void(l = r = nullptr);
  push(t);
  if (t->val <= key) {
    split(t->r, t->r, r, key), l = t;

  }
  else {
    split(t->l, l, t->l, key), r = t;
  }
  updateCnt(t);
  operation(t);
}
void splitPos(node t, node& l, node& r, int k, int add
    = 0) {
  if (!t) return void(l = r = 0);
  push(t);
  int idx = add + cnt(t->l);
  if (idx <= k)
    splitPos(t->r, t->r, r, k, idx + 1), l = t;
  else
    splitPos(t->l, l, t->l, k, add), r = t;
  updateCnt(t);
  operation(t);
}
// Merge(T1,T2): merges 2 tree into one.The tree with
    root of higher
// priority becomes the new root.
void merge(node& t, node l, node r) {
  push(l);
  push(r);
  if (!l || !r)
    t = l ? l : r;
  else if (l->priority > r->priority)
    merge(l->r, l->r, r), t = l;
  else
    merge(r->l, l, r->l), t = r;
  updateCnt(t);
  operation(t);
}

node merge_treap(node l, node r) {
  if (!l) return r;
  if (!r) return l;
  if (l->priority < r->priority) swap(l, r);
  node L, R;
  split(r, L, R, l->val);
  l->r = merge_treap(l->r, R);
  l->l = merge_treap(L, l->l);
  updateCnt(l);
  operation(l);
  return l;
}
// insert creates a set.all unique value.
void insert(int val) {
  if (!root) {
    root = new Node(val);
    return;
  }
  node l, r, mid, mid2, rr;
  mid = new Node(val);
  split(root, l, r, val);
  merge(l, l, mid);  // these 3 lines will create
      multiset.
  merge(root, l, r);
  /*split(root, l, r, val - 1);  // l contains all
      small values.
    merge(l, l, mid);            // l contains new val
        too.
    split(r, mid2, rr, val);     // rr contains all
        greater values.
    merge(root, l, rr);*/
}
// removes all similar values.
void erase(int val) {
  node l, r, mid;
  /* Removes all similar element*/
  split(root, l, r, val - 1);
  split(r, mid, r, val);
  merge(root, l, r);
  /*Removes single instance*/
  /*split(root, l, r, val - 1);
    split(r, mid, r, val);
    merge(mid,mid->l,mid->r);
    merge(l, l, mid);
    merge(root, l, r);*/
}
void clear(node cur) {
  if (!cur) return;
  clear(cur->l), clear(cur->r);
  delete cur;
}
  void clear() { clear(root); }
  void inorder(node t) {
    if (!t) return;
    inorder(t->l);
    cout << t->val << ' ';
    inorder(t->r);
  }
  void inorder() {
    inorder(root);
    puts("");
  }
  //1 indexed - xth element after sorting.
  int find_by_order(int x) {
    if (!x) return -1;
    x--;
    node l, r, mid;
    splitPos(root, l, r, x - 1);
    splitPos(r, mid, r, 0);
    int ans = -1;
    if (cnt(mid) == 1) ans = mid->val;
    merge(r, mid, r);
    merge(root, l, r);
  }
  // 1 indexed. index of val in sorted array. -1 if not
      found.
  int order_of_key(int val) {
    node l, r, mid;
    split(root, l, r, val - 1);
    split(r, mid, r, val);
    int ans = -1;
    if (cnt(mid) == 1) ans = 1 + cnt(l);
    merge(r, mid, r);
    merge(root, l, r);
    return ans;
  }
};
```

1.9 Trie Bit [61 lines] - 25d39ae1

```
struct Trie {
  struct node {
    int next[2];
    int cnt, fin;
    node() :cnt(0), fin(0) {
      for (int i = 0; i < 2; i++) next[i] = -1;
    }
  };
  vector<node>data;
  Trie() {
    data.push_back(node());
  }
  void key_add(int val) {
    int cur = 0;
    for (int i = 30; i >= 0; i--) {
      int id = (val >> i) & 1;
      if (data[cur].next[id] == -1) {
        data[cur].next[id] = data.size();
        data.push_back(node());
      }
      cur = data[cur].next[id];
      data[cur].cnt++;
    }
    data[cur].fin++;
  }
  int key_search(int val) {
    int cur = 0;
    for (int i = 30; ~i; i--) {
      int id = (val >> i) & 1;
      if (data[cur].next[id] == -1) return 0;
      cur = data[cur].next[id];
    }
    return data[cur].fin;
  }
  void key_delete(int val) {
    int cur = 0;
    for (int i = 30; ~i; i--) {
      int id = (val >> i) & 1;
      cur = data[cur].next[id];
      data[cur].cnt--;
    }
    data[cur].fin--;
  }
  bool key_remove(int val) {
    if (key_search(val)) return key_delete(val), 1;
    return 0;
  }
  int maxXor(int x) {
    int cur = 0;
    int ans = 0;
    for (int i = 30; ~i; i--) {
      int b = (x >> i) & 1;
      if (data[cur].next[!b] + 1 &&
          data[data[cur].next[!b]].cnt > 0) {
        ans += (1LL << i);
        cur = data[cur].next[!b];
      }
      else cur = data[cur].next[b];
    }
    return ans;
  }
};
```

2 Dynamic Programming

2.1 Divide and Conquer DP [26 lines] - 6000baee

```
ll G,L;///total group,cell size
ll dp[8001][801],cum[8001];
ll C[8001];///value of each cell
inline ll cost(ll l,ll r){
  return(cum[r]-cum[l-1])*(r-l+1);
}
void fn(ll g,ll st,ll ed,ll r1,ll r2){
  if(st>ed)return;
  ll mid=(st+ed)/2,pos=-1;
  dp[mid][g]=inf;
  for(int i=r1;i<=r2;i++){
    ll tcost=cost(i,mid)+dp[i-1][g-1];
    if(tcost<dp[mid][g]){
      dp[mid][g]=tcost,pos=i;
    }
  }
  fn(g,st,mid-1,r1,pos);
  fn(g,mid+1,ed,pos,r2);
}
int main(){
  for(int i=1;i<=L;i++)
    cum[i]=cum[i-1]+C[i];
  for(int i=1;i<=L;i++)
    dp[i][1]=cost(1,i);
  for(int i=2;i<=G;i++)fn(i,1,L,1,L);
}
```

2.2 Dynamic Convex Hull Trick [66 lines] - 86f3d1cf

```
const int N = 3e5 + 9;
const int mod = 1e9 + 7;

//add lines with -m and -b and return -ans to
//make this code work for minimums.(not -x)
const ll inf = -(1LL << 62);
struct line {
  ll m, b;
  mutable function<const line*() > succ;
  bool operator < (const line& rhs) const {
    if (rhs.b != inf) return m < rhs.m;
    const line* s = succ();
    if (!s) return 0;
    ll x = rhs.m;
    return b - s->b < (s->m - m) * x;
  }
};
struct CHT : public multiset<line> {
  bool bad(iterator y) {
    auto z = next(y);
    if (y == begin()) {
      if (z == end()) return 0;
      return y -> m == z -> m && y -> b <= z -> b;
    }
    auto x = prev(y);
    if (z == end()) return y -> m == x -> m && y -> b
        <= x -> b;
    return 1.0 * (x -> b - y -> b) * (z -> m - y -> m)
        >= 1.0 * (y -> b - z -> b) * (y -> m - x -> m);
  }
  void add(ll m, ll b) {
    auto y = insert({ m, b });
    y->succ = [ = ] { return next(y) == end() ? 0 :
        &*next(y); };
    if (bad(y)) {
      erase(y);
      return;
    }
    while (next(y) != end() && bad(next(y)))
        erase(next(y));
    while (y != begin() && bad(prev(y))) erase(prev(y));
  }
  ll query(ll x) {
    assert(!empty());
    auto l = *lower_bound((line) {
      x, inf
    });
    return l.m * x + l.b;
  }
};
CHT* cht;
ll a[N], b[N];
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);

  int n;
  cin >> n;
  for(int i = 0; i < n; i++) cin >> a[i];
  for(int i = 0; i < n; i++) cin >> b[i];
  cht = new CHT();
  cht -> add(-b[0], 0);
  ll ans = 0;
  for(int i = 1; i < n; i++) {
    ans = -cht -> query(a[i]);
    cht -> add(-b[i], -ans);
  }
  cout << ans << nl;
  return 0;
}
```

2.3 Knuth Optimization [32 lines] - 5f2f74dc

```
/*It is applicable where recurrence is in the form :
dp[i][j]=mini<k<j{dp[i][k]+dp[k][j]}+C[i][j]
condition for applicability is:
A[i,j-1]<=A[i,j]<=A[i+1,j]
Where,
A[i][j]-the smallest k that gives optimal answer,like-
dp[i][j]=dp[i-1][k]+C[k][j]
C[i][j]-given cost function
also applicable if: C[i][j]satisfies the following 2
    conditions:
C[a][c]+C[b][d]<=C[a][d]+C[b][c],a<=b<=c<=d
C[b][c]<=C[a][d],a<=b<=c<=d
reduces time complexity from O(n^3)to O(n^2)*/
for(int s=0;s<=k;s++)//s-length(size)of substring
  for(int l=0;l+s<=k;l++){//l-left point
    int r=l+s;//r-right point
    if(s<2){
      res[l][r]=0;//DP base-nothing to break
      mid[l][r]=l;/*mid is equal to left border*/
      continue;
    }
    int mleft=mid[l][r-1];/*Knuth's trick: getting
        bounds on m*/
    int mright=mid[l+1][r];
    res[l][r]=inf;
    for(int m=mleft;m<=mright;m++){/*iterating for m in
        the bounds only*/
      int64 tres=res[l][m]+res[m][r]+(x[r]-x[l]);
      if(res[l][r]>tres){//relax current solution
        res[l][r]=tres;
        mid[l][r]=m;
      }
    }
  }
int64 answer=res[0][k];
```

2.4 LIS O(nlogn) with full path [17 lines] - bcb566b7

```
int num[MX],mem[MX],prev[MX],array[MX],res[MX],maxlen;
void LIS(int SZ,int num[]){
  CLR(mem),CLR(prev),CLR(array),CLR(res);
  int i,k;
  maxlen=1;
  array[0]=-inf;
  RFOR(i,1,SZ+1)  array[i]=inf;
  prev[0]=-1,mem[0]=num[0];
  FOR(i,SZ){
    k=lower_bound(array,array+maxlen+1,num[i])-array;
    if(k==1)  array[k]=num[i],mem[k]=i,prev[i]=-1;
    else array[k]=num[i],mem[k]=i,prev[i]=mem[k-1];
    if(k>maxlen)  maxlen=k;
  }
  k=0;
  for(i=mem[maxlen];i!=-1;i=prev[i])res[k++]=num[i];
}
```

2.5 SOS DP [18 lines] - e5398562

```
//iterative version
for(int mask = 0; mask < (1<<N); ++mask){
  dp[mask][-1] = A[mask];  //handle base case separately
     (leaf states)
  for(int i = 0;i < N; ++i){
    if(mask & (1<<i))
      dp[mask][i] = dp[mask][i-1] +
          dp[mask^(1<<i)][i-1];
    else
      dp[mask][i] = dp[mask][i-1];
  }
  F[mask] = dp[mask][N-1];
}
//memory optimized, super easy to code.
for(int i = 0; i<(1<<N); ++i)
  F[i] = A[i];
for(int i = 0;i < N; ++i) for(int mask = 0; mask <
    (1<<N); ++mask){
  if(mask & (1<<i))
    F[mask] += F[mask^(1<<i)];
}
```

2.6 Sibling DP [26 lines] - 95945016

```
/*/dividing tree into min group such that each group
    cost not exceed k*/
ll n,k,dp[mx][mx];
vector<pair<ll,ll>>adj[mx];///must be rooted tree
ll sibling_dp(ll par,ll idx,ll remk){
  if(remk<0)return inf;
  if(adj[par].size()<idx+1)return 0;
  ll u=adj[par][idx].first;
  if(dp[u][remk]!=-1)
    return dp[u][remk];
  ll ret=inf,under=0,sibling=0;
  if(par!=0){//creating new group
    under=1+dfs(u,0,k);
    sibling=dfs(par,idx+1,remk);
    ret=min(ret,under+sibling);
  }
  //divide the current group
  ll temp=remk-adj[par][idx].second;
  for(ll chk=temp;chk>=0;chk--){
    ll siblingk=temp-chk;
    under=0,sibling=0;
    under=dfs(u,0,chk);
    sibling=dfs(par,idx+1,siblingk);
    ret=min(ret,under+sibling);
  }
  return dp[u][remk]=ret;
}
```

3 Flow

3.1 Blossom [58 lines] - 411402f2

```
// Finds Maximum matching in General Graph
// Complexity O(NM)
// mate[i] = j means i is paired with j
// source: https://codeforces.com/blog/entry␣
    /92339?#comment-810242
vector<int> Blossom(vector<vector<int>>& graph) {
  //mate contains matched edge.
  int n = graph.size(), timer = -1;
  vector<int> mate(n, -1), label(n), parent(n),
    orig(n), aux(n, -1), q;
  auto lca = [&](int x, int y) {
    for (timer++; ; swap(x, y)) {
      if (x == -1) continue;
      if (aux[x] == timer) return x;
      aux[x] = timer;
      x = (mate[x] == -1 ? -1 : orig[parent[mate[x]]]);
    }
  };
  auto blossom = [&](int v, int w, int a) {
    while (orig[v] != a) {
      parent[v] = w; w = mate[v];
      if (label[w] == 1) label[w] = 0, q.push_back(w);
      orig[v] = orig[w] = a; v = parent[w];
    }
  };
  auto augment = [&](int v) {
    while (v != -1) {
      int pv = parent[v], nv = mate[pv];
      mate[v] = pv; mate[pv] = v; v = nv;
    }
  };
  auto bfs = [&](int root) {
    fill(label.begin(), label.end(), -1);
    iota(orig.begin(), orig.end(), 0);
    q.clear();
    label[root] = 0; q.push_back(root);
    for (int i = 0; i < (int)q.size(); ++i) {
      int v = q[i];
      for (auto x : graph[v]) {
        if (label[x] == -1) {
          label[x] = 1; parent[x] = v;
          if (mate[x] == -1)
            return augment(x), 1;
          label[mate[x]] = 0; q.push_back(mate[x]);
        }
        else if (label[x] == 0 && orig[v] != orig[x]) {
          int a = lca(orig[v], orig[x]);
          blossom(x, v, a); blossom(v, x, a);
        }
      }
    }
    return 0;
  };
  // Time halves if you start with (any) maximal
      matching.
  for (int i = 0; i < n; i++)
    if (mate[i] == -1)
      bfs(i);
  return mate;
}
```

3.2 Dinic [72 lines] - 1f1b6e44

```
/*.Complexity: O(V^2 E)
  .Call Dinic with total number of nodes.
  .Nodes start from 0.
  .Capacity is long long data.
  .make graph with create edge(u,v,capacity).
  .Get max flow with maxFlow(src,des).*/
#define eb emplace_back
struct Dinic {
  struct Edge {
    int u, v;
    ll cap, flow = 0;
    Edge() {}
    Edge(int u, int v, ll cap) :u(u), v(v), cap(cap) {}
  };
  int N;
  vector<Edge>edge;
  vector<vector<int>>adj;
  vector<int>d, pt;
  Dinic(int N) :N(N), edge(0), adj(N), d(N), pt(N) {}
  void addEdge(int u, int v, ll cap) {
    if (u == v) return;
    edge.eb(u, v, cap);
    adj[u].eb(edge.size() - 1);
    edge.eb(v, u, 0);
    adj[v].eb(edge.size() - 1);
  }
  bool bfs(int s, int t) {
    queue<int>q({ s });
    fill(d.begin(), d.end(), N + 1);
    d[s] = 0;
    while (!q.empty()) {
      int u = q.front();q.pop();
      if (u == t) break;
      for (int k : adj[u]) {
        Edge& e = edge[k];
        if (e.flow<e.cap && d[e.v]>d[e.u] + 1) {
          d[e.v] = d[e.u] + 1;
          q.emplace(e.v);
        }
      }
    }
    return d[t] != N + 1;
  }
  ll dfs(int u, int T, ll flow = -1) {
    if (u == T || flow == 0) return flow;
    for (int& i = pt[u];i < adj[u].size();i++) {
      Edge& e = edge[adj[u][i]];
      Edge& oe = edge[adj[u][i] ^ 1];
      if (d[e.v] == d[e.u] + 1) {
        ll amt = e.cap - e.flow;
        if (flow != -1 && amt > flow) amt = flow;
        if (ll pushed = dfs(e.v, T, amt)) {
          e.flow += pushed;
          oe.flow -= pushed;
          return pushed;
        }
      }
    }
    return 0;
  }
  ll maxFlow(int s, int t) {
    ll total = 0;
    while (bfs(s, t)) {
      fill(pt.begin(), pt.end(), 0);
      while (ll flow = dfs(s, t)) {
        total += flow;
      }
    }
    return total;
  }

};
```

3.3 Flow [6 lines] - fa60fe03

```
Covering Problems:
> Maximum Independent Set(Bipartite): Largest set of
    nodes which do not have any edge between them. sol:
    V-(MaxMatching)
> Minimum Vertex Cover(Bipartite): -Smallest set of
    nodes to cover all the edges -sol: MaxMatching
> Minimum Edge Cover(General graph): -Smallest set of
    edges to cover all the nodes -sol: V-(MaxMatching)
    (if edge cover exists, does not exit for isolated
    nodes)
> Minimum Path Cover(Vertex disjoint) DAG: -Minimum
    number of vertex disjoint paths that visit all the
    nodes -sol: make a bipartite graph using same nodes
    in two sides, one side is "from" other is "to", add
    edges from "from" to "to", then ans is
    V-(MaxMatching)
> Minimum Path Cover(Vertex Not Disjoint) General graph:
    -Minimum number of paths that visit all the nodes
    -sol: consider cycles as nodes then it will become a
    path cover problem with vertex disjoint on DAG
```

3.4 HopCroftKarp [67 lines] - 4347b72b

```
/*.Finds Maximum Matching In a bipartite graph
  .Complexity O(E√V)
  .1-indexed
  .No default constructor
  .add single edge for (u, v)*/
struct HK {
  static const int inf = 1e9;
  int n;
  vector<int>matchL, matchR, dist;
  //matchL contains value of matched node for L part.
  vector<vector<int>>adj;
  HK(int n) :n(n), matchL(n + 1),
  matchR(n + 1), dist(n + 1), adj(n + 1) {
  }

  void addEdge(int u, int v) {
    adj[u].push_back(v);
  }
  bool bfs() {
    queue<int>q;
    for (int u = 1;u <= n;u++) {
      if (!matchL[u]) {
        dist[u] = 0;
        q.push(u);
      }
      else dist[u] = inf;
    }
    dist[0] = inf;///unmatched node matches with 0.
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      for (auto v : adj[u]) {
        if (dist[matchR[v]] == inf) {
          dist[matchR[v]] = dist[u] + 1;
          q.push(matchR[v]);
        }
      }
    }
    return dist[0] != inf;
  }
  bool dfs(int u) {
    if (!u) return true;
    for (auto v : adj[u]) {
      if (dist[matchR[v]] == dist[u] + 1
          && dfs(matchR[v])) {
        matchL[u] = v;
        matchR[v] = u;
        return true;
      }
    }
    dist[u] = inf;
    return false;
  }
  int max_match() {
    int matching = 0;
    while (bfs()) {
      for (int u = 1;u <= n;u++) {
        if (!matchL[u])
          if (dfs(u))
            matching++;
      }
    }
    return matching;
  }
};
```

3.5 Hungarian [116 lines] - 99e0df20

```
/* Complexity: O(n^3) but optimized
   It finds minimum cost maximum matching.
   For finding maximum cost maximum matching
   add -cost and return -matching()
   1-indexed */
struct Hungarian {
  long long c[N][N], fx[N], fy[N], d[N];
  int l[N], r[N], arg[N], trace[N];
  queue<int> q;
  int start, finish, n;
  const long long inf = 1e18;
  Hungarian() {}
  Hungarian(int n1, int n2) : n(max(n1, n2)) {
    for (int i = 1; i <= n; ++i) {
      fy[i] = l[i] = r[i] = 0;
      for (int j = 1; j <= n; ++j) c[i][j] = inf;
    }
  }
  void add_edge(int u, int v, long long cost) {
    c[u][v] = min(c[u][v], cost);
  }
  inline long long getC(int u, int v) {
    return c[u][v] - fx[u] - fy[v];
  }
  void initBFS() {
    while (!q.empty()) q.pop();
    q.push(start);
    for (int i = 0; i <= n; ++i) trace[i] = 0;
    for (int v = 1; v <= n; ++v) {
      d[v] = getC(start, v);
      arg[v] = start;
    }
    finish = 0;
  }
  void findAugPath() {
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      for (int v = 1; v <= n; ++v) if (!trace[v]) {
        long long w = getC(u, v);
        if (!w) {
          trace[v] = u;
          if (!r[v]) {
            finish = v;
            return;
          }
          q.push(r[v]);
        }
        if (d[v] > w) {
          d[v] = w;
          arg[v] = u;
        }
      }
    }
  }
  void subX_addY() {
    long long delta = inf;
    for (int v = 1; v <= n; ++v) if (trace[v] == 0 &&
        d[v] < delta) {
      delta = d[v];
    }
    // Rotate
    fx[start] += delta;
    for (int v = 1; v <= n; ++v) if (trace[v]) {
      int u = r[v];
      fy[v] -= delta;
      fx[u] += delta;
    }
    else d[v] -= delta;
    for (int v = 1; v <= n; ++v) if (!trace[v] && !d[v])
      {
      trace[v] = arg[v];
      if (!r[v]) {
        finish = v;
        return;
      }
      q.push(r[v]);
    }
  }
  void Enlarge() {
    do {
      int u = trace[finish];
      int nxt = l[u];
      l[u] = finish;
      r[finish] = u;
      finish = nxt;
    } while (finish);
  }
  long long maximum_matching() {
    for (int u = 1; u <= n; ++u) {
      fx[u] = c[u][1];
      for (int v = 1; v <= n; ++v) {
        fx[u] = min(fx[u], c[u][v]);
      }
    }
    for (int v = 1; v <= n; ++v) {
      fy[v] = c[1][v] - fx[1];
      for (int u = 1; u <= n; ++u) {
        fy[v] = min(fy[v], c[u][v] - fx[u]);
      }
    }
    for (int u = 1; u <= n; ++u) {
      start = u;
      initBFS();
      while (!finish) {
        findAugPath();
        if (!finish) subX_addY();
      }
      Enlarge();
    }
    long long ans = 0;
    for (int i = 1; i <= n; ++i) {
      if (c[i][l[i]] != inf) ans += c[i][l[i]];
      else l[i] = 0;
    }
    return ans;
  }
};
```

3.6 MCMF [116 lines] - 8e83e9c8

```
/*Credit: ShahjalalShohag
  .Works for both directed, undirected and with negative
     cost too
  .doesn't work for negative cycles
  .for undirected edges just make the directed flag
     false
  .Complexity: O(min(E^2 *V log V, E logV * flow))*/
using T = long long;
const T inf = 1LL << 61;
struct MCMF {
  struct edge {
    int u, v;
    T cap, cost;
    int id;
    edge(int _u, int _v, T _cap, T _cost, int _id) {
      u = _u;
      v = _v;
      cap = _cap;
      cost = _cost;
      id = _id;
    }
  };
  int n, s, t, mxid;
  T flow, cost;
  vector<vector<int>> g;
  vector<edge> e;
  vector<T> d, potential, flow_through;
  vector<int> par;
  bool neg;
  MCMF() {}
  MCMF(int _n) { // 0-based indexing
    n = _n + 10;
    g.assign(n, vector<int>());
    neg = false;
    mxid = 0;
  }
  void add_edge(int u, int v, T cap, T cost, int id =
      -1, bool directed = true) {
    if (cost < 0) neg = true;
    g[u].push_back(e.size());
    e.push_back(edge(u, v, cap, cost, id));
    g[v].push_back(e.size());
    e.push_back(edge(v, u, 0, -cost, -1));
    mxid = max(mxid, id);
    if (!directed) add_edge(v, u, cap, cost, -1, true);
  }
  bool dijkstra() {
    par.assign(n, -1);
    d.assign(n, inf);
    priority_queue<pair<T, T>, vector<pair<T, T>>,
        greater<pair<T, T>> > q;
    d[s] = 0;
    q.push(pair<T, T>(0, s));
    while (!q.empty()) {
      int u = q.top().second;
      T nw = q.top().first;
      q.pop();
      if (nw != d[u]) continue;
      for (int i = 0; i < (int)g[u].size(); i++) {
        int id = g[u][i];
        int v = e[id].v;
        T cap = e[id].cap;
        T w = e[id].cost + potential[u] - potential[v];
        if (d[u] + w < d[v] && cap > 0) {
          d[v] = d[u] + w;
          par[v] = id;
          q.push(pair<T, T>(d[v], v));
        }
      }
    }
    for (int i = 0; i < n; i++) { // update potential
      if (d[i] < inf) potential[i] += d[i];
    }
    return d[t] != inf;
  }
  T send_flow(int v, T cur) {
    if (par[v] == -1) return cur;
    int id = par[v];
    int u = e[id].u;
    T w = e[id].cost;
    T f = send_flow(u, min(cur, e[id].cap));
    cost += f * w;
    e[id].cap -= f;
    e[id ^ 1].cap += f;
    return f;
  }
  //returns {maxflow, mincost}
  pair<T, T> solve(int _s, int _t, T goal = inf) {
    s = _s;
    t = _t;
    flow = 0, cost = 0;
    potential.assign(n, 0);
    if (neg) {
      // run Bellman-Ford to find starting potential
      d.assign(n, inf);
      for (int i = 0, relax = true; i < n && relax; i++)
          {
        for (int u = 0; u < n; u++) {
          for (int k = 0; k < (int)g[u].size(); k++) {
            int id = g[u][k];
            int v = e[id].v;
            T cap = e[id].cap, w = e[id].cost;
            if (d[v] > d[u] + w && cap > 0) {
              d[v] = d[u] + w;
              relax = true;
            }
          }
        }
      }
      for (int i = 0; i < n; i++) if (d[i] < inf)
          potential[i] = d[i];
    }
    while (flow < goal && dijkstra()) flow +=
        send_flow(t, goal - flow);
    flow_through.assign(mxid + 10, 0);
    for (int u = 0; u < n; u++) {
      for (auto v : g[u]) {
        if (e[v].id >= 0) flow_through[e[v].id] = e[v ^
            1].cap;
      }
    }
    return make_pair(flow, cost);
  }
};
```

4 Game Theory

4.1 Points to be noted [14 lines] - 69c18f5f

```
>[First Write a Brute Force solution]
>Nim = all xor
>Misere Nim = Nim + corner case: if all piles are 1,
    reverse(nim)
>Bogus Nim = Nim
>Staircase Nim = Odd indexed pile Nim (Even indexed pile
    doesnt matter, as one player can give bogus moves to
    drop all even piles to ground)
>Sprague Grundy: [Every impartial game under the normal
    play convention is equivalent to a one-heap game of
    nim]
Every tree = one nim pile = tree root value; tree leaf
    value = 0; tree node value = mex of all child nodes.
[Careful: one tree node can become multiple new tree
    roots(multiple elements in one node), then the value
    of that node = xor of all those root values]
>Hackenbush(Given a rooted tree; cut an edge in one
    move; subtree under that edge gets removed; last
    player to cut wins):
 Colon:  //$G(u) = (G(v1) + 1) \oplus (G(v2) + 1) \oplus \cdots [v1, v2, \cdots$
    are childs of u]
For multiple trees ans is their xor
>Hackenbush on graph (instead of tree given an rooted
    graph):
fusion: All edges in a cycle can be fused to get a tree
    structure; build a super node, connect some single
    nodes with that super node, number of single nodes
    is the number of edges in the cycle.
Sol: [Bridge component tree] mark all bridges, a group
    of edges that are not bridges, becomes one component
    and contributes number of edges to the hackenbush.
    (even number of edges contributes 0, odd number of
    edges contributes 1)
```

5 Geometry

5.1 Geometry [870 lines] - 3810c300

```
int sign(T x) { return (x > eps) - (x < -eps); }
struct PT {
  T x, y;
  PT() { x = 0, y = 0; }
  PT(T x, T y) : x(x), y(y) {}
  PT(const PT &p) : x(p.x), y(p.y)  {}
  PT operator + (const PT &a) const { return PT(x +
      a.x, y + a.y); }
  PT operator - (const PT &a) const { return PT(x - a.x, y - a.y); }
  PT operator * (const T a) const { return PT(x * a, y * a); }
  friend PT operator * (const T &a, const PT &b) {
      return PT(a * b.x, a * b.y); }
  PT operator / (const T a) const { return PT(x / a, y / a); }
  bool operator == (PT a) const { return sign(a.x - x) == 0 && sign(a.y - y) == 0; }
  bool operator != (PT a) const { return !(*this == a); }
  bool operator < (PT a) const { return sign(a.x - x) == 0 ? y < a.y : x < a.x; }
  bool operator > (PT a) const { return sign(a.x - x) == 0 ? y > a.y : x > a.x; }
  T norm() { return sqrt(x * x + y * y); }
  T norm2() { return x * x + y * y; }
  PT perp() { return PT(-y, x); }
  T arg() { return atan2(y, x); }
  PT truncate(T r) { // returns a vector with norm r and having same direction
    T k = norm();
    if (!sign(k)) return *this;
    r /= k;
    return PT(x * r, y * r);
  }
};
istream &operator >> (istream &in, PT &p) { return in >> p.x >> p.y; }
ostream &operator << (ostream &out, PT &p) { return out << "(" << p.x << "," << p.y << ")"; }
inline T dot(PT a, PT b) { return a.x * b.x + a.y * b.y; }
inline T dist2(PT a, PT b) { return dot(a - b, a - b); }
inline T dist(PT a, PT b) { return sqrt(dot(a - b, a - b)); }
inline T cross(PT a, PT b) { return a.x * b.y - a.y * b.x; }
inline T cross2(PT a, PT b, PT c) { return cross(b - a, c - a); }
inline int orientation(PT a, PT b, PT c) { return sign(cross(b - a, c - a)); }
PT perp(PT a) { return PT(-a.y, a.x); }
PT rotateccw90(PT a) { return PT(-a.y, a.x); }
PT rotatecw90(PT a) { return PT(a.y, -a.x); }
PT rotateccw(PT a, T t) { return PT(a.x * cos(t) - a.y * sin(t), a.x * sin(t) + a.y * cos(t)); }
PT rotatecw(PT a, T t) { return PT(a.x * cos(t) + a.y * sin(t), -a.x * sin(t) + a.y * cos(t)); }
T rad_to_deg(T r) { return (r * 180.0 / PI); }
T deg_to_rad(T d) { return (d * PI / 180.0); }
T get_angle(PT a, PT b) {
  T costheta = dot(a, b) / a.norm() / b.norm();
  return acos(max((T)-1.0, min((T)1.0, costheta)));
}
bool is_point_in_angle(PT b, PT a, PT c, PT p) { // does point p lie in angle <bac
  assert(orientation(a, b, c) != 0);
  if (orientation(a, c, b) < 0) swap(b, c);
  return orientation(a, c, p) >= 0 && orientation(a, b, p) <= 0;
}
bool half(PT p) {
  return p.y > 0.0 || (p.y == 0.0 && p.x < 0.0);
}
void polar_sort(vector<PT> &v) { // sort points in counterclockwise
  sort(v.begin(), v.end(), [](PT a,PT b) {
    return make_tuple(half(a), 0.0, a.norm2()) <
        make_tuple(half(b), cross(a, b), b.norm2());
  });
}
void polar_sort(vector<PT> &v, PT o) { // sort points in counterclockwise with respect to point o
  sort(v.begin(), v.end(), [&](PT a,PT b) {
    return make_tuple(half(a - o), 0.0, (a - o).norm2())
        < make_tuple(half(b - o), cross(a - o, b - o), (b - o).norm2());
  });
}
struct line {
  PT a, b; // goes through points a and b
  PT v; T c;  //line form: direction vec [cross] (x, y) = c
  line() {}
  //direction vector v and offset c
  line(PT v, T c) : v(v), c(c) {
    auto p = get_points();
    a = p.first; b = p.second;
  }
  // equation ax + by + c = 0
  line(T _a, T _b, T _c) : v({_b, -_a}), c(-_c) {
  auto p = get_points();
    a = p.first; b = p.second;
  }
  // goes through points p and q
  line(PT p, PT q) : v(q - p), c(cross(v, p)), a(p), b(q) {}
   pair<PT, PT> get_points() { //extract any two points from this line
  PT p, q; T a = -v.y, b = v.x; // ax + by = c
  if (sign(a) == 0) {
    p = PT(0, c / b);
    q = PT(1, c / b);
  }
  else if (sign(b) == 0) {
    p = PT(c / a, 0);
    q = PT(c / a, 1);
  }
  else {
    p = PT(0, c / b);
    q = PT(1, (c - a) / b);
  }
  return {p, q};
   }
  // ax + by + c = 0
  array<T, 3> get_abc() {
    T a = -v.y, b = v.x;
    return {a, b, -c};
  }
  // 1 if on the left, -1 if on the right, 0 if on the line
  int side(PT p) { return sign(cross(v, p) - c); }
  // line that is perpendicular to this and goes through point p
  line perpendicular_through(PT p) { return {p, p + perp(v)}; }
  // translate the line by vector t i.e. shifting it by vector t
  line translate(PT t) { return {v, c + cross(v, t)}; }
  // compare two points by their orthogonal projection on this line
  // a projection point comes before another if it comes first according to vector v
  bool cmp_by_projection(PT p, PT q) { return dot(v, p) < dot(v, q); }
  line shift_left(T d) {
  PT z = v.perp().truncate(d);
  return line(a + z, b + z);
  }
};
// find a point from a through b with distance d
PT point_along_line(PT a, PT b, T d) {
  assert(a != b);
  return a + (((b - a) / (b - a).norm()) * d);
}
// projection point c onto line through a and b assuming a != b
PT project_from_point_to_line(PT a, PT b, PT c) {
  return a + (b - a) * dot(c - a, b - a) / (b - a).norm2();
}
// reflection point c onto line through a and b assuming a != b
PT reflection_from_point_to_line(PT a, PT b, PT c) {
  PT p = project_from_point_to_line(a,b,c);
  return p + p - c;
}
// minimum distance from point c to line through a and b
T dist_from_point_to_line(PT a, PT b, PT c) {
  return fabs(cross(b - a, c - a) / (b - a).norm());
}
// returns true if  point p is on line segment ab
bool is_point_on_seg(PT a, PT b, PT p) {
  if (fabs(cross(p - b, a - b)) < eps) {
    if (p.x < min(a.x, b.x) - eps || p.x > max(a.x, b.x) + eps) return false;
    if (p.y < min(a.y, b.y) - eps || p.y > max(a.y, b.y) + eps) return false;
    return true;
  }
  return false;
}
// minimum distance point from point c to segment ab that lies on segment ab
PT project_from_point_to_seg(PT a, PT b, PT c) {
  T r = dist2(a, b);
  if (sign(r) == 0) return a;
  r = dot(c - a, b - a) / r;
  if (r < 0) return a;
  if (r > 1) return b;
  return a + (b - a) * r;
}
// minimum distance from point c to segment ab
T dist_from_point_to_seg(PT a, PT b, PT c) {
  return dist(c, project_from_point_to_seg(a, b, c));
}
// 0 if not parallel, 1 if parallel, 2 if collinear
int is_parallel(PT a, PT b, PT c, PT d) {
  T k = fabs(cross(b - a, d - c));
  if (k < eps){
    if (fabs(cross(a - b, a - c)) < eps && fabs(cross(c
        - d, c - a)) < eps) return 2;
    else return 1;
  }
  else return 0;
}
// check if two lines are same
bool are_lines_same(PT a, PT b, PT c, PT d) {
  if (fabs(cross(a - c, c - d)) < eps && fabs(cross(b -
      c, c - d)) < eps) return true;
  return false;
}
// bisector vector of <abc
PT angle_bisector(PT &a, PT &b, PT &c){
  PT p = a - b, q = c - b;
  return p + q * sqrt(dot(p, p) / dot(q, q));
}
// 1 if point is ccw to the line, 2 if point is cw to
    the line, 3 if point is on the line
int point_line_relation(PT a, PT b, PT p) {
  int c = sign(cross(p - a, b - a));
  if (c < 0) return 1;
  if (c > 0) return 2;
  return 3;
}
// intersection point between ab and cd assuming unique
    intersection exists
bool line_line_intersection(PT a, PT b, PT c, PT d, PT
    &ans) {
  T a1 = a.y - b.y, b1 = b.x - a.x, c1 = cross(a, b);
  T a2 = c.y - d.y, b2 = d.x - c.x, c2 = cross(c, d);
  T det = a1 * b2 - a2 * b1;
  if (det == 0) return 0;
  ans = PT((b1 * c2 - b2 * c1) / det, (c1 * a2 - a1 *
      c2) / det);
  return 1;
}
// intersection point between segment ab and segment cd
    assuming unique intersection exists
bool seg_seg_intersection(PT a, PT b, PT c, PT d, PT
    &ans) {
  T oa = cross2(c, d, a), ob = cross2(c, d, b);
  T oc = cross2(a, b, c), od = cross2(a, b, d);
  if (oa * ob < 0 && oc * od < 0){
    ans = (a * ob - b * oa) / (ob - oa);
    return 1;
  }
  else return 0;
}
// intersection point between segment ab and segment cd
    assuming unique intersection may not exists
// se.size()==0 means no intersection
// se.size()==1 means one intersection
// se.size()==2 means range intersection
set<PT> seg_seg_intersection_inside(PT a,  PT b,  PT c,
    PT d) {
  PT ans;
  if (seg_seg_intersection(a, b, c, d, ans)) return
      {ans};
  set<PT> se;
  if (is_point_on_seg(c, d, a)) se.insert(a);
  if (is_point_on_seg(c, d, b)) se.insert(b);
  if (is_point_on_seg(a, b, c)) se.insert(c);
  if (is_point_on_seg(a, b, d)) se.insert(d);
  return se;
}
// intersection  between segment ab and line cd
// 0 if do not intersect, 1 if proper intersect, 2 if
    segment intersect
int seg_line_relation(PT a, PT b, PT c, PT d) {
  T p = cross2(c, d, a);
  T q = cross2(c, d, b);
  if (sign(p) == 0 && sign(q) == 0) return 2;
  else if (p * q < 0) return 1;
  else return 0;
}
// intersection between segament ab and line cd assuming
    unique intersection exists
bool seg_line_intersection(PT a, PT b, PT c, PT d, PT
    &ans) {
  bool k = seg_line_relation(a, b, c, d);
  assert(k != 2);
  if (k) line_line_intersection(a, b, c, d, ans);
  return k;
}
// minimum distance from segment ab to segment cd
T dist_from_seg_to_seg(PT a, PT b, PT c, PT d) {
  PT dummy;
  if (seg_seg_intersection(a, b, c, d, dummy)) return
      0.0;
  else return min({dist_from_point_to_seg(a, b, c),
      dist_from_point_to_seg(a, b, d),
    dist_from_point_to_seg(c, d, a),
        dist_from_point_to_seg(c, d, b)});
}
// minimum distance from point c to ray (starting point
    a and direction vector b)
T dist_from_point_to_ray(PT a, PT b, PT c) {
  b = a + b;
  T r = dot(c - a, b - a);
  if (r < 0.0) return dist(c, a);
  return dist_from_point_to_line(a, b, c);
}
// starting point as and direction vector ad
bool ray_ray_intersection(PT as, PT ad, PT bs, PT bd) {
  T dx = bs.x - as.x, dy = bs.y - as.y;
  T det = bd.x * ad.y - bd.y * ad.x;
  if (fabs(det) < eps) return 0;
  T u = (dy * bd.x - dx * bd.y) / det;
  T v = (dy * ad.x - dx * ad.y) / det;
  if (sign(u) >= 0 && sign(v) >= 0) return 1;
  else return 0;
}
T ray_ray_distance(PT as, PT ad, PT bs, PT bd) {
  if (ray_ray_intersection(as, ad, bs, bd)) return 0.0;
  T ans = dist_from_point_to_ray(as, ad, bs);
  ans = min(ans, dist_from_point_to_ray(bs, bd, as));
  return ans;
}
struct circle {
  PT p; T r;
  circle() {}
  circle(PT _p, T _r): p(_p), r(_r) {};
  // center (x, y) and radius r
  circle(T x, T y, T _r): p(PT(x, y)), r(_r) {};
  // circumcircle of a triangle
  // the three points must be unique
  circle(PT a, PT b, PT c) {
    b = (a + b) * 0.5;
    c = (a + c) * 0.5;
    line_line_intersection(b, b + rotatecw90(a - b), c,
        c + rotatecw90(a - c), p);
    r = dist(a, p);
  }
  // inscribed circle of a triangle
  // pass a bool just to differentiate from
      circumcircle
  circle(PT a, PT b, PT c, bool t) {
    line u, v;
    T m = atan2(b.y - a.y, b.x - a.x), n = atan2(c.y -
        a.y, c.x - a.x);
    u.a = a;
    u.b = u.a + (PT(cos((n + m)/2.0), sin((n +
        m)/2.0)));
    v.a = b;
    m = atan2(a.y - b.y, a.x - b.x), n = atan2(c.y -
        b.y, c.x - b.x);
    v.b = v.a + (PT(cos((n + m)/2.0), sin((n +
        m)/2.0)));
    line_line_intersection(u.a, u.b, v.a, v.b, p);
    r = dist_from_point_to_seg(a, b, p);
  }
  bool operator == (circle v) { return p == v.p &&
      sign(r - v.r) == 0; }
  T area() { return PI * r * r; }
  T circumference() { return 2.0 * PI * r; }
};
//0 if outside, 1 if on circumference, 2 if inside
    circle
int circle_point_relation(PT p, T r, PT b) {
  T d = dist(p, b);
  if (sign(d - r) < 0) return 2;
  if (sign(d - r) == 0) return 1;
  return 0;
}
// 0 if outside, 1 if on circumference, 2 if inside
    circle
int circle_line_relation(PT p, T r, PT a, PT b) {
  T d = dist_from_point_to_line(a, b, p);
  if (sign(d - r) < 0) return 2;
  if (sign(d - r) == 0) return 1;
  return 0;
}
//compute intersection of line through points a and b
    with
//circle centered at c with radius r > 0
vector<PT> circle_line_intersection(PT c, T r, PT a, PT
    b) {
  vector<PT> ret;
  b = b - a; a = a - c;
  T A = dot(b, b), B = dot(a, b);
  T C = dot(a, a) - r * r, D = B * B - A * C;
  if (D < -eps) return ret;
  ret.push_back(c + a + b * (-B + sqrt(D + eps)) / A);
  if (D > eps) ret.push_back(c + a + b * (-B - sqrt(D))
      / A);
  return ret;
}
//5 - outside and do not intersect
//4 - intersect outside in one point
//3 - intersect in 2 points
```

```
//2 - intersect inside in one point
//1 - inside and do not intersect
int circle_circle_relation(PT a, T r, PT b, T R) {
  T d = dist(a, b);
  if (sign(d - r - R) > 0)  return 5;
  if (sign(d - r - R) == 0) return 4;
  T l = fabs(r - R);
  if (sign(d - r - R) < 0 && sign(d - l) > 0) return 3;
  if (sign(d - l) == 0) return 2;
  if (sign(d - l) < 0) return 1;
  assert(0); return -1;
}
vector<PT> circle_circle_intersection(PT a, T r, PT b, T
    R) {
  if (a == b && sign(r - R) == 0) return {PT(1e18,
      1e18)};
  vector<PT> ret;
  T d = sqrt(dist2(a,  b));
  if (d > r + R || d + min(r,  R) < max(r,  R)) return
      ret;
  T x = (d * d - R * R + r * r) / (2 * d);
  T y = sqrt(r * r - x * x);
  PT v = (b - a) / d;
  ret.push_back(a + v * x  +  rotateccw90(v) * y);
  if (y > 0) ret.push_back(a + v * x - rotateccw90(v) *
      y);
  return ret;
}
// returns two circle c1, c2 through points a, b and of
    radius r
// 0 if there is no such circle, 1 if one circle, 2 if
    two circle
int get_circle(PT a, PT b, T r, circle &c1, circle &c2)
    {
  vector<PT> v = circle_circle_intersection(a, r, b, r);
  int t = v.size();
  if (!t) return 0;
  c1.p = v[0], c1.r = r;
  if (t == 2) c2.p = v[1], c2.r = r;
  return t;
}
// returns two circle c1, c2 which is tangent to line u,
    goes through
// point q and has radius r1; 0 for no circle, 1 if c1 =
    c2 , 2 if c1 != c2
int get_circle(line u, PT q, T r1, circle &c1, circle
    &c2) {
  T d = dist_from_point_to_line(u.a, u.b, q);
  if (sign(d - r1 * 2.0) > 0) return 0;
  if (sign(d) == 0) {
    cout << u.v.x << ' ' << u.v.y << '\n';
    c1.p = q + rotateccw90(u.v).truncate(r1);
    c2.p = q + rotatecw90(u.v).truncate(r1);
    c1.r = c2.r = r1;
    return 2;
  }
  line u1 = line(u.a + rotateccw90(u.v).truncate(r1),
      u.b + rotateccw90(u.v).truncate(r1));
  line u2 = line(u.a + rotatecw90(u.v).truncate(r1), u.b
      + rotatecw90(u.v).truncate(r1));
  circle cc = circle(q, r1);
  PT p1, p2; vector<PT> v;
  v = circle_line_intersection(q, r1, u1.a, u1.b);
  if (!v.size()) v = circle_line_intersection(q, r1,
      u2.a, u2.b);
  v.push_back(v[0]);
  p1 = v[0], p2 = v[1];
  c1 = circle(p1, r1);
  if (p1 == p2) {
    c2 = c1;
    return 1;
  }
  c2 = circle(p2, r1);
  return 2;
}
// returns area of intersection between two circles
T circle_circle_area(PT a, T r1, PT b, T r2) {
  T d = (a - b).norm();
  if(r1 + r2 < d + eps) return 0;
  if(r1 + d < r2 + eps) return PI * r1 * r1;
  if(r2 + d < r1 + eps) return PI * r2 * r2;
  T theta_1 = acos((r1 * r1 + d * d - r2 * r2) / (2 * r1
      * d)),
    theta_2 = acos((r2 * r2 + d * d - r1 * r1)/(2 * r2
      * d));
  return r1 * r1 * (theta_1 - sin(2 * theta_1)/2.) + r2
      * r2 * (theta_2 - sin(2 * theta_2)/2.);
}
// tangent lines from point q to the circle
int tangent_lines_from_point(PT p, T r, PT q, line &u,
    line &v) {
  int x = sign(dist2(p, q) - r * r);
  if (x < 0) return 0; // point in cricle
  if (x == 0) { // point on circle
    u = line(q, q + rotateccw90(q - p));
    v = u;
    return 1;
  }
  T d = dist(p, q);
  T l = r * r / d;
  T h = sqrt(r * r - l * l);
  u = line(q, p + ((q - p).truncate(l) + (rotateccw90(q
      - p).truncate(h))));
  v = line(q, p + ((q - p).truncate(l) + (rotatecw90(q
      - p).truncate(h))));
  return 2;
}
// returns outer tangents line of two circles
// if inner == 1 it returns inner tangent lines
int tangents_lines_from_circle(PT c1, T r1, PT c2, T
    r2, bool inner, line &u, line &v) {
  if (inner) r2 = -r2;
  PT d = c2 - c1;
  T dr = r1 - r2, d2 = d.norm2(), h2 = d2 - dr * dr;
  if (d2 == 0 || h2 < 0) {
    assert(h2 != 0);
    return 0;
  }
  vector<pair<PT, PT>>out;
  for (int tmp: {- 1, 1}) {
    PT v = (d * dr + rotateccw90(d) * sqrt(h2) * tmp) /
        d2;
    out.push_back({c1 + v * r1, c2 + v * r2});
  }
  u = line(out[0].first, out[0].second);
  if (out.size() == 2) v = line(out[1].first,
      out[1].second);
  return 1 + (h2 > 0);
}
// -1 if strictly inside, 0 if on the polygon, 1 if
    strictly outside
int is_point_in_triangle(PT a, PT b, PT c, PT p) {
  if (sign(cross(b - a,c - a)) < 0) swap(b, c);
  int c1 = sign(cross(b - a,p - a));
  int c2 = sign(cross(c - b,p - b));
  int c3 = sign(cross(a - c,p - c));
  if (c1<0 || c2<0 || c3 < 0) return 1;
  if (c1 + c2 + c3 != 3) return 0;
  return -1;
}
T perimeter(vector<PT> &p) {
  T ans=0; int n = p.size();
  for (int i = 0; i < n; i++) ans += dist(p[i], p[(i +
      1) % n]);
  return ans;
}
T area(vector<PT> &p) {
  T ans = 0; int n = p.size();
  for (int i = 0; i < n; i++) ans += cross(p[i], p[(i +
      1) % n]);
  return fabs(ans) * 0.5;
}
// centroid of a (possibly non-convex) polygon,
// assuming that the coordinates are listed in a
    clockwise or
// counterclockwise fashion.  Note that the centroid is
    often known as
// the "center of gravity" or "center of mass".
PT centroid(vector<PT> &p) {
  int n = p.size(); PT c(0, 0);
  T sum = 0;
  for (int i = 0; i < n; i++) sum += cross(p[i], p[(i +
      1) % n]);
  T scale = 3.0 * sum;
  for (int i = 0; i < n; i++) {
    int j = (i + 1) % n;
    c = c + (p[i] + p[j]) * cross(p[i], p[j]);
  }
  return c / scale;
}
// 0 if cw, 1 if ccw
bool get_direction(vector<PT> &p) {
  T ans = 0; int n = p.size();
  for (int i = 0; i < n; i++) ans += cross(p[i], p[(i +
      1) % n]);
  if (sign(ans) > 0) return 1;
  return 0;
}
// it returns a point such that the sum of distances
// from that point to all points in p  is minimum
// O(n log^2 MX)
PT geometric_median(vector<PT> p) {
  auto tot_dist = [&](PT z) {
    T res = 0;
    for (int i = 0; i < p.size(); i++) res +=
        dist(p[i], z);
    return res;
  };
  auto findY = [&](T x) {
    T yl = -1e5, yr = 1e5;
    for (int i = 0; i < 60; i++) {
      T ym1 = yl + (yr - yl) / 3;
      T ym2 = yr - (yr - yl) / 3;
      T d1 = tot_dist(PT(x, ym1));
      T d2 = tot_dist(PT(x, ym2));
      if (d1 < d2) yr = ym2;
      else yl = ym1;
    }
    return pair<T, T> (yl, tot_dist(PT(x, yl)));
  };
  T xl = -1e5, xr = 1e5;
  for (int i = 0; i < 60; i++) {
    T xm1 = xl + (xr - xl) / 3;
    T xm2 = xr - (xr - xl) / 3;
    T y1, d1, y2, d2;
    auto z = findY(xm1); y1 = z.first; d1 = z.second;
    z = findY(xm2); y2 = z.first; d2 = z.second;
    if (d1 < d2) xr = xm2;
    else xl = xm1;
  }
  return {xl, findY(xl).first };
}
vector<PT> convex_hull(vector<PT> &p) {
  if (p.size() <= 1) return p;
  vector<PT> v = p;
  sort(v.begin(), v.end());
  vector<PT> up, dn;
  for (auto& p : v) {
    while (up.size() > 1 && orientation(up[up.size() -
        2], up.back(), p) >= 0) {
      up.pop_back();
    }
    while (dn.size() > 1 && orientation(dn[dn.size() -
        2], dn.back(), p) <= 0) {
      dn.pop_back();
    }
    up.push_back(p);
    dn.push_back(p);
  }
  v = dn;
  if (v.size() > 1) v.pop_back();
  reverse(up.begin(), up.end());
  up.pop_back();
  for (auto& p : up) {
    v.push_back(p);
  }
  if (v.size() == 2 && v[0] == v[1]) v.pop_back();
  return v;
}
 //checks if convex or not
bool is_convex(vector<PT> &p) {
  bool s[3]; s[0] = s[1] = s[2] = 0;
  int n = p.size();
  for (int i = 0; i < n; i++) {
    int j = (i + 1) % n;
    int k = (j + 1) % n;
    s[sign(cross(p[j] - p[i], p[k] - p[i])) + 1] = 1;
    if (s[0] && s[2]) return 0;
  }
  return 1;
}
// -1 if strictly inside, 0 if on the polygon, 1 if
    strictly outside
// it must be strictly convex, otherwise make it
    strictly convex first
int is_point_in_convex(vector<PT> &p, const PT& x) { //
    O(log n)
  int n = p.size(); assert(n >= 3);
  int a = orientation(p[0], p[1], x), b =
      orientation(p[0], p[n - 1], x);
  if (a < 0 || b > 0) return 1;
  int l = 1, r = n - 1;
  while (l + 1 < r) {
    int mid = l + r >> 1;
    if (orientation(p[0], p[mid], x) >= 0) l = mid;
    else r = mid;
  }
  int k = orientation(p[l], p[r], x);
  if (k <= 0) return -k;
  if (l == 1 && a == 0) return 0;
  if (r == n - 1 && b == 0) return 0;
  return -1;
}
bool is_point_on_polygon(vector<PT> &p, const PT& z) {
  int n = p.size();
  for (int i = 0; i < n; i++) {
    if (is_point_on_seg(p[i], p[(i + 1) % n], z)) return
        1;
  }
  return 0;
}
// returns 1e9 if the point is on the polygon
int winding_number(vector<PT> &p, const PT& z) { //
    O(n)
  if (is_point_on_polygon(p, z)) return 1e9;
  int n = p.size(), ans = 0;
  for (int i = 0; i < n; ++i) {
    int j = (i + 1) % n;
    bool below = p[i].y < z.y;
    if (below != (p[j].y < z.y)) {
      auto orient = orientation(z, p[j], p[i]);
      if (orient == 0) return 0;
      if (below == (orient > 0)) ans += below ? 1 : -1;
    }
  }
  return ans;
}
// -1 if strictly inside, 0 if on the polygon, 1 if
    strictly outside
int is_point_in_polygon(vector<PT> &p, const PT& z) { //
    O(n)
  int k = winding_number(p, z);
  return k == 1e9 ? 0 : k == 0 ? 1 : -1;
}
// id of the vertex having maximum dot product with z
// polygon must need to be convex
// top - upper right vertex
// for minimum dot product negate z and return -dot(z,
    p[id])
int extreme_vertex(vector<PT> &p, const PT &z, const int
    top) { // O(log n)
  int n = p.size();
  if (n == 1) return 0;
  T ans = dot(p[0], z); int id = 0;
  if (dot(p[top], z) > ans) ans = dot(p[top], z), id =
      top;
  int l = 1, r = top - 1;
  while (l < r) {
    int mid = l + r >> 1;
    if (dot(p[mid + 1], z) >= dot(p[mid], z)) l = mid +
        1;
    else r = mid;
  }
  if (dot(p[l], z) > ans) ans = dot(p[l], z), id = l;
  l = top + 1, r = n - 1;
  while (l < r) {
    int mid = l + r >> 1;
    if (dot(p[(mid + 1) % n], z) >= dot(p[mid], z)) l =
        mid + 1;
    else r = mid;
  }
  l %= n;
  if (dot(p[l], z) > ans) ans = dot(p[l], z), id = l;
  return id;
}
// maximum distance from any point on the perimeter to
    another point on the perimeter
T diameter(vector<PT> &p) {
  int n = (int)p.size();
  if (n == 1) return 0;
  if (n == 2) return dist(p[0], p[1]);
  T ans = 0;
  int i = 0, j = 1;
  while (i < n) {
    while (cross(p[(i + 1) % n] - p[i], p[(j + 1) % n]
        - p[j]) >= 0) {
      ans = max(ans, dist2(p[i], p[j]));
      j = (j + 1) % n;
    }
    ans = max(ans, dist2(p[i], p[j]));
    i++;
  }
  return sqrt(ans);
}
// given n points, find the minimum enclosing circle of
    the points
// call convex_hull() before this for faster solution
// expected O(n)
circle minimum_enclosing_circle(vector<PT> &p) {
  random_shuffle(p.begin(), p.end());
  int n = p.size();
  circle c(p[0], 0);
  for (int i = 1; i < n; i++) {
    if (sign(dist(c.p, p[i]) - c.r) > 0) {
      c = circle(p[i], 0);
      for (int j = 0; j < i; j++) {
        if (sign(dist(c.p, p[j]) - c.r) > 0) {
          c = circle((p[i] + p[j]) / 2, dist(p[i], p[j])
              / 2);
          for (int k = 0; k < j; k++) {
            if (sign(dist(c.p, p[k]) - c.r) > 0) {
              c = circle(p[i], p[j], p[k]);
            }
          }
        }
      }
    }
  }
  return c;
}
// not necessarily convex, boundary is included in the
    intersection
// returns total intersected length
// it returns the sum of the lengths of the portions of
    the line that are inside the polygon
T polygon_line_intersection(vector<PT> p, PT a, PT b) {
  int n = p.size();
  p.push_back(p[0]);
  line l = line(a, b);
  T ans = 0.0;
  vector< pair<T, int> > vec;
  for (int i = 0; i < n; i++) {
    int s1 = orientation(a, b, p[i]);
    int s2 = orientation(a, b, p[i + 1]);
    if (s1 == s2) continue;
    line t = line(p[i], p[i + 1]);
    PT inter = (t.v * l.c - l.v * t.c) / cross(l.v,
        t.v);
    T tmp = dot(inter, l.v);
    int f;
    if (s1 > s2) f = s1 && s2 ? 2 : 1;
    else f = s1 && s2 ? -2 : -1;
    vec.push_back(make_pair((f > 0 ? tmp - eps : tmp +
        eps), f)); // keep eps very small like 1e-12
  }
  sort(vec.begin(), vec.end());
  for (int i = 0, j = 0; i + 1 < (int)vec.size(); i++){
    j += vec[i].second;
    if (j) ans += vec[i + 1].first - vec[i].first; // if
        this portion is inside the polygon
    // else ans = 0; // if we want the maximum
        intersected length which is totally inside the
        polygon, uncomment this and take the maximum of
        ans
  }
  ans = ans / sqrt(dot(l.v, l.v));
  p.pop_back();
  return ans;
}
// given a convex polygon p, and a line ab and the top
    vertex of the polygon
// returns the intersection of the line with the
    polygon
// it returns the indices of the edges of the polygon
    that are intersected by the line
// so if it returns i, then the line intersects the edge
    (p[i], p[(i + 1) % n])
array<int, 2> convex_line_intersection(vector<PT> &p, PT
    a, PT b, int top) {
  int end_a = extreme_vertex(p, (a - b).perp(), top);
  int end_b = extreme_vertex(p, (b - a).perp(), top);
  auto cmp_l = [&](int i) { return orientation(a, p[i],
      b); };
  if (cmp_l(end_a) < 0 || cmp_l(end_b) > 0)
    return {-1, -1}; // no intersection
  array<int, 2> res;
  for (int i = 0; i < 2; i++) {
    int lo = end_b, hi = end_a, n = p.size();
    while ((lo + 1) % n != hi) {
      int m = ((lo + hi + (lo < hi ? 0 : n)) / 2) % n;
      (cmp_l(m) == cmp_l(end_b) ? lo : hi) = m;
    }
    res[i] = (lo + !cmp_l(hi)) % n;
    swap(end_a, end_b);
  }
  if (res[0] == res[1]) return {res[0], -1}; // touches
      the vertex res[0]
  if (!cmp_l(res[0]) && !cmp_l(res[1]))
    switch ((res[0] - res[1] + (int)p.size() + 1) %
        p.size()) {
      case 0: return {res[0], res[0]}; // touches the
          edge (res[0], res[0] + 1)
      case 2: return {res[1], res[1]}; // touches the
          edge (res[1], res[1] + 1)
    }
  return res; // intersects the edges (res[0], res[0] +
      1) and (res[1], res[1] + 1)
}

pair<PT, int> point_poly_tangent(vector<PT> &p, PT Q,
    int dir, int l, int r) {
  while (r - l > 1) {
    int mid = (l + r) >> 1;
    bool pvs = orientation(Q, p[mid], p[mid - 1]) !=
        -dir;
    bool nxt = orientation(Q, p[mid], p[mid + 1]) !=
        -dir;
    if (pvs && nxt) return {p[mid], mid};
    if (!(pvs || nxt)) {
      auto p1 = point_poly_tangent(p, Q, dir, mid + 1,
          r);
      auto p2 = point_poly_tangent(p, Q, dir, l, mid -
          1);
      return orientation(Q, p1.first, p2.first) == dir ?
          p1 : p2;
    }
    if (!pvs) {
      if (orientation(Q, p[mid], p[l]) == dir)  r = mid
          - 1;
      else if (orientation(Q, p[l], p[r]) == dir) r =
          mid - 1;
      else l = mid + 1;
    }
    if (!nxt) {
      if (orientation(Q, p[mid], p[l]) == dir)  l = mid
          + 1;
      else if (orientation(Q, p[l], p[r]) == dir) r =
          mid - 1;
      else l = mid + 1;
    }
  }
  pair<PT, int> ret = {p[l], l};
  for (int i = l + 1; i <= r; i++) ret = orientation(Q,
      ret.first, p[i]) != dir ? make_pair(p[i], i) :
      ret;
  return ret;
}
// (ccw, cw) tangents from a point that is outside this
    convex polygon
// returns indexes of the points
// ccw means the tangent from Q to that point is in the
    same direction as the polygon ccw direction
pair<int, int>
    tangents_from_point_to_polygon(vector<PT> &p, PT Q){
  int ccw = point_poly_tangent(p, Q, 1, 0, (int)p.size()
      - 1).second;
  int cw = point_poly_tangent(p, Q, -1, 0, (int)p.size()
      - 1).second;
  return make_pair(ccw, cw);
}
// minimum distance from a point to a convex polygon
// it assumes point lie strictly outside the polygon
T dist_from_point_to_polygon(vector<PT> &p, PT z) {
  T ans = inf;
  int n = p.size();
  if (n <= 3) {
    for(int i = 0; i < n; i++) ans = min(ans,
        dist_from_point_to_seg(p[i], p[(i + 1) % n],
        z));
    return ans;
  }
  auto [r, l] = tangents_from_point_to_polygon(p, z);
  if(l > r) r += n;
  while (l < r) {
    int mid = (l + r) >> 1;
    T left = dist2(p[mid % n], z), right= dist2(p[(mid
        + 1) % n], z);
    ans = min({ans, left, right});
    if(left < right) r = mid;
    else l = mid + 1;
  }
  ans = sqrt(ans);
  ans = min(ans, dist_from_point_to_seg(p[l % n], p[(l
      + 1) % n], z));
  ans = min(ans, dist_from_point_to_seg(p[l % n], p[(l
      - 1 + n) % n], z));
  return ans;
}
// minimum distance from convex polygon p to line ab
// returns 0 is it intersects with the polygon
// top - upper right vertex
T dist_from_polygon_to_line(vector<PT> &p, PT a, PT b,
    int top) { //O(log n)
  PT orth = (b - a).perp();
  if (orientation(a, b, p[0]) > 0) orth = (a -
      b).perp();
  int id = extreme_vertex(p, orth, top);
  if (dot(p[id] - a, orth) > 0) return 0.0; //if orth
      and a are in the same half of the line, then poly
      and line intersects
  return dist_from_point_to_line(a, b, p[id]); //does
      not intersect
}
// minimum distance from a convex polygon to another
    convex polygon
// the polygon doesnot overlap or touch
T dist_from_polygon_to_polygon(vector<PT> &p1,
    vector<PT> &p2) { // O(n log n)
  T ans = inf;
  for (int i = 0; i < p1.size(); i++) {
    ans = min(ans, dist_from_point_to_polygon(p2,
        p1[i]));
  }
  for (int i = 0; i < p2.size(); i++) {
    ans = min(ans, dist_from_point_to_polygon(p1,
        p2[i]));
  }
  return ans;
}
// calculates the area of the union of n polygons (not
    necessarily convex).
// the points within each polygon must be given in CCW
    order.
```

```
// complexity: O(N^2), where N is the total number of
    points
T rat(PT a, PT b, PT p) {
    return !sign(a.x - b.x) ? (p.y - a.y) / (b.y - a.y)
        : (p.x - a.x) / (b.x - a.x);
 };
T polygon_union(vector<vector<PT>> &p) {
  int n = p.size();
  T ans=0;
  for(int i = 0; i < n; ++i) {
    for (int v = 0; v < (int)p[i].size(); ++v) {
      PT a = p[i][v], b = p[i][(v + 1) % p[i].size()];
      vector<pair<T, int>> segs;
      segs.emplace_back(0,  0), segs.emplace_back(1,
          0);
      for(int j = 0; j < n; ++j) {
        if(i != j) {
          for(size_t u = 0; u < p[j].size(); ++u) {
            PT c = p[j][u], d = p[j][(u + 1) %
                p[j].size()];
            int sc = sign(cross(b - a, c - a)), sd =
                sign(cross(b - a, d - a));
            if(!sc && !sd) {
              if(sign(dot(b - a, d - c)) > 0 && i > j) {
                segs.emplace_back(rat(a, b, c), 1),
                    segs.emplace_back(rat(a, b, d),
                    -1);
              }
            }
            else {
              T sa = cross(d - c, a - c), sb = cross(d
                  - c, b - c);
              if(sc >= 0 && sd < 0) segs.emplace_back(sa
                  / (sa - sb), 1);
              else if(sc < 0 && sd >= 0)
                  segs.emplace_back(sa / (sa - sb),
                  -1);
            }
          }
        }
      }
      sort(segs.begin(),  segs.end());
      T pre = min(max(segs[0].first, 0.0), 1.0), now,
          sum = 0;
      int cnt = segs[0].second;
      for(int j = 1; j < segs.size(); ++j) {
        now = min(max(segs[j].first, 0.0), 1.0);
        if (!cnt) sum += now - pre;
        cnt += segs[j].second;
        pre = now;
      }
      ans += cross(a, b) * sum;
    }
  }
  return ans * 0.5;
}
// returns the area of the intersection of the circle
    with center c and radius r
// and the triangle formed by the points c, a, b
T _triangle_circle_intersection(PT c, T r, PT a, PT b) {
  T sd1 = dist2(c, a), sd2 = dist2(c, b);
  if(sd1 > sd2) swap(a, b), swap(sd1, sd2);
  T sd = dist2(a, b);
  T d1 = sqrtl(sd1), d2 = sqrtl(sd2), d = sqrt(sd);
  T x = abs(sd2 - sd - sd1) / (2 * d);
  T h = sqrtl(sd1 - x * x);
  if(r >= d2) return h * d / 2;
  T area = 0;
  if(sd + sd1 < sd2) {
    if(r < d1) area = r * r * (acos(h / d2) - acos(h /
        d1)) / 2;
    else {
      area = r * r * ( acos(h / d2) - acos(h / r)) / 2;
      T y = sqrtl(r * r - h * h);
      area += h * (y - x) / 2;
    }
  }
  else {
    if(r < h) area = r * r * (acos(h / d2) + acos(h /
        d1)) / 2;
    else {
      area += r * r * (acos(h / d2) - acos(h / r)) / 2;
      T y = sqrtl(r * r - h * h);
      area += h * y / 2;
      if(r < d1) {
        area += r * r * (acos(h / d1) - acos(h / r)) /
            2;
        area += h * y / 2;
      }
      else area += h * x / 2;
    }
  }
  return area;
}
```

5.2 Rotation Matrix [39 lines] - d41f8b6c

```
struct { double x; double y; double z; } Point;
double rMat[4][4];
double inMat[4][1] = {0.0, 0.0, 0.0, 0.0};
double outMat[4][1] = {0.0, 0.0, 0.0, 0.0};
void mulMat() {
  for(int i = 0; i < 4; i++ ){
    for(int j = 0; j < 1; j++){
      outMat[i][j] = 0;
      for(int k = 0; k < 4; k++)
        outMat[i][j] += rMat[i][k] * inMat[k][j];
    }
  }
}
void setMat(double ang, double u, double v, double w){
  double L = (u * u + v * v + w * w);
  ang = ang * PI / 180.0; /*converting to radian
      value*/
  double u2 = u*u; double v2 = v*v; double w2 = w*w;
  rMat[0][0]=(u2+(v2+w2)*cos(ang))/L;
  rMat[0][1]=(u*v*(1-cos(ang))-w*sqrt(L)*sin(ang))/L;
  rMat[0][2]=(u*w*(1-cos(ang))+v*sqrt(L)*sin(ang))/L;
  rMat[0][3]=0.0;
  rMat[1][0]=(u*v*(1-cos(ang))+w*sqrt(L)*sin(ang))/L;
  rMat[1][1]=(v2+(u2+w2)*cos(ang))/L;
  rMat[1][2]=(v*w*(1-cos(ang))-u*sqrt(L)*sin(ang))/L;
  rMat[1][3]=0.0;
  rMat[2][0]=(u*w*(1-cos(ang))-v*sqrt(L)*sin(ang))/L;
  rMat[2][1]=(v*w*(1-cos(ang))+u*sqrt(L)*sin(ang))/L;
  rMat[2][2]=(w2 + (u2 + v2) * cos(ang)) / L;
  rMat[2][3]=0.0; rMat[3][0]=0.0; rMat[3][1]=0.0;
  rMat[3][2]=0.0; rMat[3][3]=1.0;
}
/*double ang;
  double u, v, w; //points = the point to be rotated
  Point point, rotated; //u,v,w=unit vector of line
  inMat[0][0] = points.x; inMat[1][0] = points.y;
  inMat[2][0] = points.z; inMat[3][0] = 1.0;
  setMat(ang, u, v, w); mulMat();
  rotated.x = outMat[0][0]; rotated.y = outMat[1][0];
  rotated.z = outMat[2][0];*/
```

6 Graph

6.1 2SAT [92 lines] - 94bbaab6

```
struct TwoSat {
  vector<bool>vis;
  vector<vector<int>>adj, radj;
  vector<int>dfs_t, ord, par;
  int n, intime;//For n node there will be 2*n node in
      SAT.
  void init(int N) {
    n = N;
    intime = 0;
    vis.assign(N * 2 + 1, false);
    adj.assign(N * 2 + 1, vector<int>());
    radj.assign(N * 2 + 1, vector<int>());
    dfs_t.resize(N * 2 + 1);
    ord.resize(N * 2 + 1);
    par.resize(N * 2 + 1);
  }
  inline int neg(int x) {
    return x <= n ? x + n : x - n;
  }
  inline void add_implication(int a, int b) {
    if (a < 0) a = n - a;
    if (b < 0) b = n - b;
    adj[a].push_back(b);
    radj[b].push_back(a);
  }
  inline void add_or(int a, int b) {
    add_implication(-a, b);
    add_implication(-b, a);
  }
  inline void add_xor(int a, int b) {
    add_or(a, b);
    add_or(-a, -b);
  }
  inline void add_and(int a, int b) {
    add_or(a, b);
    add_or(a, -b);
    add_or(-a, b);
  }
  inline void force_true(int x) {
    if (x < 0) x = n - x;
    add_implication(neg(x), x);
  }
  inline void add_xnor(int a, int b) {
    add_or(a, -b);
    add_or(-a, b);
  }
  inline void add_nand(int a, int b) {
    add_or(-a, -b);
  }
  inline void add_nor(int a, int b) {
    add_and(-a, -b);
  }
  inline void force_false(int x) {
    if (x < 0) x = n - x;
    add_implication(x, neg(x));
  }
  inline void topsort(int u) {
    vis[u] = 1;
    for (int v : radj[u]) if (!vis[v]) topsort(v);
    dfs_t[u] = ++intime;
  }
  inline void dfs(int u, int p) {
    par[u] = p, vis[u] = 1;
    for (int v : adj[u]) if (!vis[v]) dfs(v, p);
  }
  void build() {
    int i, x;
    for (i = n * 2, intime = 0;i >= 1;i--) {
      if (!vis[i]) topsort(i);
      ord[dfs_t[i]] = i;
    }
    vis.assign(n * 2 + 1, 0);
    for (i = n * 2;i > 0;i--) {
      x = ord[i];
      if (!vis[x]) dfs(x, x);
    }
  }
  bool satisfy(vector<int>& ret)//ret contains the value
      that are true if the graph is satisfiable.
  {
    build();
    vis.assign(n * 2 + 1, 0);
    for (int i = 1; i <= n * 2; i++) {
      int x = ord[i];
      if (par[x] == par[neg(x)]) return 0;
      if (!vis[par[x]]) {
        vis[par[x]] = 1;
        vis[par[neg(x)]] = 0;
      }
    }
    for (int i = 1;i <= n;i++) if (vis[par[i]])
        ret.push_back(i);
    return 1;
  }
};
```

6.2 BridgeTree [66 lines] - 4df9115e

```
int N, M, timer, compid;
vector<pair<int, int>> g[mx];
bool used[mx], isBridge[mx];
int comp[mx], tin[mx], minAncestor[mx];
vector<int> Tree[mx]; // Store 2-edge-connected
    component tree.(Bridge tree).
void markBridge(int v, int p) {
  tin[v] = minAncestor[v] = ++timer;
  used[v] = 1;
  for (auto& e : g[v]) {
    int to, id;
    tie(to, id) = e;
    if (to == p) continue;
    if (used[to]) minAncestor[v] = min(minAncestor[v],
        tin[to]);
    else {
      markBridge(to, v);
      minAncestor[v] = min(minAncestor[v],
          minAncestor[to]);
      if (minAncestor[to] > tin[v]) isBridge[id] = true;
      // if (tin[u] <= minAncestor[v]) ap[u] = 1;
    }
  }
}
void markComp(int v, int p) {
  used[v] = 1;
  comp[v] = compid;
  for (auto& e : g[v]) {
    int to, id;
    tie(to, id) = e;
    if (isBridge[id]) continue;
    if (used[to]) continue;
    markComp(to, v);
  }
}
vector<pair<int, int>> edges;
void addEdge(int from, int to, int id) {
  g[from].push_back({ to, id });
  g[to].push_back({ from, id });
  edges[id] = { from, to };
}
void initB() {
  for (int i = 0; i <= compid; ++i) Tree[i].clear();
  for (int i = 1; i <= N; ++i) used[i] = false;
  for (int i = 1; i <= M; ++i) isBridge[i] = false;
  timer = compid = 0;
}
void bridge_tree() {
  initB();
  markBridge(1, -1); //Assuming graph is connected.
  for (int i = 1; i <= N; ++i) used[i] = 0;
  for (int i = 1; i <= N; ++i) {
    if (!used[i]) {
      markComp(i, -1);
      ++compid;
    }
  }
  for (int i = 1; i <= M; ++i) {
    if (isBridge[i]) {
      int u, v;
      tie(u, v) = edges[i];
      // connect two components using edge.
      Tree[comp[u]].push_back(comp[v]);
      Tree[comp[v]].push_back(comp[u]);
      int x = comp[u];
      int y = comp[v];
    }
  }
}
```

6.3 Centroid Decomposition [39 lines] - f24b6f45

```
ll n,subsize[mx];
vector<int>adj[mx];
bool b[mx];
int cpar[mx];
vector<int>ctree[mx];

void calculatesize(ll u,ll par){
  subsize[u]=1;
  for(ll i=0;i<(ll)adj[u].size();i++){
    ll v=adj[u][i];
    if(v==par or b[v]==true)continue;
    calculatesize(v,u);
    subsize[u]+=subsize[v];
  }
}
ll getcentroid(ll u,ll par,ll n){
  ll ret=u;
  for(ll i=0;i<(ll)adj[u].size();i++){
    ll v=adj[u][i];
    if(v==par or b[v]==true)continue;
    if(subsize[v]>(n/2)){
      ret=getcentroid(v,u,n);
      break;
    }
  }
  return ret;
}
void decompose(ll u, int p){
  calculatesize(u,-1);
  ll c=getcentroid(u,-1,subsize[u]);
  b[c]=true;
  cpar[c] = p;
  //if(p != -1)ctree[p].push_back(c);
  for(ll i=0;i<(ll)adj[c].size();i++){
    ll v=adj[c][i];
    if(b[v]==true)continue;
    decompose(v, c);
  }
}
```

6.4 DSU on Tree [56 lines] - b5007715

```
int n;
//extra data you need
vector<int> adj[mxn];
vector<int> *dsu[mxn];
void call(int u, int p=-1){
  sz[u] = 1;
  for(auto v: adj[u]){
    if(v != p){
      dep[v] = dep[u]+1;
      call(v, u);
      sz[u] += sz[v];
    }
  }
}
void dfs(int u, int p = -1, int isb = 1){
  int mx=-1, big=-1;
  for(auto v: adj[u]){
    if(v != p && sz[v]>mx){
      mx = sz[v];
      big = v;
    }
  }
  for(auto v: adj[u]){
    if(v != p && v != big){
      dfs(v, u, 0);
    }
  }
  if(big != -1){
    dfs(big, u, 1);
    dsu[u] = dsu[big];
  }
  else{
    dsu[u] = new vector<int>();
  }
  dsu[u]->push_back(u);
  //calculation
  for(auto v: adj[u]){
    if(v == p || v == big) continue;
    for(auto x: *dsu[v]){
      dsu[u]->push_back(x);
      //calculation
    }
  }
  //calculate ans for node u
  if(isb == 0){
    for(auto x: *dsu[u]){
      //reverse calculation
    }
  }
}
int main() {
  //input graph
  dep[1] = 1;
  call(1);
  dfs(1);
}
```

6.5 Heavy Light Decomposition [73 lines] - 74d2c2ea

```
/*Heavy Light Decomposition
Build Complexity O(n)
Query Complexity O(lg^2 n)
Call init()with number of nodes
It's probably for the best to not do"using namespace
    hld"*/
namespace hld {
  //N is the maximum number of nodes
  /*par,lev,size corresponds to
      parent,depth,subtree-size*/
  //head[u]is the starting node of the chain u is in
  //in[u]to out[u]keeps the subtree indices
  const int N=100000+7;
  vector<int>g[N];
  int par[N],lev[N],head[N],size[N],in[N],out[N];
  int cur_pos,n;
  //returns the size of subtree rooted at u
  /*maintains the child with the largest subtree at the
      front of g[u]*/
  //WARNING: Don't change anything here specially with
      size[]if Jon Snow
  int dfs(int u,int p){
    size[u]=1,par[u]=p;
    lev[u]=lev[p]+1;
    for(auto &v : g[u]){
      if(v==p)continue;
      size[u]+=dfs(v,u);
      if(size[v]>size[g[u].front()]){
        swap(v,g[u].front());
      }
    }
    return size[u];
  }
  //decomposed the tree in an array
  //note that there is no physical array here
  void decompose(int u,int p){
    in[u]=++cur_pos;
    for(auto &v : g[u]){
      if(v==p)continue;
      head[v]=(v==g[u].front()? head[u]: v);
      decompose(v,u);
    }
    out[u]=cur_pos;
  }
  //initializes the structure with _n nodes
  void init(int _n,int root=1){
    n=_n;
    cur_pos=0;
    dfs(root,0);
    head[root]=root;
    decompose(root,0);
  }
  //checks whether p is an ancestor of u
  bool isances(int p,int u){
    return in[p]<=in[u]and out[u]<=out[p];
  }
  //Returns the maximum node value in the path u-v
  ll query(int u,int v){
    ll ret=-INF;
    while(!isances(head[u],v)){
      ret=max(ret,seg.query(1,1,n,in[head[u]],in[u]));
      u=par[head[u]];
    }
    swap(u,v);
    while(!isances(head[u],v)){
      ret=max(ret,seg.query(1,1,n,in[head[u]],in[u]));
      u=par[head[u]];
    }
    if(in[v]<in[u])swap(u,v);
    ret=max(ret,seg.query(1,1,n,in[u],in[v]));
    return ret;
  }
  //Adds val to subtree of u
  void update(int u,ll val){
    seg.update(1,1,n,in[u],out[u],val);
  }
};
```

6.6 K'th Shortest path [40 lines] - 1294302c

```
int m,n,deg[MM],source,sink,K,val[MM][12];
struct edge{
  int v,w;
}adj[MM][500];
struct info{
  int v,w,k;
  bool operator<(const info &b)const{
    return w>b.w;
  }
};
priority_queue<info,vector<info>>Q;
void kthBestShortestPath(){
  int i,j;
  info u,v;
  for(i=0;i<n;i++)
    for(j=0;j<K;j++)val[i][j]=inf;
  u.v=source,u.k=0,u.w=0;
  Q.push(u);
  while(!Q.empty()){
    u=Q.top();
    Q.pop();
    for(i=0;i<deg[u.v];i++){
      v.v=adj[u.v][i].v;
      int cost=adj[u.v][i].w+u.w;
      for(v.k=u.k;v.k<K;v.k++){
        if(cost==inf)break;
        if(val[v.v][v.k]>cost){
          swap(cost,val[v.v][v.k]);
          v.w=val[v.v][v.k];
          Q.push(v);
          break;
        }
      }
      for(v.k++;v.k<K;v.k++){
        if(cost==inf)break;
        if(val[v.v][v.k]>cost)swap(cost, val[v.v][v.k]);
      }
    }
  }
}
```

6.7 LCA [46 lines] - 3e689e11

```
const int Lg = 22;
vector<int>adj[mx];
int level[mx];
int dp[Lg][mx];
void dfs(int u) {
  for (int i = 1;i < Lg;i++)
    dp[i][u] = dp[i - 1][dp[i - 1][u]];
  for (int v : adj[u]) {
    if (dp[0][u] == v)continue;
    level[v] = level[u] + 1;
    dp[0][v] = u;
    dfs(v);
  }
}
int lca(int u, int v) {
  if (level[v] < level[u])swap(u, v);
  int diff = level[v] - level[u];
  for (int i = 0;i < Lg;i++)
    if (diff & (1 << i))
      v = dp[i][v];
  for (int i = Lg - 1;i >= 0;i--)
    if (dp[i][u] != dp[i][v])
      u = dp[i][u], v = dp[i][v];
  return u == v ? u : dp[0][u];
}
int kth(int u, int k) {
  for (int i = Lg - 1;i >= 0;i--)
    if (k & (1 << i))
      u = dp[i][u];
  return u;
}
//kth node from u to v. 0th is u.
int go(int u, int v, int k) {
  int l = lca(u, v);
  int d = level[u] + level[v] - (level[l] << 1);
  assert(k <= d);
  if (level[l] + k <= level[u]) return kth(u, k);
  k -= level[u] - level[l];
  return kth(v, level[v] - level[l] - k);
}
/*
   LCA(u,v) with root r:
   lca(u,v)^lca(u,r)^lca(v,r)
   Distance between u,v:
   level(u) + level(v) - 2*level(lca(u,v))
   */
```

6.8 SCC [43 lines] - 707e1cf1

```
/*components: number of SCC.
sz: size of each SCC.
comp: component number of each node.
```

```cpp
Create reverse graph.
Run find_scc() to find SCC.
Might need to create condensation graph by
    create_condensed().
Think about indeg/outdeg
for multiple test cases- clear
    adj/radj/comp/vis/sz/topo/condensed.*/
vector<int>adj[mx], radj[mx];

int comp[mx], vis[mx], sz[mx], components;
vector<int>topo;
void dfs(int u) {
  vis[u] = 1;
  for (int v : adj[u])
    if (!vis[v]) dfs(v);
  topo.push_back(u);
}
void dfs2(int u, int val) {
  comp[u] = val;
  sz[val]++;
  for (int v : radj[u])
    if (comp[v] == -1)
      dfs2(v, val);
}
void find_scc(int n) {
  memset(vis, 0, sizeof vis);
  memset(comp, -1, sizeof comp);
  for (int i = 1;i <= n;i++)
    if (!vis[i])
      dfs(i);
  reverse(topo.begin(), topo.end());
  for (int u : topo)
    if (comp[u] == -1)
      dfs2(u, ++components);
}
vector<int>condensed[mx];
void create_condensed(int n) {
  for (int i = 1;i <= n;i++)
    for (int v : adj[i])
      if (comp[i] != comp[v])
        condensed[comp[i]].push_back(comp[v]);
}
```

6.9 kuhn [31 lines] - bd4ddfd9

```cpp
int n, k;
vector<vector<int>> g;
vector<int> mt;
vector<bool> used;

bool try_kuhn(int v) {
    if (used[v])
        return false;
    used[v] = true;
    for (int to : g[v]) {
        if (mt[to] == -1 || try_kuhn(mt[to])) {
            mt[to] = v;
            return true;
        }
    }
    return false;
}

int main() {
    //... reading the graph ...

    mt.assign(k, -1);
    for (int v = 0; v < n; ++v) {
        used.assign(n, false);
        try_kuhn(v);
    }

    for (int i = 0; i < k; ++i)
        if (mt[i] != -1)
            printf("%d %d\n", mt[i] + 1, i + 1);
}
```

7 Math

7.1 Big Sum [13 lines] - 3c4fea67

```cpp
ll bigsum(ll a, ll b, ll m) {
  if (b == 0) return 0;
  ll sum; a %= m;
  if (b & 1) {
    sum = bigsum((a * a) % m, (b - 1) / 2, m);
    sum = (sum + (a * sum) % m) % m;
    sum = (1 + (a * sum) % m) % m;
  } else {
    sum = bigsum((a * a) % m, b / 2, m);
    sum = (sum + (a * sum) % m) % m;
  }
  return sum;
}
```

7.2 CRT [52 lines] - ff3bd658

```cpp
ll ext_gcd(ll A, ll B, ll* X, ll* Y) {
  ll x2, y2, x1, y1, x, y, r2, r1, q, r;
  x2 = 1; y2 = 0;
  x1 = 0; y1 = 1;
  for (r2 = A, r1 = B; r1 != 0; r2 = r1, r1 = r, x2 =
      x1, y2 = y1, x1 = x, y1 = y) {
    q = r2 / r1;
    r = r2 % r1;
    x = x2 - (q * x1);
    y = y2 - (q * y1);
  }
  *X = x2; *Y = y2;
  return r2;
}
/*------------BlackBox-------------*/
class ChineseRemainderTheorem {
  typedef long long vlong;
  typedef pair<vlong, vlong> pll;
  /** CRT Equations stored as pairs of vector. See
      addEquation()*/
  vector<pll> equations;
  public:
  void clear() {
    equations.clear();
  }
  /** Add equation of the form x = r (mod m)*/
  void addEquation(vlong r, vlong m) {
    equations.push_back({ r, m });
  }
  pll solve() {
    if (equations.size() == 0) return { -1,-1 }; /// No
        equations to solve
    vlong a1 = equations[0].first;
    vlong m1 = equations[0].second;
    a1 %= m1;
    /** Initially x = a_0 (mod m_0)*/
    /** Merge the solution with remaining equations */
    for (int i = 1; i < equations.size(); i++) {
      vlong a2 = equations[i].first;
      vlong m2 = equations[i].second;
      vlong g = __gcd(m1, m2);
      if (a1 % g != a2 % g) return { -1,-1 }; ///
          Conflict in equations
      /** Merge the two equations*/
      vlong p, q;
      ext_gcd(m1 / g, m2 / g, &p, &q);
      vlong mod = m1 / g * m2;
      vlong x = ((__int128)a1 * (m2 / g) % mod * q % mod
          + (__int128)a2 * (m1 / g) % mod * p % mod) %
          mod;
      /** Merged equation*/
      a1 = x;
      if (a1 < 0) a1 += mod;
      m1 = mod;
    }
    return { a1, m1 };
  }
};
```

7.3 FFT [85 lines] - 5ed04be4

```cpp
template<typename float_t>
struct mycomplex {
  float_t x, y;
  mycomplex<float_t>(float_t _x = 0, float_t _y = 0) :
      x(_x), y(_y) {}
  float_t real() const { return x; }
  float_t imag() const { return y; }
  void real(float_t _x) { x = _x; }
  void imag(float_t _y) { y = _y; }
  mycomplex<float_t>& operator+=(const
      mycomplex<float_t> &other) { x += other.x; y +=
      other.y; return *this; }
  mycomplex<float_t>& operator-=(const
      mycomplex<float_t> &other) { x -= other.x; y -=
      other.y; return *this; }
  mycomplex<float_t> operator+(const mycomplex<float_t>
      &other) const { return mycomplex<float_t>(*this)
      += other; }
  mycomplex<float_t> operator-(const mycomplex<float_t>
      &other) const { return mycomplex<float_t>(*this)
      -= other; }
  mycomplex<float_t> operator*(const mycomplex<float_t>
      &other) const {
    return {x * other.x - y * other.y, x * other.y +
        other.x * y};
  }
  mycomplex<float_t> operator*(float_t mult) const {
    return {x * mult, y * mult};
  }
  friend mycomplex<float_t> conj(const
      mycomplex<float_t> &c) {
    return {c.x, -c.y};
  }
  friend ostream& operator<<(ostream &stream, const
      mycomplex<float_t> &c) {
    return stream << '(' << c.x << ", " << c.y << ')';
  }
};
using cd = mycomplex<double>;
```

```
void fft(vector<cd> & a, bool invert) {
  int n = a.size();
  for (int i = 1, j = 0; i < n; i++) {
    int bit = n >> 1;
    for (; j & bit; bit >>= 1)
      j ^= bit;
    j ^= bit;
    if (i < j)
      swap(a[i], a[j]);
  }
  for (int len = 2; len <= n; len <<= 1) {
    double ang = 2 * PI / len * (invert ? -1 : 1);
    cd wlen(cos(ang), sin(ang));
    for (int i = 0; i < n; i += len) {
      cd w(1);
      for (int j = 0; j < len / 2; j++) {
        cd u = a[i+j], v = a[i+j+len/2] * w;
        a[i+j] = u + v;
        a[i+j+len/2] = u - v;
        w = w*wlen;
      }
    }
  }
  if (invert) {
    for (cd & x : a){
      double z = n;
      z=1/z;
      x = x*z;
    }
    // x /= n;
  }
}
void multiply (const vector<bool> & a, const
    vector<bool> & b, vector<bool> & res) {//change all
    the bool to your type needed
  vector<cd> fa (a.begin(), a.end()),  fb (b.begin(),
      b.end());
  size_t n = 1;
  while (n < max (a.size(), b.size()))  n <<= 1;
  n <<= 1;
  fa.resize (n),  fb.resize (n);
  fft (fa, false),  fft (fb, false);
  for (size_t i=0; i<n; ++i)
    fa[i] =fa[i] * fb[i];
  fft (fa, true);
  res.resize (n);
  for (size_t i=0; i<n; ++i)
    res[i] = round(fa[i].real());
  while(res.back()==0) res.pop_back();
}
void pow(const vector<bool> &a, vector<bool> &res, long
    long int k){
  vector<bool> po=a;
  res.resize(1);
  res[0] = 1;
  while(k){
    if(k&1){
      multiply(po, res, res);
    }
    multiply(po, po, po);
    k/=2;
  }
}
```

7.4 GaussElimination [39 lines] - 1203a4bb

```
template<typename ld>
int gauss(vector<vector<ld>>& a, vector<ld>& ans) {
  const ld EPS = 1e-9;
  int n = a.size();///number of equations
  int m = a[0].size() - 1;///number of variables
  vector<int>where(m, -1);///indicates which row
      contains the solution
  int row, col;
  for (col = 0, row = 0;col < m && row < n;++col) {
    int sel = row;///which row contains the maximum
        value/
    for (int i = row + 1;i < n;i++)
      if (abs(a[i][col]) > abs(a[sel][col]))
        sel = i;
    if (abs(a[sel][col]) < EPS) continue;///it's
        basically 0.
    a[sel].swap(a[row]);///taking the max row up
    where[col] = row;
    ld t = a[row][col];
    for (int i = col;i <= m;i++) a[row][i] /= t;
    for (int i = 0;i < n;i++) {
      if (i != row) {
        ld c = a[i][col];
        for (int j = col;j <= m;j++)
          a[i][j] -= a[row][j] * c;
      }
    }
    row++;
  }
  ans.assign(m, 0);
  for (int i = 0;i < m;i++)
    if (where[i] != -1)
      ans[i] = a[where[i]][m] / a[where[i]][i];
  for (int i = 0;i < n;i++) {
    ld sum = 0;
    for (int j = 0;j < m;j++)
      sum += ans[j] * a[i][j];
    if (abs(sum - a[i][m]) > EPS) ///L.H.S!=R.H.S
      ans.clear();//No solution
  }
  return row;
}
```

7.5 GaussMod2 [44 lines] - 1d49c381

```
template<typename T>
struct Gauss {
  int bits = 60;
  vector<T>table;
  Gauss() {
    table = vector<T>(bits, 0);
  }
  //call with constructor to define bit size.
  Gauss(int _bits) {
    bits = _bits;
    table = vector<T>(bits, 0);
  }
  int basis()//return rank/size of basis
  {
    int ans = 0;
    for (int i = 0;i < bits;i++)
      if (table[i])
        ans++;
    return ans;
  }
  bool can(T x)//can x be obtained from the basis
  {
    for (int i = bits - 1;i >= 0;i--) x = min(x, x ^
        table[i]);
    return x == 0;
  }
  void add(T x) {
    for (int i = bits - 1;i >= 0 && x;i--) {
      if (table[i] == 0) {
        table[i] = x;
        x = 0;
      }
      else x = min(x, x ^ table[i]);
    }
  }
  T getBest() {
    T x = 0;
    for (int i = bits - 1;i >= 0;i--)
      x = max(x, x ^ table[i]);
    return x;
  }
  void Merge(Gauss& other) {
    for (int i = bits - 1;i >= 0;i--)
        add(other.table[i]);
  }
};
```

7.6 Karatsuba Idea [5 lines] - 6686aa78

```
Three subproblems:
a = xH yH
d = xL yL
e = (xH + xL)(yH + yL) - a - d
Then xy = a rn + e rn/2 + d
```

7.7 Linear Diophatine [19 lines] - ebfad56a

```
int extended_gcd(ll a, ll b, ll& x, ll& y) {
  if (b == 0){x = 1;y = 0;return a;}
  ll x1, y1;
  ll d = extended_gcd(b, a % b, x1, y1);
  x = y1;y = x1 - y1 * (a / b);
  return d;
}
/*x'=x+(k*B/g),y'=y-(k*A/g);infinite soln
if A=B=0,C must equal 0 and any x,y is solution;
if A|B=0,(x,y)=(C/A,k)|(k,C/B)*/
bool LDE(ll A,ll B,ll C,ll &x,ll &y){
  int g=gcd(A,B);
  if(C%g!=0)return false;
  int a=A/g,b=B/g,c=C/g;
  extended_gcd(a,b,x,y);//ax+by=1
  if(g<0){a*=-1;b*=-1;c*=-1;}//Ensure gcd(a,b)=1
  x*=c;y*=c;//ax+by=c
  return true;//Solution Exists
}
```

7.8 Matrix [100 lines] - 60a4fb89

```
template<typename T>
struct Matrix {
  T MOD = 1e9 + 7;///change if necessary
  T add(T a, T b) const {
    T res = a + b;
    if (res >= MOD) return res - MOD;
    return res;
  }
```

```
T sub(T a, T b) const {
  T res = a - b;
  if (res < 0) return res + MOD;
  return res;
}
T mul(T a, T b) const {
  T res = a * b;
  if (res >= MOD) return res % MOD;
  return res;
}
int R, C;
vector<vector<T>>mat;
Matrix(int _R = 0, int _C = 0) {
  R = _R, C = _C;
  mat.resize(R);
  for (auto& v : mat) v.assign(C, 0);
}
void print() {
  for (int i = 0;i < R;i++)
    for (int j = 0;j < C;j++)
      cout << mat[i][j] << " \n"[j == C - 1];
}
void createIdentity() {
  for (int i = 0; i < R; i++)
    for (int j = 0; j < C; j++)
      mat[i][j] = (i == j);
}
Matrix operator+(const Matrix& o) const {
  Matrix res(R, C);
  for (int i = 0; i < R; i++)
    for (int j = 0; j < C; j++)
      res[i][j] = add(mat[i][j] + o.mat[i][j]);
}
Matrix operator-(const Matrix& o) const {
  Matrix res(R, C);
  for (int i = 0; i < R; i++)
    for (int j = 0; j < C; j++)
      res[i][j] = sub(mat[i][j] + o.mat[i][j]);
}
Matrix operator*(const Matrix& o) const {
  Matrix res(R, o.C);
  for (int i = 0; i < R; i++)
    for (int j = 0; j < o.C; j++)
      for (int k = 0;k < C;k++)
        res.mat[i][j] = add(res.mat[i][j],
            mul(mat[i][k], o.mat[k][j]));
  return res;
}
Matrix pow(long long x) {
  Matrix res(R, C);
  res.createIdentity();
  Matrix<T> o = *this;
  while (x) {
    if (x & 1) res = res * o;
    o = o * o;
    x >>= 1;
  }
  return res;
}
Matrix inverse()///Only square matrix && non-zero
    determinant
{
  Matrix res(R, R + R);
  for (int i = 0;i < R;i++) {
```

```
      for (int j = 0;j < R;j++)
        res.mat[i][j] = mat[i][j];
      res.mat[i][R + i] = 1;
    }
    for (int i = 0;i < R;i++) {
      ///find row 'r' with highest value at [r][i]
      int tr = i;
      for (int j = i + 1;j < R;j++)
        if (abs(res.mat[j][i]) > abs(res.mat[tr][i]))
          tr = j;
      ///swap the row
      res.mat[tr].swap(res.mat[i]);
      ///make 1 at [i][i]
      T val = res.mat[i][i];
      for (int j = 0;j < R + R;j++) res.mat[i][j] /=
          val;
      ///eliminate [r][i] from every row except i.
      for (int j = 0;j < R;j++) {
        if (j == i) continue;
        for (int k = R + R - 1;k >= i;k--) {
          res.mat[j][k] -= res.mat[i][k] * res.mat[j][i]
              / res.mat[i][i];
        }
      }
    }
    Matrix ans(R, R);
    for (int i = 0;i < R;i++)
      for (int j = 0;j < R;j++)
        ans.mat[i][j] = res.mat[i][R + j];
    return ans;
  }
};
```

7.9 Miller-Rabin-Pollard-Rho [68 lines] - 8307b44c

```
ll powmod(ll a, ll p, ll m) {///(a^p % m)
  ll result = 1;
  a %= m;
  while (p) {
    if (p & 1)
      result = (vll)result * a % m;
    a = (vll)a * a % m;
    p >>= 1;
  }
  return result;
}
bool check_composite(ll n, ll a, ll d, int s) {
  ll x = powmod(a, d, n);
  if (x == 1 || x == n - 1)
    return false;
  for (int r = 1; r < s; r++) {
    x = (vll)x * x % n;
    if (x == n - 1)
      return false;
  }
  return true;
}
bool MillerRabin(ll n) {
  if (n < 2) return false;
  int r = 0;
  ll d = n - 1;
  while ((d & 1) == 0) {
    d >>= 1;
    r++;
  }
  for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31,
      37}) {
    if (n == a) return true;
    if (check_composite(n, a, d, r))
      return false;
  }
  return true;
}
ll mult(ll a, ll b, ll mod) {
  return (vll)a * b % mod;
}
ll f(ll x, ll c, ll mod) {
  return (mult(x, x, mod) + c) % mod;
}
ll rho(ll n) {
  if (n % 2 == 0) return 2;
  ll x = myrand() % n + 1, y = x, c = myrand() % n + 1,
      g = 1;
  while (g == 1) {
    x = f(x, c, n);
    y = f(y, c, n);
    y = f(y, c, n);
    g = __gcd(abs(x - y), n);
  }
  return g;
}
set<ll>prime;
void prime_factorization(ll n) {
  if (n == 1) return;
  if (MillerRabin(n)) {
    prime.insert(n);
    return;
  }
  ll x = n;
  while (x == n) x = rho(n);
  prime_factorization(x);
  prime_factorization(n / x);
}
//call prime_factorization(n) for prime factors.
//call MillerRabin(n) to check if prime.
```

7.10 Mod Inverse [5 lines] - a45c7f67

```
int modInv(int a, int m) {
    int x, y; //if g==1 Inverse doesn't exist
    int g = gcdExt(a, m, x, y);
    return (x % m + m) % m;
}
```

7.11 NTT [96 lines] - b8108f51

```
ll power(ll a, ll p, ll mod) {
  if (p==0)   return 1;
  ll ans = power(a, p/2, mod);
  ans = (ans * ans)%mod;
  if(p%2)     ans = (ans * a)%mod;
  return ans;
}
int primitive_root(int p) {
  vector<int> factor;
  int phi = p-1, n = phi;
  for (int i=2; i*i<=n; i++) {
    if (n%i)    continue;
    factor.push_back(i);
    while (n%i==0)  n/=i;
  }
  if (n>1)    factor.push_back(n);
  for (int res =2; res<=p; res++) {
    bool ok = true;
    for (int i=0; i<factor.size() && ok; i++)
      ok &= power(res, phi/factor[i], p) != 1;
    if (ok) return res;
  }
  return -1;
}
int nttdata(int mod, int &root, int &inv, int &pw) {
  int c = 0, n = mod-1;
  while (n%2==0)  c++, n/=2;
  pw = (mod-1)/n;
  int g = primitive_root(mod);
  root = power(g, n, mod);
  inv = power(root, mod-2, mod);
  return c;
}
const int M = 786433;
struct NTT {
  int N;
  vector<int> perm;
  int mod, root, inv, pw;
  NTT(){}
  NTT(int mod, int root, int inv, int pw) : mod(mod),
      root(root), inv(inv), pw(pw) {}
  void precalculate() {
    perm.resize(N);
    perm[0] = 0;
    for (int k=1; k<N; k<<=1) {
      for (int i=0; i<k; i++) {
        perm[i] <<= 1;
        perm[i+k] = 1 + perm[i];
      }
    }
  }
  void fft(vector<ll> &v, bool invert = false) {
    if (v.size() != perm.size()) {
      N = v.size();
      assert(N && (N&(N-1)) == 0);
      precalculate();
    }
    for (int i=0; i<N; i++)
      if (i < perm[i])
        swap(v[i], v[perm[i]]);
    for (int len = 2; len <= N; len <<=1) {
      ll factor = invert ? inv: root;
      for (int i=len; i<pw; i<<=1)
        factor = (factor * factor) % mod;
      for (int i=0; i<N; i+=len) {
        ll w = 1;
        for (int j=0; j<len/2; j++) {
          ll x = v[i+j], y = (w*v[i+j+len/2])%mod;
          v[i+j] = (x+y)%mod;
          v[i+j+len/2] = (x-y+mod)%mod;
          w = (w*factor)%mod;
        }
      }
    }
    if (invert) {
      ll n1 = power(N, mod-2, mod);
      for (ll &x: v)  x = (x*n1)%mod;
    }
  }
  vector<ll> multiply(vector<ll> a, vector<ll> &b) {
    while (a.size() && a.back() == 0)   a.pop_back();
    while (b.size() && b.back() == 0)   b.pop_back();
    int n = 1;
    while (n < a.size() + b.size()) n<<=1;
    a.resize(n);
    b.resize(n);
    fft(a);
    fft(b);
    for (int i=0; i<n; i++) a[i] = (a[i] * b[i])%M;
    fft(a, true);
    while (a.size() && a.back() == 0)   a.pop_back();
    return a;
  }
//     int mod=786433, root, inv, pw;
//     nttdata(mod, root, inv, pw);
//     NTT nn = NTT(mod, root, inv, pw);
};
```

7.12 No of Digits in n! in base B [7 lines] - 21d4aeb2

```
ll NoOfDigitInNFactInBaseB(ll N,ll B){
  ll i;
  double ans=0;
  for(i=1;i<=N;i++)ans+=log(i);
  ans=ans/log(B),ans=ans+1;
  return(ll)ans;
}
```

7.13 SOD Upto N [16 lines] - 458dbeec

```
ll SOD_UpTo_N(ll N){
  ll i,j,ans=0;///upto N in Sqrt(N)
  for(i=1;i*i<=N;i++){
    j=N/i;
    ans+=((j*(j+1))/2)-(((i-1)*i)/2);
    ans+=((j-i)*i);
  }
  return ans;
}
ll SODUptoN(ll N){
  ll res=0,u=sqrt(N);
  for(ll i=1;i<=u;i++)
    res+=(N/i)-i;
  res*=2,res+=u;
  return res;
}
```

7.14 Sieve Phi Mobius [26 lines] - 966c3571

```
const int N = 1e7;
vector<int>pr;
int mu[N + 1], phi[N + 1], lp[N + 1];
void sieve() {
  phi[1] = 1, mu[1] = 1;
  for (int i = 2; i <= N; i++) {
    if (lp[i] == 0) {
      lp[i] = i;
      phi[i] = i - 1;
      pr.push_back(i);
    }
    for (int j = 0; j < pr.size() && i * pr[j] <= N;
        j++) {
      lp[i * pr[j]] = pr[j];
      if (i % pr[j] == 0) {
        phi[i * pr[j]] = phi[i] * pr[j];
        break;
      }
      else
        phi[i * pr[j]] = phi[i] * phi[pr[j]];
    }
  }
  for (int i = 2;i <= N;i++) {
    if (lp[i / lp[i]] == lp[i]) mu[i] = 0;
    else mu[i] = -1 * mu[i / lp[i]];
  }
}
```

8 Misc

8.1 Bit hacks [12 lines] - 5103c91a

```
# x & -x is the least bit in x.
# iterate over all the subsets of the mask
for (int s=m; ; s=(s-1)&m) {
 ... you can use s ...
 if (s==0)  break;
}
# c = x&-x, r = x+c; (((r^x) >> 2)/c) | r is the
next number after x with the same number of bits set.
# __builtin_popcount(x) //number of ones in binary
  __builtin_popcountll(x) // for long long
# __builtin_clz(x) // number of leading zeros
  __builtin_ctz(x) // number of trailing zeros, they
      also have long long version
```

8.2 Bitset C++ [13 lines] - 9c765780

```
bitset<17>BS;
BS[1] = BS[7] = 1;
cout<<BS._Find_first()<<endl; // prints 1
bs._Find_next(idx). This function returns first set bit
    after index idx.for example:

bitset<17>BS;
BS[1] = BS[7] = 1;
cout<<BS._Find_next(1)<<','<<BS._Find_next(3)<<endl; //
    prints 7,7
So this code will print all of the set bits of BS:

for(int i=BS._Find_first();i< BS.size();i =
    BS._Find_next(i))
    cout<<i<<endl;
//Note that there isn't any set bit after idx,
    BS._Find_next(idx) will return BS.size(); same as
    calling BS._Find_first() when bitset is clear;
```

8.3 Template [33 lines] - 52af6a79

```
// #pragma GCC optimize("O3,unroll-loops")
// #pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <typename A, typename B> ostream&
    operator<<(ostream& os, const pair<A, B>& p) {
    return os << '(' << p.first << ", " << p.second <<
    ')'; }
template <typename T_container, typename T = typename
    enable_if<!is_same<T_container, string>::value,
    typename T_container::value_type>::type> ostream&
    operator<<(ostream& os, const T_container& v) { os
    << '{'; string sep; for (const T& x : v) os << sep
    << x, sep = ", "; return os << '}'; }
```

```
void dbg_out() { cerr << endl; }
template <typename Head, typename... Tail> void
    dbg_out(Head H, Tail... T) { cerr << " " << H;
    dbg_out(T...); }

#ifdef SMIE
#define debug(args...) cerr << "(" << #args << "):",
    dbg_out(args)
#else
#define debug(args...)
#endif

template <typename T> inline T gcd(T a, T b) { T c;while
    (b) { c = b;b = a % b;a = c; }return a; } // better
    than __gcd
ll powmod(ll a, ll b, ll MOD) { ll res = 1;a %=
    MOD;assert(b >= 0);for (; b; b >>= 1) { if (b &
    1)res = res * a % MOD;a = a * a % MOD; }return res;
    }
template <typename T>using orderedSet = tree<T,
    null_type, less_equal<T>, rb_tree_tag,
    tree_order_statistics_node_update>;
//order_of_key(k) - number of element strictly less than
    k
//find_by_order(k) - k'th element in set.(0
    indexed)(iterator)

mt19937
rng(chrono::steady_clock::now().time_since_epoch()
    .count());
//uniform_int_distribution<int>(0, i)(rng)
int main(int argc, char* argv[]) {
  ios_base::sync_with_stdio(false);//DON'T CC++
  cin.tie(NULL);//DON'T use for interactive
  int seed = atoi(argv[1]);
}
```

8.4 build [2 lines] - c31604dc

```
#!/bin/bash
>&2 echo -e "Making [$2]\t: $1.cpp" && g++ -std=gnu++17
    -Wshadow -Wall -Wextra -Wno-unused-result -O2 -g
    -fsanitize=undefined -fsanitize=address $2 "$1.cpp"
    -o "$1"
```

8.5 check [15 lines] - d5ff3010

```
#!/bin/bash
build $1
TESTNO=0
for INP in $1.in*; do
  printf "\n=========\n"
  printf "INPUT %d" $TESTNO
  printf "\n=========\n"
    cat $INP
  printf "\n=========\n"
  printf "OUTPUT %d" $TESTNO
  printf "\n=========\n"
    ./$1 < $INP
    mv $INP $1.in$TESTNO 2>/dev/null
    TESTNO=$((TESTNO+1))
done
```

8.6 debug [3 lines] - 693f59a7

```
#!/bin/bash
build "$1" -DSMIE && >&2 echo -e "Running\t\t:
    $1\n------------------------------" && "./$1"
```

8.7 stress [15 lines] - af79edf1

```
#!/bin/bash
build $1 $2 && build $1_gen $2 && build $1_brute $2 &&
for((i = 1; ; ++i)); do
    echo -e "\nTest Case "$i
    ./$1_gen $i > inp
    ./$1 < inp > out1
    ./$1_brute < inp > out2
    diff -w out1 out2 || break
done
echo -e "===========\nINPUT\n-----"
cat inp
echo -e "\nOUTPUT\n------"
cat out1
echo -e "\nEXPECTED\n--------"
cat out2
```

8.8 vimrc [14 lines] - d9b3ef7e

```
filetype plugin indent on
set rnu wfw hls is ar aw wrap mouse=a

let mapleader=' '
im jk <esc>
tno jk <c-w>N
no <leader>d "_d
im {<cr> {<cr>}<esc>O
nn ff :let @+ = expand("%:p")<cr>
nn cd :cd %:h<cr>

au BufNewFile *.cpp -r ./template.cpp | 14

ca hash w !cpp -dD -P -fpreprocessed \| tr -d
    '[:space:]' \| md5sum \| cut -c-6
```

9 String

9.1 Aho-Corasick [124 lines] - ebba1edc

```
const int NODE=3000500;///Maximum Nodes
const int LGN=30;     ///Maximum Number of Tries
const int MXCHR=53;   ///Maximum Characters
const int MXP=5005;   ///
struct node {
  int val;
  int child[MXCHR];
  vector<int>graph;
  void clear(){
    CLR(child,0);
    val=0;
    graph.clear();
  }
}Trie[NODE+10];
int maxNodeId,fail[NODE+10],par[NODE+10];
int nodeSt[NODE+10],nodeEd[NODE+10];
vlong csum[NODE+10],pLoc[MXP];
void resetTrie(){
  maxNodeId=0;
}
int getNode(){
  int curNodeId=++maxNodeId;
  Trie[curNodeId].clear();
  return curNodeId;
}
inline void upd(vlong pos){
  csum[pos]++;
}
inline vlong qry(vlong pos){
  vlong res=csum[pos];
  return res;
}
struct AhoCorasick {
  int root,size,euler;
  void clear(){
    root=getNode();
    size=euler=0;
  }
  inline int getname(char ch){
    if(ch=='-')return 52;
    else if(ch>='A' && ch<='Z')return 26+(ch-'A');
    else return(ch-'a');
  }
  void addToTrie(string &s,int id){
  //Add string s to the Trie in general way
    int len=SZ(s),cur=root;
    FOR(i,0,len-1){
      int c=getname(s[i]);
      if(Trie[cur].child[c]==0){
        int curNodeId=getNode();
        Trie[curNodeId].val=c;
        Trie[cur].child[c]=curNodeId;
      }
      cur=Trie[cur].child[c];
    }
    pLoc[id]=cur;
    size++;
  }
  void calcFailFunction(){
    queue<int>Q;
    Q.push(root);
    while(!Q.empty()){
      int s=Q.front();
      Q.pop();
    //Add all the children to the queue:
      FOR(i,0,MXCHR-1){
        int t=Trie[s].child[i];
        if(t!=0){
          Q.push(t);
          par[t]=s;
        }
      }
      if(s==root){/*Handle special case when s is
          root*/
        fail[s]=par[s]=root;
        continue;
      }
 //Find fall back of s:
      int p=par[s],f=fail[p];;
      int val=Trie[s].val;
 /*Fall back till you found a node who has got val as a
     child*/
      while(f!=root && Trie[f].child[val]==0){
        f=fail[f];
      }
      fail[s]=(Trie[f].child[val]==0)? root :
          Trie[f].child[val];
 //Self fall back not allowed
      if(s==fail[s]){
        fail[s]=root;
```

```
   }
   Trie[fail[s]].graph.push_back(s);
  }
 }
 void dfs(int pos){
  ++euler;
  nodeSt[pos]=euler;
  for(auto x: Trie[pos].graph){
   dfs(x);
  }
  nodeEd[pos]=euler;
 }
 //Returns the next state
 int goTo(int state,int c){
  if(Trie[state].child[c]!=0){/*No need to fall
    back*/
   return Trie[state].child[c];
  }
 //Fall back now:
  int f=fail[state];
  while(f!=root && Trie[f].child[c]==0){
   f=fail[f];
  }
  int res=(Trie[f].child[c]==0)?
    root:Trie[f].child[c];
  return res;
 }
 /*Iterate through the whole text and find all the
   matchings*/
 void findmatching(string &s){
  int cur=root,idx=0;
  int len=SZ(s);
  while(idx<len){
   int c=getname(s[idx]);
   cur=goTo(cur,c);
   upd(nodeSt[cur]);
   idx++;
  }
 }
}acorasick;
```

9.2 Double Hasing [50 lines] - f434a7a8

```
struct SimpleHash {
  int len;
  long long base, mod;
  vector<int> P, H, R;
  SimpleHash() {}
  SimpleHash(string str, long long b, long long m) {
    base = b, mod = m, len = str.size();
    P.resize(len + 4, 1), H.resize(len + 3, 0),
      R.resize(len + 3, 0);
    for (int i = 1; i <= len + 3; i++)
      P[i] = (P[i - 1] * base) % mod;
    for (int i = 1; i <= len; i++)
      H[i] = (H[i - 1] * base + str[i - 1] + 1007)
        % mod;
    for (int i = len; i >= 1; i--)
      R[i] = (R[i + 1] * base + str[i - 1] + 1007)
        % mod;
  }
  inline int range_hash(int l, int r) {
    int hashval = H[r + 1] - ((long long)P[r - l +
      1] * H[l] % mod);
    return (hashval < 0 ? hashval + mod : hashval);
  }
  inline int reverse_hash(int l, int r) {
    int hashval = R[l + 1] - ((long long)P[r - l +
      1] * R[r + 2] % mod);
    return (hashval < 0 ? hashval + mod : hashval);
  }
};
struct DoubleHash {
  SimpleHash sh1, sh2;
  DoubleHash() {}
  DoubleHash(string str) {
    sh1 = SimpleHash(str, 1949313259, 2091573227);
    sh2 = SimpleHash(str, 1997293877, 2117566807);
  }
  long long concat(DoubleHash& B , int l1 , int r1 ,
    int l2 , int r2) {
    int len1 = r1 - l1+1 , len2 = r2 - l2+1;
    long long x1 = sh1.range_hash(l1, r1) ,
    x2 = B.sh1.range_hash(l2, r2);
    x1 = (x1 * B.sh1.P[len2]) % 2091573227;
    long long newx1 = (x1 + x2) % 2091573227;
    x1 = sh2.range_hash(l1, r1);
    x2 = B.sh2.range_hash(l2, r2);
    x1 = (x1 * B.sh2.P[len2]) % 2117566807;
    long long newx2 = (x1 + x2) % 2117566807;
    return (newx1 << 32) ^ newx2;
  }
  inline long long range_hash(int l, int r) {
    return ((long long)sh1.range_hash(l, r) << 32) ^
      sh2.range_hash(l, r);
  }
  inline long long reverse_hash(int l, int r) {
    return ((long long)sh1.reverse_hash(l, r) << 32)
      ^ sh2.reverse_hash(l, r);
  }
};
```

9.3 KMP [23 lines] - 378a30cd

```
char P[maxn],T[maxn];
int b[maxn],n,m;
void kmpPreprocess(){
 int i=0,j=-1;
 b[0]=-1;
 while(i<m){
  while(j>=0 and P[i]!=P[j])
   j=b[j];
   i++;j++;
   b[i]=j;
 }
}
void kmpSearch(){
 int i=0,j=0;
 while(i<n){
  while(j>=0 and T[i]!=P[j])
   j=b[j];
   i++;j++;
  if(j==m){
   //pattern found at index i-j
  }
 }
}
```

9.4 Manacher [16 lines] - 4ff0e22e

```
vector<int> manacher_odd(string s) {
 int n = s.size();
 s = "$" + s + "^";
 vector<int> p(n + 2);
 int l = 1, r = 1;
 for(int i = 1; i <= n; i++) {
  p[i] = max(0, min(r - i, p[l + (r - i)]));
  while(s[i - p[i]] == s[i + p[i]]) {
   p[i]++;
  }
  if(i + p[i] > r) {
   l = i - p[i], r = i + p[i];
  }
 }
 return vector<int>(begin(p) + 1, end(p) - 1);
}
```

9.5 Palindromic Tree [30 lines] - b398a8f0

```
struct PalindromicTree{
 int n,idx,t;
 vector<vector<int>> tree;
 vector<int> len,link;
 string s; // 1-indexed
 PalindromicTree(string str){
  s="$"+str;
  n=s.size();
  len.assign(n+5,0);
  link.assign(n+5,0);
  tree.assign(n+5,vector<int>(26,0));
 }
 void extend(int p){
  while(s[p-len[t]-1]!=s[p]) t=link[t];
  int x=link[t],c=s[p]-'a';
  while(s[p-len[x]-1]!=s[p]) x=link[x];
  if(!tree[t][c]){
   tree[t][c]=++idx;
   len[idx]=len[t]+2;
   link[idx]=len[idx]==1?2:tree[x][c];
  }
  t=tree[t][c];
 }
 void build(){
  len[1]=-1,link[1]=1;
  len[2]=0,link[2]=1;
  idx=t=2;
  for(int i=1;i<n;i++) extend(i);
 }
};
```

9.6 Prefix Function Automaton [21 lines] - 5a2cc30b

```
/* create prefix function array in 26n.*/

int aut[mxn][26];
int lps[mxn];

void automaton(string &s){
 int n = s.size();
 aut[0][s[0] - 'a'] = 1;
 for(int i = 1; i < n; i++){
  for(int j = 0; j < 26; j++){
   if(j == s[i] - 'a'){
    aut[i][j] = i + 1;
    lps[i + 1] = aut[lps[i]][j];
   }
   else {
    aut[i][j] = aut[lps[i]][j];
```

```
      }
    }
    cout << lps[i + 1] << endl;
  }
}
```

9.7 Suffix Array [78 lines] - 582667ab

```
struct SuffixArray {
  vector<int> p, c, rank, lcp;
  vector<vector<int>> st;
  SuffixArray(string const& s) {
    build_suffix(s + char(1));
    build_rank(p.size());
    build_lcp(s + char(1));
    build_sparse_table(lcp.size());
  }
  void build_suffix(string const& s) {
    int n = s.size();
    const int MX_ASCII = 256;
    vector<int> cnt(max(MX_ASCII, n), 0);
    p.resize(n); c.resize(n);
    for (int i = 0; i < n; i++) cnt[s[i]]++;
    for (int i=1; i<MX_ASCII; i++) cnt[i]+=cnt[i-1];
    for (int i = 0; i < n; i++) p[--cnt[s[i]]] = i;
    c[p[0]] = 0;
    int classes = 1;
    for (int i = 1; i < n; i++) {
      if (s[p[i]] != s[p[i-1]]) classes++;
      c[p[i]] = classes - 1;
    }
    vector<int> pn(n), cn(n);
    for (int h = 0; (1 << h) < n; ++h) {
      for (int i = 0; i < n; i++) {
        pn[i] = p[i] - (1 << h);
        if (pn[i] < 0) pn[i] += n;
      }
      fill(cnt.begin(), cnt.begin() + classes, 0);
      for (int i = 0; i < n; i++) cnt[c[pn[i]]]++;
      for (int i=1; i<classes; i++) cnt[i]+=cnt[i-1];
      for (int i=n-1;i>=0;i--) p[--cnt[c[pn[i]]]]=pn[i];
      cn[p[0]] = 0; classes = 1;
      for (int i = 1; i < n; i++) {
        pair<int, int> cur = {c[p[i]], c[(p[i] + (1 <<
            h)) % n]};
        pair<int, int> prev = {c[p[i-1]], c[(p[i-1] + (1
            << h)) % n]};
        if (cur != prev) ++classes;
        cn[p[i]] = classes - 1;
      }
      c.swap(cn);
    }
  }
  void build_rank(int n) {
    rank.resize(n, 0);
    for (int i = 0; i < n; i++) rank[p[i]] = i;
  }
  void build_lcp(string const& s) {
    int n = s.size(), k = 0;
    lcp.resize(n - 1, 0);
    for (int i = 0; i < n; i++) {
      if (rank[i] == n - 1) {
        k = 0;
        continue;
      }
      int j = p[rank[i] + 1];
      while (i + k < n && j + k < n && s[i+k] == s[j+k])
        k++;
      lcp[rank[i]] = k;
      if (k) k--;
    }
  }
  void build_sparse_table(int n) {
    int lim = __lg(n);
    st.resize(lim + 1, vector<int>(n)); st[0] = lcp;
    for (int k = 1; k <= lim; k++)
      for (int i = 0; i + (1 << k) <= n; i++)
        st[k][i] = min(st[k - 1][i], st[k - 1][i + (1 <<
            (k - 1))]);
  }
  int get_lcp(int i) { return lcp[i]; }
  int get_lcp(int i, int j) {
    if (j < i) swap(i, j);
    j--; /*for lcp from i to j we don't need last lcp*/
    int K = __lg(j - i + 1);
    return min(st[K][i], st[K][j - (1 << K) + 1]);
  }
};
```

9.8 Suffix Automata [109 lines] - 16b40d71

```
const int mxc = 26;
/*
  + link      - longest suffix belonging to another
      endpos-equivalent class.
  + len       - largest string length ending in current
      state.
  + firstPos - first occurance of substring ending at
      current state.
  + adj       - suffix link tree.
  + sz        - number of states.
  + occ       - number of times state occured in string.
  + dist      - number of distinct substring.
  + cnt & SA - for count sorting the nodes.
*/
struct SuffixAutomata{
  struct state{
    int link, len, firstPos;
    int next[mxc];
    bool is_clone;
    state(){}
    state(int l){
      len = l, link = -1;
      is_clone = false;
      for(int i=0;i<mxc;i++)next[i] = -1;
    }
  };
  vector<state>t;
  int sz, last;
  vector<ll>cnt,dist, occ,SA;
  vector<vector<int>> adj;
  SuffixAutomata(){
    t.pb(state(0));
    occ.pb(0);
    last = sz = 0;
  }
  int getID(char c){ return c - 'a';}
  void extend(char c){
    int idx = ++sz, p = last, id = getID(c);
    t.pb(state(t[last].len + 1));
    t[idx].firstPos = t[idx].len - 1;
    occ.pb(1);
    while(p!= -1 and t[p].next[id] == - 1){
      t[p].next[id] = idx;
      p = t[p].link;
    }
    if(p==-1) t[idx].link = 0;
    else{
      int q = t[p].next[id];
      if(t[p].len+1 == t[q].len) t[idx].link = q;
      else{
        int clone = ++sz;
        state x = t[q];
        x.len = t[p].len+1;
        t.pb(x);
        t[clone].firstPos = t[q].firstPos;
        t[clone].is_clone = true;
        occ.pb(0);
        while(p!=-1 and t[p].next[id]==q){
          t[p].next[id] = clone;
          p = t[p].link;
        }
        t[idx].link = t[q].link = clone;
      }
    }
    last = idx;
  }
  void build(string &s){
    for(char c:s) extend(c);
    cnt = dist = SA = vector<ll>(sz+1);
    adj.resize(sz+1);
    for(int i=0;i<=sz;i++)cnt[t[i].len]++;
    for(int i=1;i<=sz;i++)cnt[i]+=cnt[i-1];
    for(int i=0;i<=sz;i++) SA[--cnt[t[i].len]] = i;

    for(int i=sz;i>0;i--){
      int idx = SA[i];
      occ[t[idx].link]+=occ[idx];
      adj[t[idx].link].pb(idx);
      dist[idx] = 1;
      for(int j=0;j<mxc;j++){
        if(t[idx].next[j]+1){
          dist[idx]+=dist[t[idx].next[j]];
        }
      }
    }
    for(int i=0;i<mxc;i++){
      if(t[0].next[i]+1) dist[0]+=dist[t[0].next[i]];
    }
  }
  pair<int,int> LCS( string& s){
    int mxlen = 0, bestpos = -1, pos = 0,len = 0;
    int u = 0;
    for(char c:s){
      int v = getID(c);
      while( u and t[u].next[v]!=-1){
        u = t[u].link;
        len = t[u].len;
      }
      if(t[u].next[v]+1){
        len++;
        u = t[u].next[v];
      }
      if(len>mxlen){
        mxlen = len;
        bestpos = pos;
      }
      pos++;
    }
    return {bestpos - mxlen + 1, mxlen};
  }
  state &operator[](int index) { return t[index];}
};
```

9.9 Trie [28 lines] - 6b8f900b

```
const int maxn=100005;
struct Trie{
  int next[27][maxn];
  int endmark[maxn],sz;
  bool created[maxn];
  void insertTrie(string& s){
    int v=0;
    for(int i=0;i<(int)s.size();i++){
      int c=s[i]-'a';
      if(!created[next[c][v]]){
        next[c][v]=++sz;
        created[sz]=true;
      }
      v=next[c][v];
    }
    endmark[v]++;
  }
  bool searchTrie(string& s){
    int v=0;
    for(int i=0;i<(int)s.size();i++){
      int c=s[i]-'a';
      if(!created[next[c][v]])
        return false;
      v=next[c][v];
    }
    return(endmark[v]>0);
  }
};
```

9.10 Z-Algorithm [19 lines] - 3e017493

```
void compute_z_function(const char*S,int N){
  int L=0,R=0;
  for(int i=1;i<N;++i){
    if(i>R){
      L=R=i;
      while(R<N && S[R-L]==S[R])++R;
      Z[i]=R-L,--R;
    }
    else{
      int k=i-L;
      if(Z[k]<R-i+1)Z[i]=Z[k];
      else{
        L=i;
        while(R<N && S[R-k]==S[R])++R;
        Z[i]=R-L,--R;
      }
    }
  }
}
```

10 Random

10.1 Combinatorics

- $\sum\limits_{k=0}^{n} \binom{n-k}{k} = Fib_{n+1}$
- $\binom{n}{k} + \binom{n}{k+1} = \binom{n+1}{k+1}$
- $k\binom{n}{k} = n\binom{n-1}{k-1}$
- Number of binary sequences of length n such that no two 0's are adjacent $= Fib_{n+1}$
- Number of non-negative solution of $x_1 + x_2 + x_3 + ... + x_k = n$ is $\binom{n+k-1}{n}$

10.1.1 Catalan Number

- $C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \frac{(2n)!}{(n+1)!n!}$
- $C_0 = 1, C_1 = 1, C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}$
- $1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786$
- Number of correct bracket sequences consisting of n opening brackets.
- Number of ways to completely parenthesize n+1 factors.
- The number of triangulations of a convex polygon with +2 sides (i.e. the number of partitions of polygon into disjoint triangles by using the diagonals).
- The number of ways to connect the 2n points on a circle to form n disjoint i.e. non-intersecting chords.
- The number of monotonic lattice paths from point (0,0) to point (n,n) in a square lattice of size $n \times n$, which do not pass above the main diagonal
- Number of permutation of length n that can be stack sorted.
- The number of non-crossing partitions of a set of n elements.
- The number of rooted full binary tree with n+1 leaves.
- The number of Dyck words of length 2n. A string consisting of n X's and n Y's such that no string prefix has more Y's than X's.
- Number of permutation of length n with no three-term increasing subsequence.
- Number of ways to tile a stairstep shape of height n with n rectangle.
- $C_n^k = \frac{k+1}{n+1}\binom{2n-k}{n-k}$ denote the number of bracket sequences of size 2n with the first k elements being (.
- $N(n,k) = \frac{1}{n}\binom{n}{k}\binom{n}{k-1}$
- The number of expressions containing n pairs of correct parentheses, which contain k distinct nestings. $N(4,2) = 6$ $()((())), (())(()), (()(())), ((()())), ((())()), ((()))()$
- The number of paths from (0,0) to (2n, 0) with steps only northeast and southeast, not staying below the x-axis with k peaks. And sum of all number of peaks is Catalan number.

10.1.2 Stirling Number of the First Kind

- Count permutation according to their number of cycles.
- $S(n,k)$ count the number of permutation of n elements with k disjoint cycles.
- $S(n,k) = (n-1) \times S(n-1,k) + S(n-1,k-1), S(0,0) = 1, S(n,0) = S(0,n) = 0$
- $S(n,1) = (n-1)!$
- $S(n,n-1) = \binom{n}{2}$
- $\sum_{k=0}^{n} S(n,k) = n!$

10.1.3 Stirling Numbers of the Second Kind

- Number of ways to partition a set of n objects into k non-empty subsets.
- $S(n,k) = k * S(n-1,k) + S(n-1,k-1), S(0,0) = 1, S(n,0) = S(0,n) = 0$
- $S(n,2) = 2^{n-1} - 1$
- $S(n,k) = \frac{1}{k!}\sum\limits_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n$
- $S(n,k) * k! =$ number of ways to color n nodes using colors from 1 to k such that each color is used at least once.

10.1.4 Bell Number

- Counts the number of partitions of a set.
- $B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} * B_k$
- $B_n = \sum_{k=0}^{n} S(n,k)$, where S is Stirling number of second kind.
- The number of multiplicative partitions of a square free number with i prime factors is the i-th Bell number.
- $B(p^m + n) \equiv mB(n) + B(n+1) (\mod p)$
- If a deck is shuffled by removing and reinserting the top card n times, there are n^n possible shuffles. The number of shuffles that return the deck to its original order is B_n, so the probability of returning to the original order is B_n/n^n.

10.1.5 Lucas Theorem

- If p is prime then $\binom{p^a}{k} \equiv 0 \mod p$
- For non-negative integers m and n and a prime p: $\binom{m}{n} = \prod_{i=0}^{k} \binom{m_i}{n_i} (\mod p)$ where $m = m_k p^k + m_{k-1} p^{k-1} + ... + m_1 p + m_0$ $n = n_k p^k + n_{k-1} p^{k-1} + ... + n_1 p + n_0$ are the base p expansion.

10.1.6 Derangement

- A permutation such that no element appears in its original position.
- $d(n) = (n-1) * (d(n-1) + d(n-2)), d(0) = 1, d(1) = 0$
- $d(n) = nd(n-1) + (-1)^n = \lfloor \frac{n!}{e} \rfloor, n \geq 1$

10.1.7 Burnside Lemma

Given a group G of symmetries and a set X, the number of elements of X up to symmetry equals

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

where X^g are the elements fixed by $g(g.x = x)$ If f(n) counts "configurations" of some sort of length n, we can ignore rotational symmetry using $G = \mathbb{Z}_n$ to get

$$g(n) = \frac{1}{n} \sum_{k=0}^{n-1} f(gcd(n,k)) = \frac{1}{n} \sum_{k|n} f(k)\phi(n/k)$$

10.1.8 Eulerian Number

- $E(n,k)$ is the number of permutations of the numbers 1 to n in which exactly k elements are greater than the previous element.
- $E(n,k) = (n-k)E(n-1,k-1) + (k+1)E(n-1,k), E(n,0) = E(n,n-1) = 1$
- $E(n,k) = \sum_{j=0}^{k} (-1)^j \binom{n+1}{j} (k+1-j)^n$
- $E(n,k) = E(n, n-1-k)$
- $E(0,k) = [k = 0]$
- $E(n,1) = 2^n - n - 1$

10.2 Number Theory

10.2.1 Mobius Function and Inversion

- define $\mu(n)$ as the sum of the primitive nth roots of unity depending on the factorization of n into prime factors:

$$\mu(x) = \begin{cases} 0 & \text{n is not square free} \\ 1 & \text{n has even number of prime factors} \\ -1 & \text{n has odd number of prime factors} \end{cases}$$

- Mobius Inversion:

$$g(n) = \sum_{d|n} f(d) \leftrightarrow f(n) = \sum_{d|n} \mu(d) g(n/d)$$

- $\sum_{d|n} \mu(d) = [n = 1]$
- $\phi(n) = \sum_{d|n} \mu(d).\frac{n}{d} = n \sum_{d|n} \frac{\mu(d)}{d} = \sum_{d|n} d.\mu(\frac{n}{d})$
- $a|b \to \phi(a)|\phi(b)$
- $\phi(mn) = \phi(m).\phi(n).\frac{d}{\phi(d)}$ where $d = gcd(m,n)$
- $\phi(n^m) = n^{m-1}\phi(n)$
- $\sum_{i=1}^{n} [gcd(i,n) = k] = \phi(\frac{n}{k})$
- $\sum_{i=1}^{n} gcd(i,n) = \sum_{d|n} d.\phi(\frac{n}{d})$
- $\sum_{i=1}^{n} \frac{1}{gcd(i,n)} = \sum_{d|n} \frac{1}{d}.\phi(\frac{n}{d}) = \frac{1}{n} \sum_{d|n} d.\phi(d)$
- $\sum_{i=1}^{n} \frac{i}{gcd(i,n)} = \frac{n}{2}. \sum_{d|n} \frac{1}{d}.\phi(\frac{n}{d}) = \frac{n}{2}.\frac{1}{n} \sum_{d|n} d.\phi(d)$
- $\sum_{i=1}^{n} \frac{n}{gcd(i,n)} = 2. \sum_{i=1}^{n} \frac{i}{gcd(i,n)} - 1$

10.2.2 GCD and LCM

- gcd(a,b) = gcd(b, a mod b)
- If $a|b.c$, and gcd(a,b) = d, then $(a/d)|c$.
- GCD is a multiplicative function.
- gcd(a, lcm(b,c)) = lcm(gcd(a,b), gcd(a,c))
- $gcd(n^a - 1, n^b - 1) = n^{gcd(a,b)} - 1$

10.2.3 Gauss Circle Theorem

- Determine the number of lattice points in a circle centered at the origin with radius r.
- number of pairs (m,n) such that $m^2 + n^2 \leq r^2$
- $N(r) = 1 + 4 \sum_{i=0}^{\infty} (\lfloor \frac{r^2}{4i+1} \rfloor - \lfloor \frac{r^2}{4i+3} \rfloor)$

10.2.4 Pick's Theorem

According to Pick's Theorem We can calculate the area of any polygon by just counting the number of Interior and Boundary lattice points of that polygon. If number of interior points are I and number of boundary lattice points are B then Area (A) of polygon will be:

$$Area = I + B/2 - 1$$

where I is the number of points in the interior shape, B stands for the number of points on the boundary of the shape.

10.2.5 Formula Cheatsheet

- $\sum_{i=1}^{n} = \frac{1}{m+1}[(n+1)^{m+1} - 1 - \sum_{i=1}^{n}((i+1)^{m+1} - i^{m+1} - (m+1)i^m)]$
- $\sum_{i=0}^{n} c^i = \frac{c^{n+1}-1}{c-1}, c \neq 1$
- $\sum_{i=0}^{\infty} c^i = \frac{1}{1-c}, \sum_{i=1}^{\infty} c^i = \frac{c}{1-c}, |c| < 1$
- $H_n = \sum_{i=1}^{n} \frac{1}{n}, \sum_{i=1}^{n} iH_i = \frac{n(n+1)}{2} H_n - \frac{n(n-1)}{4}$
- $\sum_{k=0}^{n} \binom{r+k}{k} = \binom{r+n+1}{n}$