KUET_NlogN Team Notebook

Kazi Rifat Al Muin, Siyam Khan,
Hassan Mohammad Naquibul Hoque

Contents

1 Data Structure

1.1 DSU [23 lines] - 72e19576

```
vector <ll> par, sum, diff, sz;
ll find(ll a) {
    if(par[a] == a) return a;
    ll root = find(par[a]); // recursion should be
        called first to update diff & par
    diff[a] += diff[par[a]]; // adds all parent offset
    return par[a] = root;
}
void merge(ll a, ll b){
    ll ra = find(a);
    ll rb = find(b);
    if(ra != rb){
        if(sz[ra] <= sz[rb]){
            par[ra] = rb;
            sz[rb] += sz[ra];
            diff[ra] = sum[ra] - sum[rb];
        }
        else{
            par[rb] = ra;
            sz[ra] += sz[rb];
            diff[rb] = sum[rb] - sum[ra];
        }
    }
}
```

1.2 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;
  }
};
```

1.3 MO with Update [43 lines] - a0826346

```
//1 indexed
//Complexity:O(S × Q + Q × 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();
  }
}
```

1.4 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. */
```

1.5 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 Lazy [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 Segment Tree [41 lines] - 17c5e235

```
struct node{
    ll sum, maxi, mini;
    node() { sum = 0; maxi = -1e17; mini = 1e17; }
};
node tree[N*4];
node merge(node a, node b){
    node ans;
    ans.sum = a.sum + b.sum;
    ans.mini = min(a.mini, b.mini);
    ans.maxi = max(a.maxi, b.maxi);
    return ans;
}
void build(int id, int l, int r){ // (1, 0, n-1)
    if(l == r){
        tree[id] = node();
        return;
    }
    ll mid = (l+r)/2;
    build(2*id, l, mid);
    build(2*id + 1, mid+1, r);
    tree[id] = merge(tree[2*id], tree[2*id + 1]);
}
void update(int id, int l, int r, int pos, ll val){
    if(pos < l || pos > r) return;
    if(l == r){
        tree[id].sum = val;
        tree[id].mini = val;
        tree[id].maxi = val;
        return;
    }
    ll mid = (l+r)/2;
    update(2*id, l, mid, pos, val);
    update(2*id + 1, mid+1, r, pos, val);
    tree[id] = merge(tree[2*id], tree[2*id + 1]);
}
node query(int id, int l, int r, int lq, int rq){
    if(lq > r || rq < l) return node();
    if(l >= lq && r <= rq) return tree[id];
    ll mid = (l+r)/2;
    return merge(query(2*id, l, mid, lq, rq), query(2*id
        + 1, mid+1, r, lq, rq));
}
```

1.8 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
```

1.9 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 Digit Dp [19 lines] - f18e4b74

```
ll dp[20][20][2]; // How many zeros
ll digitDP(const string &num, ll pos = 0, ll cnt = 0,
    bool tight = 1, bool isStart = 1){
    if(pos == num.size()) return isStart ? 1 : cnt;
    else if(dp[pos][cnt][tight] != -1) return
        dp[pos][cnt][tight];
    ll ans = 0, lim = tight ? num[pos] - '0' : 9;
    for(int digit = 0; digit <= lim; digit++){
        ans += digitDP(num, pos + 1, cnt + (!isStart &&
            digit == 0), tight && digit == lim, isStart
            && digit == 0);
    }
    return isStart ? ans : dp[pos][cnt][tight] = ans;
}
void solve() {
    ll l, r;
    cin >> l >> r;
    memset(dp, -1, sizeof(dp));
    ll ans1 = digitDP(to_string(r));
    memset(dp, -1, sizeof(dp));
    ll ans2 = digitDP(to_string(l - 1));
    cout << ans1 - ans2 << "\n";
}
```

2.2 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);
}
```

2.3 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;
}
```

2.4 Knapsack DP [12 lines] - ef62fd72

```
// unbounded knapsack: ascending
for(auto x : v){
    for(int i = x; i <= N; i++){
        dp[i] = (dp[i] + dp[i-x]) % mod;
    }
}
// bounded knapsack: descending
for(auto x : v){
    for(int i = N; i >= x; i--){
        dp[i] = (dp[i] + dp[i-x]) % mod;
    }
}
```

2.5 Knuth Optimization [32 lines] - 5f2f74dc

```cpp
/*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];
```

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

```cpp
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];
}
```

2.7 SOS DP [18 lines] - e5398562

```cpp
//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)];
}
```

2.8 Sibling DP [26 lines] - 95945016

```cpp
/*/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 [59 lines] - a41c2f2c

```cpp
// Finds Maximum matching in General Graph
// Complexity O(NM)
// mate[i] = j means i is paired with j
// Usage: `auto mate = Blossom(graph);` where `graph` is
    vector<vector<int>>.
// 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 Flow [21 lines] - ae00b72c

```
Covering Problems:
- Maximum Independent Set (Bipartite): V - MaxMatching
- Minimum Vertex Cover (Bipartite): = MaxMatching
- Minimum Edge Cover (General): V - MaxMatching (no
    isolated vertices)
- Min Path Cover (vertex-disjoint, DAG): V - MaxMatching
    (reduce to bipartite)
- Min Path Cover (general): contract cycles -> DAG
    reduction
Matching | problem & when:
- Find maximum set of edges with no shared endpoints
    (bipartite case).
- Use for assignment, scheduling, pairing; Konig gives
    min vertex cover from matching.

Templates (how to use):
- Kuhn: set `n,k,g`; `mt.assign(k,-1);
    for(v=0..n-1){used.assign(n,false); try_kuhn(v);}`
```

```
  -> result: `mt[r]=l` or -1; complexity O(n*m).
- Hopcroft{Karp: `HK hk(L); hk.addEdge(u,v); int M =
    hk.max_match();`
  -> use `matchL`/`matchR`; complexity O(E*sqrt(V)).
- Blossom (`code/Flow/Blossom.cpp`): `auto mate =
    Blossom(graph);` for general graphs.
- Flow reduction: connect `s->L(1)`, `L->R(1)`,
    `R->t(1)`; run max-flow; unit flows on `L->R` are
    matches.
Tips:
- Prefer Hopcroft{Karp for bipartite; Blossom for
    non-bipartite.
- Seed Kuhn with a greedy matching to speed it; use
    Kőnig to recover min vertex cover.
- Keep adjacency 0-based for Kuhn/Blossom; this repo's
    HK is 1-based.
```

3.3 HopCroftKarp [66 lines] - 779699b6

```cpp
/*.Finds Maximum Matching In a bipartite graph
  .Complexity O(E√V)
  .1-indexed
  .No default constructor
  .add single edge for (u, v)
  Usage: `HK hk(L); hk.addEdge(u,v); int M =
      hk.max_match();` */
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;
  }
};
```

3.4 kuhn [34 lines] - 8b05d01e

```cpp
// Problem: find a maximum matching in a bipartite graph
     (no two chosen edges share a vertex).
// Where to use: assignment problems, scheduling,
     pairing tasks/agents, Kőnig's theorem for
// minimum vertex cover in bipartite graphs, minimum
     path cover in DAGs (via reduction).
// Brief usage: set `n` = left size, `k` = right size;
     fill `g[0..n-1]` with right indices (0..k-1).
// Call: `mt.assign(k, -1); for (int v=0; v<n; ++v) {
     used.assign(n,false); try_kuhn(v); }`
// Result: `mt[r] = l` if right r matched with left l,
     else -1. Complexity: O(n*m).
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);
}
```

4 Game Theory

4.1 Inclusion Exclusion with Nim [33 lines] - 1a428d17

```cpp
#define CheckBit(x, k) ((x >> k) & 1ll)
bool NimGame(vector <ll> v){
    ll nimsum = 0;
    for(auto x : v) nimsum ^= x;
    return nimsum != 0; // Alice win, If last pick win
}
void solve()
{
    ll n, m;
    cin >> n >> m;
    vector <ll> v(m), h(n), jinish(n);
    for(int i = 0; i < n; i++) cin >> h[i];
    for(int i = 0; i < m; i++) cin >> v[i];
    reverse(v.begin(), v.end());
    for(int j = 0; j < n; j++) {
        ll marked = 0;
        for(ll mask = 1; mask < 1ll << m; mask++){
            vector <ll> taken;
            for(int i = 0; i < 32; i++){
                if(CheckBit(mask, i))
                    taken.push_back(v[i]);
            }
            ll lcm = taken[0], ok = 1;
            for(int i = 1; i < taken.size(); i++){
                lcm = (lcm * taken[i]) / __gcd(lcm,
                    taken[i]);
            }
            if(taken.size() & 1) marked += (h[j] / lcm);
            else marked -= (h[j] / lcm);
        }
        jinish[j] = marked + 1;
    }
    if(NimGame(jinish)) cout << "Alice\n";
    else cout << "Bob\n";
}
```

4.2 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) ⊕ (G(v2) + 1) ⊕ ··· [v1, v2, ···
    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 Basic Geometry [113 lines] - 00878049

```
typedef long long ll;
typedef long double ld;
#define PI acos(-1.0)
#define eps 1e-7
#define point pair <double,double>
#define x first
#define y second

point operator + (point a, point b) { return {a.x + b.x, a.y + b.y}; }
point operator - (point a, point b) { return {a.x - b.x, a.y - b.y}; }
double operator | (point a, point b) { return a.x * b.x + a.y * b.y; }
double operator * (point a, point b) { return a.x * b.y - a.y * b.x; }

point operator * (point a, double m) { return {a.x * m, a.y * m}; }
point operator / (point a, double m) { return {a.x / m, a.y / m}; }
double val(point a) { return sqrt(a | a); }
tuple <double, double, double> pointToLine(point a, point b){
    return {( b.y - a.y), (a.x - b.x), (a.y * (b.x - a.x) - a.x * (b.y - a.y)) };
}
pair <point, point> lineToPoint(double a, double b, double c){
    if(a == 0) return {{1, -1 * c/b}, {0, -1 * c/b}};
    else if(b == 0) return {{-1 * c/a, 1}, {-1 * c/a, 0}};
    else return {{-1 * c/a, 0}, {0, -1 * c/b}};
}

double pointLineDistance(point p, double a, double b, double c){
    return fabs(a * (p.x) + b * (p.y) + c) / (sqrt(a*a + b*b));
}
double pointLineDistance(point p, point a, point b){
    return fabs((p-a) * (b-a)) / val(b-a);
}
double pointRayDistance(point p, point a, point b){
    if(((p-a)|(b-a)) < 0) return val(p-a);
    else return fabs((p-a) * (b-a)) / val(b-a);
}
double pointSegmentDistance(point p, point a, point b){
    if(((p-a)|(b-a)) < 0 && ((p-b)|(a-b)) > 0) return val(p-a);
    else if(((p-a)|(b-a)) > 0 && ((p-b)|(a-b)) < 0) return val(p-b);
    else return fabs((p-a) * (b-a)) / val(b-a);
}
double segmentSegmentDistance(point a, point b, point c, point d){
    bool dif1 = ((b-a)*(c-a) >= 0 && (b-a)*(d-a) <= 0) || ((b-a)*(c-a) <= 0 && (b-a)*(d-a) >= 0);
    bool dif2 = ((d-c)*(a-c) >= 0 && (d-c)*(b-c) <= 0) || ((d-c)*(a-c) <= 0 && (d-c)*(b-c) >= 0);
    if(dif1 == true && dif2 == true) return 0;
    else return min({
        pointSegmentDistance(a,c,d),
            pointSegmentDistance(b,c,d),
        pointSegmentDistance(c,a,b),
            pointSegmentDistance(d,a,b)
    });
}
point intersection(double a1, double b1, double c1, double a2, double b2, double c2){
    double x = (b1 * c2 - b2 * c1) / (a1 * b2 - a2 * b1);
    double y = (c1 * a2 - c2 * a1) / (a1 * b2 - a2 * b1);
    return {x, y};
}
point intersection(point a, point b, point c, point d){
    auto [a1, b1, c1] = pointToLine(a, b);
    auto [a2, b2, c2] = pointToLine(c, d);
    return intersection(a1,b1,c1,a2,b2,c2);
}
bool isOnLine(point p, double a, double b, double c){
    return fabs(a*p.x + b*p.y + c) <= eps; }
bool isOnRay(point p, point a, point b){ return fabs((p-a)*(b-a)) <= eps && ((p-a)|(b-a)) >= 0; }
bool isOnSegment(point p, point a, point b){ return fabs(val(p-a) + val(b-p) - val(b-a)) <= eps; }
bool isParallel(point a, point b, point c, point d) {
    return fabs((b-a)*(d-a)) <= eps; }
bool isSameSide(point p, point q, point p1, point p2){
    if(((p-p1) * (p2-p1)) * ((q-p1) * (p2-p1)) >= 0)
        return true;
    else return false;
}
bool isSameSide(point p, point q, double a, double b, double c){
    return (a*p.x + b*p.y + c) * (a*q.x + b*q.y + c) >= 0;
}
double rayRayDistance(point a, point b, point c, point d){
    double ans = min(pointRayDistance(a,c,d), pointRayDistance(c,a,b));
    if(isParallel(a,b,c,d)) return ans;
    else if(isOnRay(intersection(a,b,c,d), a, b) && isOnRay(intersection(a,b,c,d), c, d)) return 0;
    else return ans;
}
double area_of_triangle(point a, point b, point c){
    return fabs((a - c) * (b - c)) / 2.0;
}
double area_of_polygon(vector<point> &p){
    double area = 0.0;
    int n = p.size();
    for(int i = 0; i < n; i++){
        area += (p[i] * p[(i+1)%n]) / 2.0; // anticlockwise = +ve area, clockwise = -ve area
    }
    return fabs(area);
}
// Angle Bisector
point p = ((y - x) * val(z - x)) / val(y-x); // vector towards (y-x) with length |z-x|
p = p + (z - x); // resultant vector
p = p + x; // translating start point at x from (0,0)
auto [a, b, c] = pointToLine(x, p);


// Formula for Basic Geometry Operations
circumradius = (a * b * c) / (4 * area); //pori
inradius = area / s; //onto
isosceles_side = (b / 4) * sqrt(4 * a * a - b * b); // a same
equilateral_area = (sqrt(3) / 4) * a * a;
regular_polygon_area = (n * a * a / 4) * (1 / tan(M_PI / n));
point_line_distance = abs(a*x + b*y + c) / sqrt(a*a + b*b);
two_point_distance = sqrt((x2 - x1)*(x2 - x1) + (y2 - y1)*(y2 - y1));
area_triangle_sine = 0.5 * a * b * sin(C);
line_intercept = y1 - (perpendicular_slope * x1);
perpendicular_slope = -1 / line_slope;
sine_rule = a / sin(A) == b / sin(B) == c / sin(C) == 2 * circumradius;
cosine_rule = c * c = a * a + b * b - 2 * a * b * cos(C);
herons_area = sqrt(s * (s - a) * (s - b) * (s - c)); // s = (a+b+c)/2
centroid = ( (x1 + x2 + x3) / 3 , (y1 + y2 + y3) / 3 )
orthocenter = (tanA*x1 + tanB*x2 + tanC*x3) / (tanA + tanB + tanC) , (tanA*y1 + tanB*y2 + tanC*y3) / (tanA + tanB + tanC) )
incenter = ( (a*x1 + b*x2 + c*x3) / (a + b + c) , (a*y1 + b*y2 + c*y3) / (a + b + c) )
circumcenter = intersection of perpendicular bisectors of any two sides
median = ( (x1 + x2) / 2 , (y1 + y2) / 2 )
```

5.2 Geometry [536 lines] - 349abbca

```
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; }
```

```cpp
  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;
}

// CONVEX HULL
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;
}

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
```

```cpp
  // 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);
```

```cpp
  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;
}

// -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
```

```cpp
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 };
}

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;
}

// -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);
}
```

5.3 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 Articulation Bridge [24 lines] - ce2d2a56

```
int timer = 0;
vector <int> G[N];
bool visited[N];
int disc[N], low[N];
vector <pair <int, int>> bridges;

void DFS(int v, int par = -1){
    visited[v] = true;
    disc[v] = low[v] = timer;
    timer++;
    for(auto child : G[v]){
        if(!visited[child]){
            DFS(child, v);
            low[v] = min(low[child], low[v]);
            if(disc[v] < low[child]){
                bridges.push_back({min(v,child), max(v,
                    child)});
            }
        }
        else{
            if(child == par) continue;
            low[v] = min(low[child], low[v]);
        }
    }
}
```

6.2 Articulation Point [25 lines] - ce1ae03f

```
vector <int> G[N];
bool visited[N];
int disc[N], low[N];
bool mark[N]; // articulation point marker
int timer;

void DFS(int v, int par = -1){
    visited[v] = true;
    disc[v] = low[v] = timer;
    timer++;
    int children = 0;
    for(auto child : G[v]){
        if(child == par) continue;
        if(!visited[child]){
            DFS(child, v);
            low[v] = min(low[child], low[v]);
            if(par != -1 && low[child] >= disc[v])
                mark[v] = true;
            children++;
        }
        else{
            low[v] = min(low[v], disc[child]);
        }
    }
    if(par == -1 && children > 1) mark[v] = true;
}
```

6.3 Basic [113 lines] - 38f9507f

```
struct info{
    ll v, w;
    bool const operator < (const info node) const{
        return w > node.w;
    }
};
void dijkstra(int start){
    fill(dist, dist + N, 1e17);
    dist[start] = 0;
    priority_queue <info> q;
    q.push({start, 0});
    while(!q.empty()){
        info cur = q.top();
        q.pop();
        if(cur.w > dist[cur.v]) continue;
        for(auto x : G[cur.v]){
            ll v = x.first;
            ll w = x.second;
            if(dist[v] > cur.w + w){
                dist[v] = cur.w + w;
                q.push({v, dist[v]});
            }
        }
    }
}
// BFS
while(!q.empty()){
    auto[r,c,face,time] = q.front(); q.pop();
    if(r == n-1 && c == n-1 && face == 0){
        cout << time << "\n";
        break;
    }
    int nr = r + dr[face], nc = c + dc[face];
    if(!visited[r][c][(face+1)%4]){
        q.push({r, c, (face+1)%4, time+1});
        visited[r][c][(face+1)%4] = true;
    }
    if(nr < 0 || nr >= n || nc < 0 || nc >= n ||
        s[nr][nc] == '#' || visited[nr][nc][face])
        continue;
    q.push({nr, nc, face, time+1});
    visited[nr][nc][face] = true;
}
// DP on Trees
void dfs(int u, int par = -1){
    for(auto v : G[u]){
```

```
        if(v == par) continue;
        dfs(v, u);  // FIRST solve child's dp
        // THEN use child's dp to update parent u's dp
        // accumulate into dp[u][0] (when a_u = L[u]):
        dp[u][0] += max(
            dp[v][0] + abs(L[u] - L[v]),    // child
                picks L[v]
            dp[v][1] + abs(L[u] - R[v])     // child
                picks R[v]
        );
        // accumulate into dp[u][1] (when a_u = R[u]):
        dp[u][1] += max(
            dp[v][0] + abs(R[u] - L[v]),    // child
                picks L[v]
            dp[v][1] + abs(R[u] - R[v])     // child
                picks R[v]
        );
        // Update dp[u] using best from children
        // assuming a_u is L[u] or R[u]
    }
}
// Floyd Warshall
for(int k = 0; k < n; k++){
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            dis[i][j] = min(dis[i][j], dis[i][k] +
                dis[k][j]);
        }
    }
}

// MST - Kruskal
int Find(int x) {
    if(parent[x] != x) parent[x] = Find(parent[x]);
    return parent[x];
}
void Union(int x, int y) {
    int root_x = Find(x);
    int root_y = Find(y);
    if(root_x != root_y){
        parent[root_x] = root_y;
    }
}
ll Kruskal(vector <tuple<ll,ll,ll>> edges) {
    vector <tuple<ll,ll,ll>> MST;
    sort(edges.begin(), edges.end());
    ll sum = 0;
    for(auto [w, u, v] : edges){
        if(Find(u) != Find(v)){
            MST.push_back({w, u, v});
            Union(u, v);
            sum += w;
        }
        if(MST.size() == n - 1) break;
    }
    if(MST.size() == n - 1) return sum;
    else return -1;
}
// Topological Sort - Kahn's Algorithm
void kahn_topological_sort(){
    queue <int> q;
    for(int i = 0; i < n; i++){
        if(indegree[i] == 0) q.push(i);
    }
    while(!q.empty()){
        int u = q.front();
        q.pop();
        topo_order.push_back(u);
        for(auto v : G[u]){
            indegree[v]--;
            if(indegree[v] == 0) q.push(v);
        }
    }
}
```

6.4 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 componets using edge.
      Tree[comp[u]].push_back(comp[v]);
      Tree[comp[v]].push_back(comp[u]);
      int x = comp[u];
      int y = comp[v];
    }
  }
}
```

6.5 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);
  }
}
```

6.6 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);
}
```

6.7 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);
  }
};
```

6.8 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]);
      }
    }
  }
}
```

6.9 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))
   */
```

6.10 Multisource BFS [57 lines] - 9f24161e

```
int dr[4] = {1, -1, 0, 0};
int dc[4] = {0, 0, 1, -1};
int n, m, ans = 0;
vector<pair<int,int>> fire;
vector<vector<int>> firetime;
vector<vector<bool>> visited;
vector<vector<char>> s;
bool isvalid(int x, int y, int timer){
    if(x < 0 || y < 0 || x >= n || y >= m) return false;
    if(firetime[x][y] <= timer) return false;
    return true;
}
bool isfree(int x, int y, int timer){
    if(!isvalid(x, y, timer)) return false;
    if(x == 0 || y == 0 || x == n - 1 || y == m - 1)
        return true;
    return false;
}
void firebfs(){
    queue <tuple<int,int,int>> q;
    for(auto [x,y] : fire){
        q.push({x, y, 0});
    }
    while(!q.empty()){
        auto[r, c, timer] = q.front();
        timer++; q.pop();
        for(int i = 0; i < 4; i++){
            int rr = r + dr[i];
            int cc = c + dc[i];
            if(isvalid(rr, cc, timer)){
                firetime[rr][cc] = timer;
                q.push({rr, cc, timer});
            }
        }
    }
}
bool escapebfs(int sr, int sc){
    queue <tuple<int,int,int>> q;
    q.push({sr, sc, 0});
    visited[sr][sc] = true;
    while(!q.empty()){
        auto[r, c, timer] = q.front();
        timer++; q.pop();
        for(int i = 0; i < 4; i++){
            int rr = r + dr[i];
            int cc = c + dc[i];
            if(isfree(rr, cc, timer)){
                ans = timer;
                return true;
            }
            if(isvalid(rr, cc, timer) &&
                !visited[rr][cc]){
                visited[rr][cc] = true;
                q.push({rr, cc, timer});
            }
        }
    }
    return false;
}
```

6.11 SCC [31 lines] - 9982799c

```
const int mxn = 100003;
vector<int> adj[mxn], stak;
int tin[mxn], low[mxn], n, color[mxn];// component no 1
    based;
bool onstack[mxn];
int timer = 0, sccCount = 0;
void dfs(int at) {
  stak.push_back(at);
  onstack[at] = true;
  tin[at] = low[at] = ++timer;

  for (int to : adj[at]) {
    if (!tin[to]) {
      dfs(to);
      low[at] = min(low[at], low[to]);
    } else if (onstack[to])
      low[at] = min(low[at], tin[to]);
  }

  if (tin[at] == low[at]) {
    ++sccCount;
    while (1) {
      int node = stak.back();
      stak.pop_back();
      onstack[node] = false;
      color[node] = sccCount;
      if (node == at) break;
    }
  }
}
for (int i = 1; i <= n; i++)
  if (!tin[i]) dfs(i);
```

7 Math

7.1 Basic [132 lines] - 7c2ca56e

```
// a + ar + ar^2 + ar^3 + ... + ar^n-1 = a(r^(n) - 1) /
    (r - 1)
// 1 + a + a^2 + a^3 + a^4 + .... + a^(n-1) = (a^n - 1)
    / (a - 1)
// 1 + a + a^2 + a^3 + a^4 + a^5 = 1 + a^2 + (a^2)^2 +
    a(1 + a^2 + (a^2)^2)
// bigsum(a, 6) = bigsum(a^2, 3) + a * bigsum(a^2, 3)
// 1 + a + a^2 + a^3 + a^4 = 1 + a(1 + a + a^2 + a^3)
// bigsum(a, 5) = 1 + a * bigsum(a, 4)

ll bigsum(ll a, ll n) {
    if(n == 0 || n == 1) return n % mod;
    if(n & 1) return ((a % mod) * bigsum(a % mod, n - 1)
        + 1) % mod;
    ll x = bigsum((a * a) % mod, n / 2);
    return (x + (a * x) % mod) % mod;
}

ll power(ll a, ll b) {
    ll result = 1;
    a = a % mod;
    while (b > 0){
        if (b & 1) result = (result * a) % mod;
        a = (a * a) % mod;
        b >>= 1ll;
    }
    return result;
}

inline ll modInverse(ll a) { return power(a, mod - 2); }

ll formula(ll a, ll n){
    if(a == 1) return n;
    return ((power(a, n) - 1) * (modInverse(a - 1))) %
        mod; // if mod is prime
}

int NotPrime[N], phi[N];
// coprime = No common factors between two numbers
    except 1
// phi(n) = number of numbers less than n that are
    coprime with n
// phi(n) = n * (1 - 1/p1) * (1 - 1/p2) * ... * (1 -
    1/pk)
// phi(n) = n * ((p1 - 1)/p1) * ((p2 - 1)/p2) * ... *
    ((pk - 1)/pk)
// Here p1, p2, ... , pk - everyone devides n because
    they are prime divisors
// In this seive, phi[j] is always divisible by i (The
    Prime Factor)
void seivePhi(){
    for(int i = 1; i < N; i ++) phi[i] = i;
    NotPrime[1] = 1;
    for(int i = 2; i < N; i++){
        if(!NotPrime[i]){
            for(int j = i; j < N; j += i){
                NotPrime[j] = 1;
                phi[j] = (phi[j] / i) * (i - 1);
            }
        }
    }
}
// Properties of Phi :
//     1. if p is a prime number, phi[p] = p - 1;
//     2. if p is a prime number and n is a positive
    integer, phi[p^n] = p^n - p^(n-1);
```

```
//     3. if a and m are coprime, a^phi[m] = 1 mod m
    (Euler's Theorem)
//     4. if a and m are coprime and m is prime, a^(m-1)
    = 1 mod m (Fermat's little theorem)

int phi(int n){
    int ans = n;
    for(int i = 2; 1ll * i * i <= n; i++){
        if(n % i == 0){ // i is prime
            while(n % i == 0){
                n /= i;
            }
        }
        ans -= (ans / i); // n * (1 - 1/p1) -> (n -
            n/p1)
    }
    if(n > 1){
        ans -= (ans / n); // there can be only one prime
            factor > sqrt(n)
    }
    return ans;
}

int spf[N];
vector <int> primes;

void sieve() {
    for(int i = 1; i < N; i++) spf[i] = i;
    for(int i = 2; 1ll * i * i < N; i++) {
        if(spf[i] == i) { // i is prime
            for(int j = i * i; j < N; j += i) {
                if(spf[j] == j) spf[j] = i;
            }
        }
    }
    for(int i = 2; i < N; i++){
        if(spf[i] == i) primes.push_back(i);
    }
}
vector <pair<ll,ll>> factor(ll n) {
    vector <pair<ll,ll>> fact;
    for(auto p : primes){
        if(1ll * p * p > n) break;
        if(n % p == 0){
            ll power = 0;
            while(n % p == 0){
                n /= p;
                power++;
            }
            fact.push_back({p, power});
        }
    }
    if(n > 1) fact.push_back({n, 1});
    return fact;
}

// Modular Arithmetic Functions
static inline ll mulmod(ll a, ll b, ll m) {
    a %= m; b %= m;
    return (ll)((__int128)a * b % m);
}
ll powmod(ll a, long long b, ll m) {
    a %= m;
    if (a < 0) a += m;
    ll res = 1 % m;
    while (b > 0) {
        if (b & 1) res = mulmod(res, a, m);
        a = mulmod(a, a, m);
        b >>= 1;
    }
    return res;
}

ll modMul(ll a, ll b) { return mulmod(a, b, mod); }
ll modAdd(ll a, ll b) { return ( (a%mod) + (b%mod) ) %
    mod; }
ll modSub(ll a, ll b) { return ( (a%mod) - (b%mod) + mod
    ) % mod; }
ll modInverse(ll a)   { return powmod((a%mod+mod)%mod,
    mod-2, mod); }
ll modDiv(ll a, ll b) { return modMul(a,
    modInverse(b)); }

// Exponents reduced modulo (mod-1)
ll p  = (powmod(2, n, mod-1) - 1 + (mod-1)) % (mod-1);
```

7.2 CRT [52 lines] - ff3bd658

```
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
      addEqation()*/
  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 Coprime Subsequence Mobius [31 lines] - dc967df5

```
const int N = 1e5 + 7, mod = 1e9 + 7;
ll cnt[N];  // cnt[x] = frequency of x in the input
ll d[N];    // d[i]   = # of input elements divisible by
    i
ll f[N];    // f[i]   = # of non-empty subseq's all
    divisible by i
ll mob[N];  // Mobius function mu(i)
ll power(ll a, ll b) { }
void coprime_subsequence_mobius(){
    int n; cin >> n;
    for(int i = 0; i < n; i++){
        int x; cin >> x;
        cnt[x]++;
    }
    mob[1] = 1;
    for(int i = 1; i < N; i++){
        for(int j = i + i; j < N; j += i){
            mob[j] -= mob[i];
        }
    }
    for(int i = 1; i < N; i++){
        for(int j = i; j < N; j += i){
            d[i] += cnt[j];
        }
    }
    ll ans = 0;
    for(int i = 1; i < N; i++){
        if(d[i] == 0 || mob[i] == 0) continue;  //
            nothing to add/subtract
        f[i] = (power(2, d[i]) - 1 + mod) % mod; //
            number of non-empty subsequences from d[i]
        ans = ((ans + mob[i] * f[i]) % mod + mod) %
            mod; // inclusion{exclusion using mobius
    }
    cout << ans << "\n";
}
```

7.4 Extented GCD [19 lines] - 45160ec3

```
// Extended GCD: finds gcd(a,b) and x,y such that a*x +
    b*y = gcd(a,b)
int gcdExt(int a, int b, int &x, int &y) {
    if (b == 0) {
        x = 1; y = 0; // base: a*1 + 0*0 = a
        return a;
    }
    int x1, y1;
```

```
    int g = gcdExt(b, a % b, x1, y1); // recurse
    x = y1; // transform back
    y = x1 - (a / b) * y1;
    return g;
}

// Modular inverse: returns x where (a*x) % m = 1, exists only if gcd(a,m) = 1
int modInv(int a, int m) {
    int x, y;
    int g = gcdExt(a, m, x, y); // a*x + m*y = gcd(a,m), if g==1 then x is inverse
    return (x % m + m) % m; // normalize to positive
}
```

7.5 FFT [85 lines] - 5ed04be4

```
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;
  }
}
```

7.6 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;
}
```

7.7 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]);
  }
};
```

7.8 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
```

7.9 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
}
```

7.10 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;
  }
};
```

7.11 Miller-Rabin-Pollard-Rho [46 lines] - d12e2f09

```
namespace rho {
  ll mul(ll a, ll b, ll mod) {
    ll ret = __int128(a) * b - mod * (ll)(1.L / mod * a
      * b);
    return ret + mod * (ret < 0) - mod * (ret >=
      (ll)mod);
  }

  ll bigMod(ll a, ll e, ll mod) {
    ll ret = 1;
    while (e) {
      if (e & 1) ret = mul(ret, a, mod);
      a = mul(a, a, mod);
      e >>= 1;
    }
    return ret;
  }
  bool isPrime(ll n) {
    if (n < 2 or n % 6 % 4 != 1) return (n | 1) == 3;
    ll a[] = {2, 325, 9375, 28178, 450775, 9780504,
      1795265022};
    ll s = __builtin_ctzll(n - 1), d = n >> s;
    for (ll x : a) {
      ll p = bigMod(x % n, d, n), i = s;
      while (p != 1 and p != n - 1 and x % n and i--) p
        = mul(p, p, n);
      if ((p != n - 1) and i != s) return 0;
    }
    return 1;
  }
  ll pollard(ll n) {
    auto f = [&](ll x) { return mul(x, x, n) + 1; };
    ll x = 0, y = 0, t = 0, prod = 2, i = 1, q;
    while (t++ % 40 or gcd(prod, n) == 1) {
      if (x == y) x = ++i, y = f(x);
      if ((q = mul(prod, max(x, y) - min(x, y), n)))
        prod = q;
      x = f(x), y = f(f(y));
    }
    return gcd(prod, n);
```

```
  }
  vector<ll> factors(ll n) {  // return unsorted factors
    if (n == 1) return {};
    if (isPrime(n)) return {n};
    ll x = pollard(n);
    auto l = factors(x), r = factors(n / x);
    l.insert(l.end(), r.begin(), r.end());
    return l;
  }

};
```

7.12 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);
};
```

7.13 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;
}
```

7.14 SOD Upto N [42 lines] - 10b7fcd5

```
int gpf[N];
ll SOD[N];

void seive(){ // O(N log log N)
    gpf[1] = 1;
    for(int i = 2; i < N; i++){
        if(gpf[i] == 0){
            for(int j = i; j <= N; j += i){
                gpf[j] = i; // greatest prime factor
            }
        }
    }
}

void FindSOD(){ // sum of divisors up to N in O(N log log N)
    seive();
    for(int i = 2; i < N; i++){
        ll x = i;
        ll sod = 1;
        while(x > 1){
            ll p = gpf[x];
            ll mul = p;
            while(x % p == 0){
                x /= p;
                mul *= p; // mul = pi^ni (at the end of loop)
            }
            mul -= 1;
            mul /= (p-1); // geometric series: (p^(n+1)-1)/(p-1)
            sod *= mul;
        }
        SOD[i] = sod;
    }
}

ll NOD(ll n) {
    vector<pair<ll, ll>> fact = factor(n);
    ll divisors = 1;
    for(auto x : fact){
        divisors *= (x.second + 1);
    }
    return divisors;
}
```

7.15 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]];
  }
}
```

7.16 nCr [46 lines] - 79b48d4a

```
ll power(ll a, ll b) {
    ll result = 1;
    a = a % mod;
    while (b > 0){
        if (b & 1) result = (result * a) % mod;
        a = (a * a) % mod;
        b >>= 1ll;
    }
    return result;
}

int f[N], invf[N];

void pre() {
    f[0] = 1;
    for (int i = 1; i < N; i++) {
        f[i] = 1LL * i * f[i - 1] % mod;
    }
    invf[N - 1] = power(f[N - 1], mod - 2);
    for (int i = N - 2; i >= 0; i--) {
        invf[i] = 1LL * invf[i + 1] * (i + 1) % mod;
    }
}

int nCr(int n, int r) {
    if (n < r or n < 0) return 0;
    return 1LL * f[n] * invf[r] % mod * invf[n - r] %
        mod;
}

// Catalan Number: 1, 1, 2, 5, 14, 42, 132, 429, 1430,
    4862, ...
// C(n) = (1 / (n + 1)) * (2n choose n) = (2n choose n)
    - (2n choose n + 1)
ll catalan = (nCr(2*n, n) % mod - nCr(2*n, n+1) % mod +
    mod) % mod;
// Hocky Stick Identity:
// sum_{k=r}^{n} (k choose r) = (n+1 choose r+1)
ll hockey_stick = nCr(n + 1, r + 1);
ll vandermonde = nCr(m + n, r);
ll C[N + 2][N + 2];
void pre(){
    C[0][0] = 1;
    for(int n = 1; n <= N; n++){
        C[n][0] = C[n][n] = 1;
        for(int k = 1; k < n; k++){
            C[n][k] = C[n - 1][k - 1] + C[n - 1][k];
        }
    }
}
```

8 Misc

8.1 BigSum [35 lines] - 2a3cf6fd

```
string findSum(string str1, string str2)
{
    if(str1.length() > str2.length()) swap(str1, str2);
    string str = "";
    int n1 = str1.length(), n2 = str2.length();
    reverse(str1.begin(), str1.end());
    reverse(str2.begin(), str2.end());
    int carry = 0;
    for(int i = 0; i < n1; i++)
    {
        int sum = ((str1[i]-'0')+(str2[i]-'0') + carry);
        str.push_back(sum % 10 + '0');
        carry = sum / 10;
    }
    for(int i = n1; i < n2; i++)
    {
        int sum = ((str2[i]-'0') + carry);
        str.push_back(sum % 10 + '0');
        carry = sum / 10;
    }
    if(carry) str.push_back(carry + '0');
    reverse(str.begin(), str.end());
    return str;
}
int compare(string str1, string str2)
{
    int n1 = str1.length(), n2 = str2.length();
    if (n1 < n2) return -1;
    if (n2 < n1) return 1;
    for (int i = 0; i < n1; i++) {
        if (str1[i] < str2[i]) return -1;
        else if (str1[i] > str2[i]) return 1;
    }
    return 0;
} // 0 equal, 1 str1>str2, -1 str1<str2
```

8.2 Bit Hacks [51 lines] - 15b43442

```
#define ckbit(n, k) ((n) & (1LL << (k)))
#define toggle(n, k) ((n) ^= (1LL << (k)))
#define setbit(n, k) ((n) |= (1LL << (k)))
#define unsetbit(n, k) ((n) &= ~(1LL << (k)))
#define lowbit(n) ((n) & -(n))
#define highbit(n) (63 - __builtin_clzll(n))
// ----- Iterate Submasks (non-zero) -----
#define ForSubmask(s, m)  for (ll s = (m); s; s = (s -
    1) & (m))
// ----- Iterate Supersets of m inside [0, 1<<n) -----
#define ForSuperset(s, m, n)  for (ll s = (m); s < (1LL
    << (n)); s = (s + 1) | (m))
// ----- Gosper's Hack | next mask with same popcount
    -----
inline ll NextCombination(ll x){
    ll u = x & -x;
    ll v = x + u;
    return v + (((v ^ x) / u) >> 2);
}
// XOR Properties
a | b = a ^ b + a & b
a ^ (a & b) = (a | b) ^ b
b ^ (a & b) = (a | b) ^ a
(a & b) ^ (a | b) = a ^ b
// Addition
a + b = a | b + a & b
a + b = a ^ b + 2(a & b)
// Subtraction
a - b = (a ^ (a & b)) - ((a | b) ^ a)
a - b = ((a | b) ^ b) - ((a | b) ^ a)
a - b = (a ^ (a & b)) - (b ^ (a & b))
a - b = ((a | b) ^ b) - (b ^ (a & b))
// Precompute XOR from 1 to n
ll prexor(ll n){
    if(n % 4 == 0) return n;
    else if(n % 4 == 1) return 1;
    else if(n % 4 == 2) return n + 1;
    else return 0;
}
// Bit Hacks
# 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

// __int128 for 128 bit integer operations
```

8.3 Bitset C++ [33 lines] - 62a433dd

```
// Declaration & Initialization
bitset<100> bs, bs1(42); // empty, from decimal
bitset<8> bs2("11001010"); // from binary string

// Set/Reset/Flip
bs.set(), bs.set(5), bs[5] = 1; // set all, set bit 5,
    direct assignment
bs.reset(), bs.reset(5); // reset all, reset bit 5
bs.flip(), bs.flip(5); // flip all, flip bit 5

// Query
bs.test(5), bs.all(), bs.any(), bs.none(); // test bit,
    all set, any set, none set
bs.count(), bs.size(); // count set bits, total size

// Bitwise Operations
bitset<8> a("10101010"), b("11001100"), c;
c = a & b, c = a | b, c = a ^ b, c = ~a; // AND, OR,
    XOR, NOT
c = a << 2, c = a >> 2; // left shift, right shift

// Conversion
bs.to_ulong(), bs.to_ullong(), bs.to_string(); // to
    unsigned long, unsigned long long, string

// Find Set Bits
bitset<17> BS; BS[1] = BS[7] = 1;
BS._Find_first(); // first set bit (or size() if none)
BS._Find_next(idx); // next set bit after idx (or size()
    if none)
for(int i = BS._Find_first(); i < BS.size(); i =
    BS._Find_next(i))
    cout << i << endl; // iterate all set bits

// Applications
for(int m = 0; m < (1 << n); m++) bitset<32> bs(m); //
    subset generation
bool isPow2 = (bs.count() == 1); // check power of 2
```

8.4 Custom Hash [19 lines] - 496c4203

```cpp
struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }
    size_t operator()(const pair<int,int>& p) const
        noexcept {
        static const uint64_t FIXED_RANDOM =

                chrono::steady_clock::now().time_since_epoch().count();

        uint64_t key = (uint64_t(uint32_t(p.first)) <<
            32)
                    | uint32_t(p.second);
        return splitmix64(key + FIXED_RANDOM);
    }
    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM =
            chrono::steady_clock::now().time_since_epoch().count();

        return splitmix64(x + FIXED_RANDOM);
    }
};
```

8.5 Important [94 lines] - a982d219

```cpp
// Ternary Search Template
double left = 0, right = 1e8;
// unimodal function: exactly one minima or maxima
while(fabs(right - left) > eps){
    double m1 = left + (right - left) / 3;
    double m2 = right - (right - left) / 3;
    double f1 = 0, f2 = 0;
    for(int i = 0; i < n; i++){
        f1 = max(f1, t[i] + fabs(x[i] - m1));
        f2 = max(f2, t[i] + fabs(x[i] - m2));
    } // ........min........
    if(f1 > f2) left = m1;
    else if(f1 < f2) right = m2;
    else left = m1, right = m2;
}
template <const int32_t MOD>
struct modint {
  int32_t value;
  modint() = default;
  modint(int32_t value_) : value(value_) {}
  inline modint<MOD> operator + (modint<MOD> other)
      const { int32_t c = this->value + other.value;
      return modint<MOD>(c >= MOD ? c - MOD : c); }
  inline modint<MOD> operator - (modint<MOD> other)
      const { int32_t c = this->value - other.value;
      return modint<MOD>(c <    0 ? c + MOD : c); }
  inline modint<MOD> operator * (modint<MOD> other)
      const { int32_t c = (int64_t)this->value *
      other.value % MOD; return modint<MOD>(c < 0 ? c +
      MOD : c); }
  inline modint<MOD> & operator += (modint<MOD> other) {
      this->value += other.value; if (this->value >=
      MOD) this->value -= MOD; return *this; }
  inline modint<MOD> & operator -= (modint<MOD> other) {
      this->value -= other.value; if (this->value <
      0) this->value += MOD; return *this; }
  inline modint<MOD> & operator *= (modint<MOD> other) {
      this->value = (int64_t)this->value * other.value %
      MOD; if (this->value < 0) this->value += MOD;
      return *this; }
  inline modint<MOD> operator - () const { return
      modint<MOD>(this->value ? MOD - this->value : 0);
      }
  modint<MOD> pow(uint64_t k) const { modint<MOD> x =
      *this, y = 1; for (; k; k >>= 1) { if (k & 1) y
      *= x; x *= x; } return y; }
  modint<MOD> inv() const { return pow(MOD - 2); }  //
      MOD must be a prime
  inline modint<MOD> operator /  (modint<MOD> other)
      const { return *this *  other.inv(); }
  inline modint<MOD> operator /= (modint<MOD> other)
      { return *this *= other.inv(); }
  inline bool operator == (modint<MOD> other) const {
      return value == other.value; }
  inline bool operator != (modint<MOD> other) const {
      return value != other.value; }
  inline bool operator < (modint<MOD> other) const {
      return value < other.value; }
  inline bool operator > (modint<MOD> other) const {
      return value > other.value; }
};
template <int32_t MOD> modint<MOD> operator * (int64_t
    value, modint<MOD> n) { return modint<MOD>(value) *
    n; }
template <int32_t MOD> modint<MOD> operator * (int32_t
    value, modint<MOD> n) { return modint<MOD>(value %
    MOD) * n; }
template <int32_t MOD> istream & operator >> (istream &
    in, modint<MOD> &n) { return in >> n.value; }
template <int32_t MOD> ostream & operator << (ostream &
    out, modint<MOD> n) { return out << n.value; }

using mint = modint<mod>;
struct combi{
  int n; vector<mint> facts, finvs, invs;
  combi(int _n): n(_n), facts(_n), finvs(_n), invs(_n){
    facts[0] = finvs[0] = 1;
    invs[1] = 1;
    for (int i = 2; i < n; i++) invs[i] =  invs[mod % i]
        * (-mod / i);
    for(int i = 1; i < n; i++){
      facts[i] = facts[i - 1] * i;
      finvs[i] = finvs[i - 1] * invs[i];
    }
  }
  inline mint fact(int n) { return facts[n]; }
  inline mint finv(int n) { return finvs[n]; }
  inline mint inv(int n) { return invs[n]; }
  inline mint ncr(int n, int k) { return n < k or k < 0
      ? 0 : facts[n] * finvs[k] * finvs[n-k]; }
};
combi C(N);
ll power(ll a, ll b) {
    ll ans = 1;
    bool flag = (a >= mod);
    while(b) {
        if(b & 1) {
            ans *= a;
            if(ans >= mod) {
                ans %= mod;
                flag = true;
            }
        }
        a *= a;
        if(a >= mod) {
            a %= mod;
            flag = true;
        }
        b >>= 1;
    }
    return (ans + (flag * mod)) % mod;
}
set <string> getPerm(string s) {
    int n = s.size();
    vector<int> v(n);
    for(int i = 0; i < n; i++) v[i] = i;
    set <string> st;
    do{
        string ss = s;
        // cout << ss << " ";
        for(int i = 0; i < n; i++){
            ss[i] = s[v[i]];
        }
        st.insert(ss);
    } while (next_permutation(v.begin(), v.end()));
    return st;
}
```

8.6 N-Queen [15 lines] - c51ac310

```cpp
bool place(vector <int> &x, int row, int col) {
    for(int j = 0; j < row; j++){
        if(x[j] == col || abs(x[j] - col) == abs(row -
            j)) return false;
    }
    return true;
}
void NQueens(vector <int> &x, int row, int n) {
    for(int col = 0; col < n; col++){
        if(place(x, row, col)){
            x[row] = col;
            if(row == n - 1) ans.push_back(x);
            else NQueens(x, row + 1, n);
        }
    }
}
```

8.7 Template [33 lines] - 52af6a79

```cpp
// #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 << '}'; }
```

```cpp
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]);
}
```

8.8 debug scripts [49 lines] - 72496740

```bash
#!/bin/bash
# compact debug helper: build/test/debug/stress
# Install: mkdir -p ~/bin && chmod +x ~/bin/* && add to
    PATH

build(){ # build <name> [flags]
  g++ -std=gnu++17 -Wall -O2 -g
      -fsanitize=undefined,address $2 "$1.cpp" -o "$1"
}

test_all(){ # test <name>
  build "$1" || return 1
  for IN in "$1".in*; do
    echo "=== $IN ==="; cat "$IN"
    echo "--- OUTPUT ---"; ./$1 < "$IN"
  done
}

debug(){ # debug <name>
  build "$1" -DSMIE || return 1
  ./$1
}

stress(){ # stress <name>
  build "$1" || return 1
  build "${1}_gen" || return 1
  build "${1}_brute" || return 1
  i=0
  while true; do ((i++)); echo "Test $i"
    ./${1}_gen "$i" > inp
    ./$1 < inp > out1
    ./${1}_brute < inp > out2
    if ! diff -w out1 out2 >/dev/null; then
      echo "FAILED on test $i"; cat inp; echo "Your:";
          cat out1; echo "Exp:"; cat out2
      break
    fi
  done
}

# Quick: build name | test name | debug name | stress
    name

if [[ "${BASH_SOURCE[0]}" == "${0}" ]]; then
  cmd="$1"; shift || true
  case "$cmd" in
    build) build "$@" ;;
    test) test_all "$@" ;;
    debug) debug "$@" ;;
    stress) stress "$@" ;;
    *) echo "Usage: $0 {build|test|debug|stress} name"
        ;;
  esac
fi
```

8.9 pbds inversions [21 lines] - 9cb3c2cf

```cpp
#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 <class T>
using pbds = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;
// greater for descending, less_equal for muiltiset
typedef long long ll;
// Count inversions in array using PBDS - O(n log n)
// Returns number of pairs (i, j) where i < j and arr[i]
    > arr[j]
template <class T>
ll countInversions(vector<T> &arr) {
    ll inv = 0;
    pbds<T> pset;
    for(int i = arr.size() - 1; i >= 0; i--) { //
        traverse from right to left
        inv += pset.order_of_key(arr[i]); // count
            elements < arr[i] in right side
        pset.insert(arr[i]);
    }
    return inv;
}
```

8.10 sublime-build [12 lines] - ea28f58d

```
// Windows:
{
  "cmd": "g++.exe -std=c++17 $file -o
      $file_base_name.exe && $file_base_name.exe <
      inputf.in > outputf.in",
  "shell": true,
  "working_dir": "$file_path"
}
// Linux/MacOS:
{
  "cmd": "g++ -std=c++17 $file -o $file_base_name &&
      ./$file_base_name < inputf.in > outputf.in",
  "shell": true,
  "working_dir": "$file_path"
}
```

9 String

9.1 Double Hasing [50 lines] - f434a7a8

```cpp
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 concate(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);
    }
};
```

9.2 KMP [33 lines] - c11dbb44

```
vector<int> build_lps(string p) {
    vector<int> lps(p.size());
    int j = 0;
    for(int i = 1; i < p.size(); ) {
        if(p[i] == p[j]){
            lps[i] = j + 1;
            ++i, j++;
        }
        else{
            if(j != 0) j = lps[j - 1];
            else lps[i] = 0, ++i;
        }
    }
    return lps;
}
int kmp(string s, string p) {
    vector<int> lps = build_lps(p);
    int psz = p.size(), sz = s.size(), ans = 0;
    int i = 0, j = 0; // i -> s, j -> p;
    while(i < s.size()) {
        if(s[i] == p[j]){
            ++i, j++;
        }
        else{
            if(j != 0) j = lps[j - 1];
            else ++i;
        }
        if(j == p.size()){
            ans++;
        }
    }
    return ans;
}
```

9.3 Manacher [41 lines] - c534b74d

```
struct Manacher {
    vector<int> p[2];  // p[0]: even-length, p[1]:
        odd-length palindromes
    // p[1][i] = (maximum half-length of odd-length
        palindrome centered at i)
    // p[0][i] = (maximum half-length of even-length
        palindrome centered at i)
    // For s = "abbabba",
    // p[1][3] = 3, "abbabba" centered at index 3 has a
        length of 7, and 7/2 = 3.
    // p[0][2] = 2, "abba" centered between index 1 and
        2 has a length of 4, and 4/2 = 2.
    Manacher(string s) {
        int n = s.size();
        p[0].resize(n + 1);
        p[1].resize(n);
        for (int z = 0; z < 2; z++) {
            for (int i = 0, l = 0, r = 0; i < n; i++) {
                int t = r - i + !z;  // calculate how
                    much we can reuse
                if (i < r) p[z][i] = min(t, p[z][l +
                    t]);  // reuse previous results if
                    possible
                // Expand around center i for the
                    current z (even/odd)
                int L = i - p[z][i], R = i + p[z][i] -
                    !z;
                while (L >= 1 && R + 1 < n && s[L - 1]
                    == s[R + 1]) {
                    p[z][i]++; L--; R++;
                }
                if (R > r) l = L, r = R;
            }
        }
    }
    bool is_palindrome(int l, int r) { // O(1)
        int mid = (l + r + 1) / 2;
        int len = r - l + 1;
        // Check if the palindrome from mid can cover
            the substring of length 'len'
        return 2 * p[len % 2][mid] + len % 2 >= len;
    }
    // Returns the total number of palindromic
        substrings in the string
    ll total_palindromic_substrings() { // O(n)
        ll total = 0;
        for (int z = 0; z < 2; z++) {
            for (int v : p[z]) {
                total += v;
            }
        }
        return total;
    }
};
```

9.4 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);
  }
};
```

9.5 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;
  }
}
```

9.6 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]);
  }
};
```

9.7 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);
  }
};
```

9.8 Z-Algorithm [11 lines] - 3923b6ef

```
vector<int> z_function(string s) {
  int n = s.size();
  vector<int> z(n);
  int l = 0, r = 0;
  for (int i = 1; i < n; i++) {
    if (i < r) z[i] = min(r - i, z[i - l]);
    while (i + z[i] < n && s[z[i]] == s[i + z[i]])
      z[i]++;
    if (i + z[i] > r) l = i, r = i + z[i];
  }
  return z;
}
```

10 Random

10.1 Combinatorics

- $\sum_{k=0}^{n} \binom{n-k}{k} = Fib_{n+1}$
- $\binom{n}{k} + \binom{n}{k+1} = \binom{n+1}{k+1}$
- $k\binom{n}{k} = n\binom{n-1}{k-1}$
- 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.
- $S(n,k) = k * S(n-1,k) + S(n-1,k-1), S(0,0) = 1, S(n,0) = S(0,n) = 0$
- $S(n,2) = 2^{n-1} - 1$
- $S(n,k) = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n$
- $S(n,k) * k!$ = 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

- $\sum_{i=1}^{n} = \frac{1}{m+1}[(n+1)^{m+1} - 1 - \sum_{i=1}^{n}((i+1)^{m+1} - i^{m+1} - (m+1)i^m)]$
- $\sum_{i=0}^{n} c^i = \frac{c^{n+1}-1}{c-1}, c \neq 1$
- $\sum_{i=0}^{\infty} c^i = \frac{1}{1-c}, \sum_{i=1}^{\infty} c^i = \frac{c}{1-c}, |c| < 1$
- $\sum_{i=0}^{n-1} ar^i = a\,\frac{1-r^n}{1-r},\ r \neq 1$
- $\sum_{i=1}^{n} ir^{i-1} = \frac{1-(n+1)r^n+nr^{n+1}}{(1-r)^2},\ r \neq 1$
- $\sum_{i=1}^{n} ir^i = \frac{r\left(1-(n+1)r^n+nr^{n+1}\right)}{(1-r)^2},\ r \neq 1$
- $1 + x + x^2 + x^3 + \cdots = \frac{1}{1-x},\ |x| < 1$
- $(1+x)^\alpha = \sum_{i=0}^{\infty} \binom{\alpha}{i} x^i,\ |x| < 1$
- $\frac{1}{1+x^2} = \sum_{i=0}^{\infty} (-1)^i x^{2i},\ |x| < 1$
- $\arctan x = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{2i+1},\ |x| \leq 1,\ x \neq \pm 1$
- $\frac{1}{1-x^2} = \sum_{i=0}^{\infty} x^{2i},\ |x| < 1$
- $\frac{1}{(1-x)^2} = \sum_{i=1}^{\infty} ix^{i-1},\ |x| < 1$
- $\frac{x}{(1-x)^2} = \sum_{i=1}^{\infty} ix^i,\ |x| < 1$
- $\ln(1+x) = \sum_{i=1}^{\infty} (-1)^{i+1} \frac{x^i}{i},\ |x| < 1$
- $\ln\left(\frac{1+x}{1-x}\right) = 2\sum_{i=0}^{\infty} \frac{x^{2i+1}}{2i+1},\ |x| < 1$
- $H_n = \sum_{i=1}^{n} \frac{1}{n}, \sum_{i=1}^{n} iH_i = \frac{n(n+1)}{2} H_n - \frac{n(n-1)}{4}$
- $\sum_{k=0}^{n} \binom{r+k}{k} = \binom{r+n+1}{n}$
- $\sum_{i=r}^{n} \binom{i}{r} = \binom{n+1}{r+1}$
- $\sum_{i=0}^{n} i\binom{n}{i} = n2^{n-1}$
- $\sum_{i=0}^{n} i^2\binom{n}{i} = n(n+1)2^{n-2}$
- $\sum_{i=0}^{n} k^i\binom{n}{i} = (k+1)^n$
- $\sum_{i=1}^{n} F_i = F_{n+2} - 1$
- $\sum_{i=1}^{n} F_{2i} = F_n F_{n+1}$
- $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$
- $\sum_{i=1}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2$

10.2.6 3D Geometry

- Distance between points $A(x_1, y_1, z_1)$ and $B(x_2, y_2, z_2)$: $|AB| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$
- Area of triangle with edge vectors $\vec{u}, \vec{v}$: $\frac{1}{2}\|\vec{u} \times \vec{v}\|$
- Volume of parallelepiped with generators $\vec{u}, \vec{v}, \vec{w}$: $V = |\vec{u} \cdot (\vec{v} \times \vec{w})|$
- Distance from point $P(x_0, y_0, z_0)$ to plane $ax+by+cz+d = 0$: $\frac{|ax_0+by_0+cz_0+d|}{\sqrt{a^2+b^2+c^2}}$
- Plane through three points A, B, C: $\vec{n} = \overrightarrow{AB} \times \overrightarrow{AC}$, equation $\vec{n} \cdot (\vec{r} - \vec{r}_A) = 0$
- Plane from point $P_0(x_0, y_0, z_0)$ and normal $\vec{n} = (a, b, c)$: $a(x - x_0) + b(y - y_0) + c(z - z_0) = 0$
- Distance between parallel planes $ax + by + cz + d_1 = 0$ and $ax + by + cz + d_2 = 0$: $\frac{|d_1-d_2|}{\sqrt{a^2+b^2+c^2}}$
- Distance from point P to line through A with direction $\vec{d}$: $\frac{\|\overrightarrow{AP} \times \vec{d}\|}{\|\vec{d}\|}$
- Shortest distance between skew lines $L_1 : \vec{r} = \vec{a}_1 + s\vec{d}_1$ and $L_2 : \vec{r} = \vec{a}_2 + t\vec{d}_2$: $\frac{|(\vec{a}_2-\vec{a}_1)\cdot(\vec{d}_1\times\vec{d}_2)|}{\|\vec{d}_1\times\vec{d}_2\|}$
- Line through two points $A(x_1, y_1, z_1)$ and $B(x_2, y_2, z_2)$: $\frac{x-x_1}{x_2-x_1} = \frac{y-y_1}{y_2-y_1} = \frac{z-z_1}{z_2-z_1}$
- Plane through $A(x_1, y_1, z_1), B(x_2, y_2, z_2), C(x_3, y_3, z_3)$: $\begin{vmatrix} x - x_1 & y - y_1 & z - z_1 \\ x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_1 & y_3 - y_1 & z_3 - z_1 \end{vmatrix} = 0$
- Coplanarity of A, B, C, D: $\begin{vmatrix} x_A & y_A & z_A & 1 \\ x_B & y_B & z_B & 1 \\ x_C & y_C & z_C & 1 \\ x_D & y_D & z_D & 1 \end{vmatrix} = 0$
- Plane intercept form cutting axes at a, b, c: $\frac{x}{a} + \frac{y}{b} + \frac{z}{c} = 1$
- Angle between two planes with normals $\vec{n}_1, \vec{n}_2$: $\cos\theta = \frac{|\vec{n}_1\cdot\vec{n}_2|}{\|\vec{n}_1\|\|\vec{n}_2\|}$
- Line-plane intersection for line $\vec{p} = \vec{p}_0 + t\vec{d}$ and plane $ax + by + cz + d = 0$: $t = -\frac{ax_0+by_0+cz_0+d}{ad_x+bd_y+cd_z}$ when denominator non-zero
- Parametric line from point $P_0(x_0, y_0, z_0)$ with direction (a, b, c): $x = x_0 + at, y = y_0 + bt, z = z_0 + ct$
- Line with direction ratios (l, m, n) through P_0: symmetric form $\frac{x-x_0}{l} = \frac{y-y_0}{m} = \frac{z-z_0}{n}$; direction cosines $(\cos\alpha, \cos\beta, \cos\gamma)$ satisfy $\cos^2\alpha + \cos^2\beta + \cos^2\gamma = 1$
- Angle between line with direction ratios (l, m, n) and plane $ax + by + cz + d = 0$: $\sin\theta = \frac{|al+bm+cn|}{\sqrt{l^2+m^2+n^2}\sqrt{a^2+b^2+c^2}}$
- Midpoint of segment joining $A(x_1, y_1, z_1)$ and $B(x_2, y_2, z_2)$: $M\left(\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2}, \frac{z_1+z_2}{2}\right)$
- Centroid of triangle with vertices A, B, C: $G\left(\frac{x_A+x_B+x_C}{3}, \frac{y_A+y_B+y_C}{3}, \frac{z_A+z_B+z_C}{3}\right)$
- Centroid of tetrahedron with vertices A, B, C, D: $\left(\frac{x_A+x_B+x_C+x_D}{4}, \frac{y_A+y_B+y_C+y_D}{4}, \frac{z_A+z_B+z_C+z_D}{4}\right)$
- Sphere of radius r: surface $S = 4\pi r^2$, volume $V = \frac{4}{3}\pi r^3$
- Right circular cone (radius r, height h): $V = \frac{1}{3}\pi r^2 h$, lateral area $\pi r\sqrt{r^2 + h^2}$