Team Note Book
University of Asia Pacific
December 20, 2025
Team : UAP_Akatsuki

Contents:

1 Data Structure

1.1 BIT(Range Sum Query)

```
int main() {
  dynamicQueries();
  int n; long long k;
  cin >> n >> k;
  vector<int> a(n + 1);
  for(int i = 1; i <= n; i++) cin >> a[i];
  //m: 1=sum==k,2=sum>=k,3=sum<=k,4=sum>k,5=sum<k
  cout << countSubarraySum(a, k, 1) << "\n";
}
```

```
// ---------------- BIT STRUCTURE ----------------
struct BIT {
  int n;
  vector<long long> tree;
  BIT(int _n) { n = _n; tree.assign(n + 5, 0); }
  void upd(int i, long long val) { // point update
    while (i <= n) { tree[i] += val; i += (i & -i); }
  }
  long long qry(int i) { // prefix sum
    long long ans = 0;
    while (i > 0) { ans += tree[i]; i -= (i & -i); }
    return ans;
  }
  long long range_qry(int l, int r) { // sum[l..r]
    if (l > r) return 0;
    return qry(r) - qry(l - 1);
  }
};

// --------- FIXED ARRAY SUBARRAY SUM ------------
long long countSubarraySum(vector<int> &a, long long
k, int mode = 1) {
  int n = a.size() - 1; // 1-indexed
  vector<long long> pref(n + 1);
  vector<long long> all;
  pref[0] = 0; all.push_back(0);

  for (int i = 1; i <= n; i++) {
    pref[i] = pref[i - 1] + a[i];
    all.push_back(pref[i]);
    all.push_back(pref[i] - k);
  }

  // Coordinate compression
  sort(all.begin(), all.end());
  all.erase(unique(all.begin(), all.end()),
all.end());
  auto getIndex = [&](long long x){ return
lower_bound(all.begin(), all.end(), x) - all.begin()
+ 1; };

  BIT bit(all.size());
  long long ans = 0;
  bit.upd(getIndex(0), 1); // empty prefix

  for (int i = 1; i <= n; i++) {
    long long target = pref[i] - k;
    if (mode == 1) {
      ans += bit.range_qry(getIndex(target),
getIndex(target));
    } else if (mode == 2) {
      int id = upper_bound(all.begin(), all.end(),
target) - all.begin();
      ans += bit.range_qry(1, id);
    } else if (mode == 3) {
      int id = lower_bound(all.begin(), all.end(),
target) - all.begin() + 1;
      ans += bit.range_qry(id, all.size());
    } else if (mode == 4) {
      int id = lower_bound(all.begin(), all.end(),
target) - all.begin();
      ans += bit.range_qry(1, id);
    } else if (mode == 5) {
      int id = upper_bound(all.begin(), all.end(),
target) - all.begin() + 1;
      ans += bit.range_qry(id, all.size());
    }
    bit.upd(getIndex(pref[i]), 1);
  }
  return ans;
}
// ---------DYNAMIC RANGE SUM QUERIES -----------
void dynamicQueries() {
  int n, Q;
  long long k;
  cin >> n >> Q >> k;
  vector<long long> a(n + 1);
  BIT bit(n);
  for (int i = 1; i <= n; i++) {
    cin >> a[i];
    bit.upd(i, a[i]);
  }
  while(Q--) {
    int type; cin >> type;
    if(type == 1) { // range query
      int l, r; cin >> l >> r;
      long long sum = bit.range_qry(l, r);
      if(sum == k)
        cout << "YES\n";
      else cout << "NO\n";
    } else if(type == 2) { // point update
      int p; long long c; cin >> p >> c;
      long long diff = c - a[p];
      bit.upd(p, diff);
      a[p] = c;}}}
```

1.2 Disjoint Interval Set

```
// maintain consecutive same values (char or number)
// Type 1: max segment length containing index i, Type
         2: remove value and split segment
  string s; cin >> s;
  set<pair<int,int>> sp;
  int st = 0;
  for(int i = 1; i <= s.size(); i++){
    if(i == s.size() || s[i] != s[st]){
      sp.insert({st, i - 1});
      st = i;}}
  int q; cin >> q;
  while(q--){
    int t,n; cin >> t >> n;
    if(t == 1){
      auto it = sp.lower_bound({n + 1, - 1});
      if(it != sp.begin()){
        --it;
        int b = it -> first; int e = it -> second;
        if(b <= n && n <= e){
          cout << e - b + 1 << endl;}
        else cout << 0 << endl;
      }else cout << 0 << endl;
    }else{
      if(s[n] == '#') continue;
      auto it = sp.lower_bound({n + 1, - 1});
      if(it != sp.begin()){
        --it;
        int b = it -> first; int e = it -> second;
        if(b <= n && n <= e){
          sp.erase(it);
          if(b <= n - 1) sp.insert({b, n - 1});
          if (n + 1 <= e) sp.insert({n + 1, e});
          s[n] = '#';
        }}}}
```

1.3 Bit Binary Search

```
// --- Bit Binary Search in o(log(n)) ---
const int M = 20
const int N = 1 << M
int lower_bound(int val){
    int ans = 0, sum = 0;
    for(int i = M - 1; i >= 0; i--){
        int x = ans + (1 << i);
        if(sum + bit[x] < val)
        ans = x, sum += bit[x];}
    return ans + 1;}
```

1.4 DSU

```
/*use when you need to group, merge, check connectivity or (e.g. same group, MST, or cycle check)*/
#include <bits/stdc++.h>
using namespace std;
const int N = 3e5 + 9;
struct DSU {
    vector<int> par, rnk, sz;
    int c;
    DSU(int n) : par(n + 1), rnk(n + 1, 0), sz(n + 1, 1), c(n) {
        for (int i = 1; i <= n; ++i) par[i] = i;
    }
    int find(int i) { return (par[i] == i ? i : (par[i] = find(par[i]))); }
    bool same(int i, int j) { return find(i) == find(j); }
    int get_size(int i) { return sz[find(i)]; }
    int count() { return c; }
    int merge(int i, int j) {
        if ((i = find(i)) == (j = find(j))) return -1;
        --c;
        if (rnk[i] > rnk[j]) swap(i, j);
        par[i] = j;
        sz[j] += sz[i];
        if (rnk[i] == rnk[j]) rnk[j]++;
        return j;}
};
int32_t main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n, m; cin >> n >> m;
    DSU dsu(n);
    for (int i = 0; i < m; i++) {
        int k; cin >> k;
        if (k == 0) continue;
        int first; cin >> first;
        for (int j = 1; j < k; j++) {
            int x; cin >> x;
            dsu.merge(first, x);}}
 for (int i = 1; i <= n; i++) cout << dsu.get_size(i) << " "; cout << "\n";
}
```

1.5 GP Hash Table

```
#include <ext/pb_ds/assoc_container.hpp> //two-sum,diff-pairs (a[i]-a[j]=k)
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
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()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM = chrono::
        steady_clock::now().time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }};
gp_hash_table<long long,int,custom_hash> mp;
int main() {
    int n; long long target;
    cin >> n >> target;
    vector<long long> a(n+1);
    for(int i=1;i<=n;i++) cin >> a[i];
    mp.clear();
    for(int i=1;i<=n;i++){
        if(mp[target - a[i]]){
            cout << mp[target - a[i]] << " " << i << "\n";
            return 0;}
        mp[a[i]] = i;}cout << "IMPOSSIBLE\n";}
```

1.6 MOs Algorithm

```
// Fixed array: subarray sum, distinct count, freq-square sum
#include <bits/stdc++.h>
using namespace std;
const int N = 200005;
const int B = 440;
struct Query {
  int l, r, id, type;
  bool operator<(const Query &other) const {
    if(l / B != other.l / B) return l / B < other.l / B;
    return ((l / B) & 1) ? r > other.r : r < other.r;
  }};
int a[N];
int cnt[N];
long long ans_sum[N], ans_dist[N], ans_sq[N];
long long sum_result = 0, distinct_result = 0, sq_result = 0;
// Add element a[i] to current segment
inline void add(int i) {
  sum_result += a[i];
  if(cnt[a[i]] == 0) distinct_result++;
  sq_result -= 1LL * cnt[a[i]] * cnt[a[i]] * a[i];
  cnt[a[i]]++;
  sq_result += 1LL * cnt[a[i]] * cnt[a[i]] * a[i];
}
// Remove element a[i] from current segment
inline void remove(int i) {
  sum_result -= a[i];
  sq_result -= 1LL * cnt[a[i]] * cnt[a[i]] * a[i];
  cnt[a[i]]--;
  sq_result += 1LL * cnt[a[i]] * cnt[a[i]] * a[i];
  if(cnt[a[i]] == 0) distinct_result--;}
int main() {
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
  int n, q;
  cin >> n >> q;
  vector<int> vals;
  for(int i = 1; i <= n; i++) {
    cin >> a[i];
    vals.push_back(a[i]);}
  sort(vals.begin(), vals.end());
  vals.erase(unique(vals.begin(), vals.end()), vals.end());
  for(int i = 1; i <= n; i++)
    a[i] = lower_bound(vals.begin(), vals.end(), a[i]) - vals.begin() + 1;

  vector<Query> Q(q);
  for(int i = 0; i < q; i++) {
    cin >> Q[i].l >> Q[i].r;
    Q[i].type = 1; // type: 1=sum, 2=distinct, 3=sum-of-squares
    Q[i].id = i;
  }
  sort(Q.begin(), Q.end());
  int l = 1, r = 0;
  for(auto &qu : Q) {
    int L = qu.l, R = qu.r;
    while(r < R) add(++r);
    while(r > R) remove(r--);
    while(l < L) remove(l++);
    while(l > L) add(--l);
    if(qu.type == 1) ans_sum[qu.id] = sum_result;
    if(qu.type == 2) ans_dist[qu.id] = distinct_result;
    if(qu.type == 3) ans_sq[qu.id] = sq_result;
  }
  for(int i = 0; i < q; i++) {
    if(Q[i].type == 1) cout << ans_sum[i] << "\n";
    if(Q[i].type == 2) cout << ans_dist[i] << "\n";
    if(Q[i].type == 3) cout << ans_sq[i] << "\n";}
}
```

1.7 Merge Sort Tree

```
// Mergesort Tree is a segment tree that stores the sorted subarray
#define vi vector<int> #define pb push_back const int N = 2e5 + 5;
vi st[4*N];
int n, s[N];
void build(int p, int l, int r) {
    if (l == r) { st[p].pb(s[l]); return; }
    build(2*p, l, (l+r)/2);
    build(2*p+1, (l+r)/2+1, r);
    st[p].resize(r-l+1);
    merge(st[2*p].begin(), st[2*p].end(),
          st[2*p+1].begin(), st[2*p+1].end(),
          st[p].begin());}
int query(int p, int l, int r, int i, int j, int a, int b) {
    if (j < l or i > r) return 0;
    if (i <= l and j >= r)
        return upper_bound(st[p].begin(), st[p].end(), b)
             - lower_bound(st[p].begin(), st[p].end(), a);
    return query(2*p, l, (l+r)/2, i, j, a, b)
         + query(2*p+1, (l+r)/2+1, r, i, j, a, b);}
```

1.8 Monotonous Queue

```
// used for Sliding Window Maximum/Minimum in O(n) time (fixed-size window).
const int N = 3e5 + 9;
struct monotonous_queue {
    int a[N + 10], b[N + 10], l = 0, r = -1;
    bool isMax;
    void setType(bool flag) { isMax = flag; }
    void push(int val) {
        int cnt = 0;
        if(isMax) {
            while(l <= r && a[r] <= val) { // max: remove smaller
                cnt += b[r] + 1;
                r--;}
        }
        else {
            while(l <= r && a[r] >= val) { // min: remove larger
                cnt += b[r] + 1;
                r--;}
        }
        a[++r] = val; b[r] = cnt;}
    int top() { return a[l]; }
    void pop() {
        if(l > r) return;
        if(b[l] > 0) { b[l]--; return; }
        l++;
    }
};
int main() {
    int n, k; cin >> n >> k;
    vector<int> arr(n); for(int i = 0; i < n; i++) cin >> arr[i];
    monotonous_queue mq;
    mq.setType(false); // true = max, false = min
    for(int i = 0; i < n; i++) {
        mq.push(arr[i]);
        if(i >= k - 1) {
            cout << mq.top() << " ";
            mq.pop();
        }
    }
}
```

```
Input
8 3
1 3 -1 -3 5 3 6 7
Output
3 3 5 5 6 7
//max each sunarray
```

1.9 Ordered Set

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
using namespace std;
template <typename T> using o_set =
tree<T, null_type, less<T>, rb_tree_tag, tree_order_statistics_node_update>;
template <typename T, typename R> using o_map =
tree<T, R, less<T>, rb_tree_tag, tree_order_statistics_node_update>;
int main() {
    int i, j, k, n, m;
    o_set<int> se;
    se.insert(1);
    se.insert(2);
    cout << *se.find_by_order(0) << endl; // k-th smallest element
    cout << se.order_of_key(2) << endl; // number of elements less than 2
    o_map<int, int> mp;
    mp.insert({1, 10});
    mp.insert({2, 20});
    cout << mp.find_by_order(0)->second << endl; // value of k-th smallest
    cout << mp.order_of_key(2) << endl; // number of keys less than 2
    return 0;
}
```

1.10 Segment Tree Lazy

```
const int N = 5e5 + 9;
long long a[N];
struct ST {
    #define lc (n<<1)
    #define rc ((n<<1)+1)
    long long t[4*N];
    long long lazyAdd[4*N];
    long long lazySet[4*N];
    bool hasSet[4*N];
    ST() {
        memset(t,0,sizeof t);
        memset(lazyAdd,0,sizeof lazyAdd);
        memset(lazySet,0,sizeof lazySet);
        memset(hasSet,0,sizeof hasSet);}
    // ================= Combine Function =================
    inline long long combine(long long a, long long b){
        return a + b; // SUM; MIN : return min(a,b); MAX : return max(a,b);}
    // ================= Neutral element =================
    inline long long neutral(){
        return 0; // SUM; MIN: return LLONG_MAX; MAX: return LLONG_MIN;}
    // ================= Push Lazy Updates =================
    void push(int n,int b,int e){
        if(hasSet[n]){
            t[n] = lazySet[n]*(e-b+1); // SUM
            // For MIN/MAX: t[n] = lazySet[n];
            if(b!=e){
                lazySet[lc] = lazySet[rc] = lazySet[n];
                hasSet[lc] = hasSet[rc] = true;
                lazyAdd[lc] = lazyAdd[rc] = 0;
            }
            hasSet[n] = false; lazySet[n] = 0;}
        if(lazyAdd[n]!=0){
            t[n] += lazyAdd[n]*(e-b+1); // SUM
            // For MIN/MAX: t[n] += lazyAdd[n];
            if(b!=e){
                if(hasSet[lc]) lazySet[lc] += lazyAdd[n];
                else lazyAdd[lc] += lazyAdd[n];

                if(hasSet[rc]) lazySet[rc] += lazyAdd[n];
                else lazyAdd[rc] += lazyAdd[n];
            }
            lazyAdd[n] = 0;}
    }
    void pull(int n){ t[n] = combine(t[lc],t[rc]); }
    // ================= Build =================
    void build(int n,int b,int e){
        if(b==e){ t[n] = a[b]; return; }
        int mid=(b+e)/2;
        build(lc,b,mid);
        build(rc,mid+1,e);
        pull(n);}
     // ================= Range Add =================
    void rangeAdd(int n,int b,int e,int i,int j,long long v){
        push(n,b,e);
        if(j<b||e<i) return;
        if(i<=b&&e<=j){ lazyAdd[n]+=v; push(n,b,e); return; }
        int mid=(b+e)/2;
        rangeAdd(lc,b,mid,i,j,v);
        rangeAdd(rc,mid+1,e,i,j,v);
        pull(n);}
     // ================= Range Set =================
    void rangeSet(int n,int b,int e,int i,int j,long long v){
        push(n,b,e);
        if(j<b||e<i) return;
        if(i<=b&&e<=j){ hasSet[n]=true; lazySet[n]=v; lazyAdd[n]=0; push(n,b,e);
                        return; }
        int mid=(b+e)/2;
        rangeSet(lc,b,mid,i,j,v);
        rangeSet(rc,mid+1,e,i,j,v);
        pull(n);}
     // ================= Point Update =================
    void pointUpdate(int n,int b,int e,int idx,long long val){
        push(n,b,e);
        if(b==e){ t[n] = val; return; }
        int mid=(b+e)/2;
        if(idx<=mid) pointUpdate(lc,b,mid,idx,val);
        else pointUpdate(rc,mid+1,e,idx,val);
        pull(n);}
     // ================= Range Query =================
    long long query(int n,int b,int e,int i,int j){
        push(n,b,e);
        if(j<b||e<i) return neutral();
        if(i<=b&&e<=j) return t[n];
        int mid=(b+e)/2;
        return combine(query(lc,b,mid,i,j), query(rc,mid+1,e,i,j));}
};
int32_t main(){
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n=5;
    for(int i=1;i<=n;i++) a[i] = i; // 1 2 3 4 5
    ST seg;
    seg.build(1,1,n);
    seg.rangeAdd(1,1,n,2,4,10); // [2,4] += 10
    seg.rangeSet(1,1,n,3,5,5); // [3,5] = 5
    seg.pointUpdate(1,1,n,1,100); // idx=1 → 100
    cout << "Sum [1,5] = " << seg.query(1,1,n,1,5) << '\n';
    return 0;
}
```

1.11 Segment Tree Marge Function

```
 // SUM
 int t[4 * N];
 int marge(int x, int y) {
 return x + y;}
 // MIN
 int t[4 * N];
 int marge(int x, int y) {
 return min(x , y);}
 // NUMBER OF MINIMUMS
 struct node {
  int mn, count;};
node t[4 * N];
node merge(node l, node r) {
    node ans = {inf, 0};
    ans.mn = min(l.mn, r.mn);
    if (l.mn == ans.mn) {
        ans.count += l.count;
    }
    if (r.mn == ans.mn) {
        ans.count += r.count;
    }
    return ans;}
// NUMBER OF MAXIMUM
struct node {
    int mx, count;};
node t[4 * N];
node merge(node l, node r) {
    node ans = {inf, 0};
    ans.mx = max(l.mx, r.mx);
    if (l.mx == ans.mx) {
        ans.count += l.count;
    }
    if (r.mx == ans.mx) {
        ans.count += r.count;
    }
    return ans;}
// SEGMENT WITH MAXIMUM SUM
struct node {
    int mx, sum, pref, suf;};
node t[4 * N];
node marge(node a, node b) {
    int curr_max = max({a.mx, b.mx, a.suf + b.pref});
    int curr_pref = max(a.pref, a.sum + b.pref);
    int curr_suf = max(b.suf, b.sum + a.suf);
    int curr_sum = a.sum + b.sum;
    return {curr_max, curr_sum, curr_pref, curr_suf};}
```

1.12 Segment Tree

```
const int N = 3e5 + 9;
int a[N];
struct ST {
    int t[4 * N];
    static const int inf = 1e9;
    ST() {
        memset(t, 0, sizeof t);}
    int marge(int x, int y) {
        return x + y;}
    void build(int n, int b, int e) {
        if (b == e) {
            t[n] = a[b]; // Update
            return;}
        int mid = (b + e) >> 1;
        int l = n << 1, r = l | 1;
        build(l, b, mid);
        build(r, mid + 1, e);
        t[n] = marge(t[l], t[r]); // change this
    }
    void upd(int n, int b, int e, int i, int x) {
        if (b > i || e < i) return;
        if (b == e && b == i) {
            t[n] = x; // update
            return;
        }
        int mid = (b + e) >> 1;
        int l = n << 1, r = l | 1;
        upd(l, b, mid, i, x);
        upd(r, mid + 1, e, i, x);
        t[n] = marge(t[l], t[r]); // change this
    }
};
```

```
int query(int n, int b, int e, int i, int j) {
    if (b > j || e < i) return -inf; // return appropriate value
    if (b >= i && e <= j) return t[n];
    int mid = (b + e) >> 1;
    int l = n << 1, r = l | 1;
    return marge(query(l, b, mid, i, j), query(r, mid + 1, e, i, j));}
} t;
int32_t main() {
    int n = 5;
    for (int i = 1; i <= n; i++) {a[i] = i;}
    t.build(1, 1, n); // building the segment tree
    t.upd(1, 1, n, 2, 10); // assigning 10 to index 2 (a[2] := 10)
    cout << t.query(1, 1, n, 1, 5) << '\n'; // range max query on [1,5]
    return 0;
}
```

1.13 Sparse Table

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 9;
int t[N][18];
int a[N];
void build(int n) {
    for (int i = 1; i <= n; ++i) t[i][0] = a[i];
    for (int k = 1; k < 18; ++k) {
        for (int i = 1; i + (1 << k) - 1 <= n; ++i) {
            t[i][k] = min(t[i][k - 1], t[i + (1 << (k - 1))][k - 1]);
            // ---------- Other possible operations ----------
     // Range Maximum: t[i][k] = max(t[i][k - 1], t[i + (1 << (k-1))][k-1]);
     // Range GCD:     t[i][k] = gcd(t[i][k - 1], t[i + (1 << (k-1))][k-1]);
     // Bitwise AND:   t[i][k] = t[i][k-1] & t[i + (1 << (k-1))][k-1];
     // Bitwise OR:    t[i][k] = t[i][k-1] | t[i + (1 << (k-1))][k-1];
        }
    }
}

int query(int l, int r) {
    int k = 31 - __builtin_clz(r - l + 1);
    return min(t[l][k], t[r - (1 << k) + 1][k]);
    // ---------- Other operations ----------
    // Range Maximum: return max(t[l][k], t[r - (1 << k) + 1][k]);
    // Range GCD:     return gcd(t[l][k], t[r - (1 << k) + 1][k]);
}
int32_t main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    int n; cin >> n;
    for (int i = 1; i <= n; i++) cin >> a[i]; // 1-based indexing
    build(n);
    int q; cin >> q;
    while (q--) {
        int l, r; cin >> l >> r;     // 0-based indexing input
        l++; r++;          // Convert to 1-based indexing
        cout << query(l, r) << '\n';
    }
    return 0;
}
```

1.14 Trie

```
#include<bits/stdc++.h>
using namespace std;
const int N = 3e5 + 9;
struct Trie {
    static const int B = 31; // Number of bits for integers
    struct node {
        node* nxt[2];
        int sz;
        node() {nxt[0] = nxt[1] = NULL; sz = 0;}
    } *root;
    Trie() {root = new node();}
    void insert(int val) {
        node* cur = root;
        cur->sz++;
        for (int i = B - 1; i >= 0; i--) {
            int b = val >> i & 1;
            if (cur->nxt[b] == NULL) cur->nxt[b] = new node();
            cur = cur->nxt[b]; cur->sz++;}
    }
    int query(int x, int k) {
        node* cur = root;
        int ans = 0;
        for (int i = B - 1; i >= 0; i--) {
            if (cur == NULL) break;
            int b1 = x >> i & 1;
            int b2 = k >> i & 1;
            if (b2 == 1) {
                if (cur->nxt[b1]) ans += cur->nxt[b1]->sz;
                cur = cur->nxt[!b1];
            } else cur = cur->nxt[b1];
        }return ans;}
    int get_max(int x) {
        node* cur = root;
        int ans = 0;
        for (int i = B - 1; i >= 0; i--) {
            int k = x >> i & 1;
            if (cur->nxt[!k]) {
                cur = cur->nxt[!k];
                ans <<= 1;
                ans++;
            } else {
                cur = cur->nxt[k];
                ans <<= 1;}
        }return ans;}
    int get_min(int x) {
        node* cur = root;
        int ans = 0;
        for (int i = B - 1; i >= 0; i--) {
            int k = x >> i & 1;
            if (cur->nxt[k]) {
                cur = cur->nxt[k];
                ans <<= 1;
            } else {
                cur = cur->nxt[!k];
                ans <<= 1;
                ans++;}
        }return ans;
    }
    void del(node* cur) {
        for (int i = 0; i < 2; i++)  if (cur->nxt[i]) del(cur->nxt[i]);
        delete(cur);}
} t;
int32_t main() {
    int n, k; cin >> n >> k;
    int cur = 0;
    long long ans = 1LL * n * (n + 1) / 2; // Total number of subarrays
    t.insert(cur);
    for (int i = 0; i < n; i++) {
        int x; cin >> x;
        cur ^= x;           // Prefix XOR
        ans -= t.query(cur, k); // Subarrays with XOR >= k
        t.insert(cur);
    }
    cout << ans << '\n';}
```

2 Dynamic Programming

2.1 Array Description

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 9; const int MOD = 1e9 + 7;
int n, m;
int a[N];
int dp[N][105];
int f(int i, int last) {
    if (i == n + 1) return 1;
    if (dp[i][last] != -1) return dp[i][last];
    int ans = 0, l, r;
    if (a[i] > 0) l = r = a[i];
    else {
        if (i == 1) {
            l = 1; r = m;}
        else{
            l = max(1, last - 1);
            r = min(m, last + 1);}
    }
    for (int cur = l; cur <= r; cur++) {
        if (i == 1 || abs(last - cur) <= 1) {
            ans += f(i + 1, cur);
            if (ans >= MOD) ans -= MOD;
        }}
    return dp[i][last] = ans;
}
int32_t main() {
    cin >> n >> m;
    for (int i = 1; i <= n; i++) cin >> a[i];
    memset(dp, -1, sizeof(dp));
    cout << f(1, 0) << "\n";
} /*You know that an array has n integers between 1 and m,
  and the absolute difference between two adjacent values is at most 1.
  Given a description of the array where some values may be unknown,
  your task is to count the number of arrays that match the
  description.*/
```

2.2 Book Shop

```
 #include <bits/stdc++.h>
 using namespace std;
 int dp[N][E];
 int v[N], p[N];
 int32_t main() {
     int n, m; cin >> n >> m;
     for (int i = 1; i <= n; i++) cin >> v[i];
     for (int i = 1; i <= n; i++) cin >> p[i];
     dp[0][0] = 0;
     for (int i = 1; i <= n; i++) {
         for (int j = 1; j <= m; j++) {
             dp[i][j] = dp[i - 1][j];
             if (j - v[i] >= 0){
                   dp[i][j] = max(dp[i][j], dp[i - 1][j - v[i]] + p[i]);
                 }
         }
     }
     cout << dp[n][m] << "\n";
 }
 /*You are in a book shop which sells n different books.
 You know the price and number of pages of each book.
 You have decided that the total price of your purchases
 will be at most x. What is the maximum number of pages you can
 buy?
 You can buy each book at most once.*/
```

2.3 Coin Combinations

```
#include <bits/stdc++.h>
using namespace std;
const int N = 105, X = 1e6 + 9, mod = 1e9 + 7;
int n, dp[X], v[N];
int coin_combination(int c) {
    if (c == 0) return 1;
    int &ans = dp[c];
    if (ans != -1) return ans;
    ans = 0;
    for (int i = 1; i <= n; i++) {
        if (c >= v[i]) {
            ans += coin_combination(c - v[i]);
            ans %= mod;
        }
    }
    return ans;
}
int32_t main() {
    cin >> n;
    int m; cin >> m;
    for (int i = 1; i <= n; i++) cin >> v[i];
    memset(dp, -1, sizeof(dp));
    cout << coin_combination(m) << "\n";
    return 0;
}
/*number of distinct ways you can produce a money sum x using
the available coins.
if the coins are {2,3,5} and the desired sum is 9, there are 8
*/
```

ways:
2+2+5
2+5+2
5+2+2
3+3+3
2+2+2+3
2+2+3+2
2+3+2+2
3+2+2+2

2.4 Coins

```
#include <bits/stdc++.h>
using namespace std;
const int N = 3001;
int n;
bool vis[N][N];
double dp[N][N], p[N];
double f(int i, int head, int tail) {
    if (i == n + 1) return head > tail ? 1 : 0;
    if (vis[i][head]) return dp[i][head];
    vis[i][head] = true;
    double ans = p[i] * f(i + 1, head + 1, tail);
    ans += (1 - p[i]) * f(i + 1, head, tail + 1);
    return dp[i][head] = ans;
}
int32_t main() {
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> p[i];
    cout << fixed << setprecision(10) << f(1, 0, 0) << "\n";
    return 0;
}
/*There are N coins, numbered 1,2,,N. For each i (1 iN ), when Coin i is
tossed, it comes up heads with probability pi and tails with probability 1
pi.*/
```

2.5 Dice Combinations

```
/*Your task is to count the number of ways to construct sum n by throwing a
dice one or more times. Each throw produces an outcome between 1 and 6.*/
int32_t main() {
    int n; cin >> n;
    memset(dp, -1, sizeof dp);
    cout << ways(n) % mod << endl;
    return 0;
}
#include <bits/stdc++.h>
using namespace std;
#define int long long
const int N = 1e6 + 9, mod = 1e9 + 7;
int dp[N];
int ways(int n) {
    if (n < 0) return 0;
    if (n == 1) return 1;
    if (n == 2) return 2;
    if (n == 3) return 4;
    if (n == 4) return 8;
    if (n == 5) return 16;
    if (n == 6) return 32;
    if (dp[n] != -1) return dp[n];
    return dp[n] = (ways(n - 1) + ways(n - 2) + ways(n - 3) +
                    ways(n - 4) + ways(n - 5) + ways(n - 6)) % mod;
}
```

2.6 Frog1

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
const int N = 1e5 + 9, inf = 1e9;
int n, v[N], dp[N];
int frog(int i) {
    if (i > n) return inf;
    if (i == n) return 0;
    if (dp[i] != -1) return dp[i];
    int ans = inf;
    if (i + 1 <= n) ans = min(ans, frog(i + 1) + abs(v[i] - v[i + 1]));
    if (i + 2 <= n) ans = min(ans, frog(i + 2) + abs(v[i] - v[i + 2]));
    return dp[i] = ans;
}
int32_t main() {
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> v[i];
    memset(dp, -1, sizeof dp);
    cout << frog(1) << endl;
    return 0;
}
/*If the frog is currently on Stone i, jump to Stone i+1 or Stone i+2.
Here, a cost of hi hj is incurred, where j is the stone to land on.*/
```

2.7 Frog2

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
const int N = 1e5 + 9, inf = 1e9;
int n, k, a[N], dp[N];
int frog2(int i) {
    if (i > n) return inf;
    if (i == n) return 0;
    if (dp[i] != -1) return dp[i];
    int ans = inf;
    for (int j = i + 1; j <= i + k; j++) {
        ans = min(ans, frog2(j) + abs(a[i] - a[j]));
    }
    return dp[i] = ans;
}
int32_t main() {
    cin >> n >> k;
    for (int i = 1; i <= n; i++) cin >> a[i];
    memset(dp, -1, sizeof dp);
    cout << frog2(1) << endl;
    return 0;
}
/*If the frog is currently on Stone i, jump to Stone i+1, i+2...i+k (k <=
100). Here, a cost of hi hj is incurred, where j is the stone to land on.*/
```

2.8 Grid

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1005, mod = 1e9 + 7;
int n, m, dp[N][N];
char g[N][N];
int f(int i, int j) {
    if (i == n && j == m) return 1;
    if (dp[i][j] != -1) return dp[i][j];
    int ans = 0;
    if (i < n && g[i + 1][j] == '.') ans = f(i + 1, j) % mod;
    if (j < m && g[i][j + 1] == '.') ans = (ans + f(i, j + 1)) % mod;
    return dp[i][j] = ans;
}
int32_t main() {
    cin >> n >> m;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            cin >> g[i][j];
        }
    }
    memset(dp, -1, sizeof dp);
    cout << f(1, 1) % mod << endl;
}
// Print the number of Taro's paths from Square (1,1) to (H,W), modulo 10^9
+ 7.
```

2.9 Knapsack

```
#include <bits/stdc++.h>
using namespace std;
const int N = 105, Wight = 1e5 + 9;
int n, W;
int dp[N][Wight], v[N], w[N];
int backpack(int i, int W) {
    if (i > n) return 0;
    if (dp[i][W] != -1) return dp[i][W];
    int ans = backpack(i + 1, W);
    if (W - w[i] >= 0)  ans = max(ans, backpack(i + 1, W - w[i]) + v[i]);
    return dp[i][W] = ans;
}
int32_t main() {
    cin >> n >> W;
    for (int i = 1; i <= n; i++) cin >> w[i] >> v[i];
    memset(dp, -1, sizeof dp);
    int ans = backpack(1, W);
    cout << ans << endl;
}
```

2.10 LCS

```
void print(int i, int j) {
    if (i >= a.size() || j >= b.size()) return;
    if (a[i] == b[j]) {
        cout << a[i];
        print(i + 1, j + 1);
        return;
    }
    int x = f(i + 1, j);
    int y = f(i, j + 1);
    if (x >= y) print(i + 1, j);
    else print(i, j + 1);
}
int32_t main() {
    cin >> a >> b;
    memset(dp, -1, sizeof dp);
    print(0, 0);
    return 0;
}
```

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 3030;
string a, b;
int dp[N][N];
int f(int i, int j) {
    if (i >= a.size() || j >= b.size()) return 0;
    if (dp[i][j] != -1) return dp[i][j];
    int ans = 0;
    if (a[i] == b[j]) {
        ans = max(ans, f(i + 1, j + 1) + 1);
    }
    else {
        ans = max(ans, f(i + 1, j));
        ans = max(ans, f(i, j + 1));
    }
    return dp[i][j] = ans;
}
```

2.11 Longest Path

```cpp
#include <bits/stdc++.h>
using namespace std;
vector<int> dp(100001);
vector<vector<int>> adj(100001);
int dfs(int x) {
    if (dp[x]) return dp[x];
    for (auto e : adj[x]) {
        dp[e] = dfs(e);
        dp[x] = max(dp[e] + 1, dp[x]);
    }
    return dp[x];
}
int main() {
    int n, m; cin >> n >> m;
    for (int i = 0; i < m; ++i) {
        int a, b;
        cin >> a >> b;
        a--; b--; // convert to 0-indexed
        adj[a].push_back(b);
    }
    for (int i = 0; i < n; ++i) dfs(i);
    int ans = 0;
    for (int i = 0; i < n; ++i) ans = max(dp[i], ans);
    cout << ans;
}
// Length of the longest directed path in a Directed Acyclic Graph (DAG).
/* There is a directed graph G with N vertices and M edges.
The vertices are numbered 1, 2, ..., N. For each i (1 ≤ i ≤ M),
the i-th directed edge goes from vertex xi to yi. G does not contain
cycles.
Find the length of the longest directed path in G.
Here, the length of a directed path is the number of edges in it.*/
```

2.12 Minimizing Coins

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 105, X = 1e6 + 9, inf = 1e9;
int n, x, a[N];
int dp[N][X];
int32_t main() {
    cin >> n >> x;
    for (int i = 1; i <= n; i++) cin >> a[i];
    dp[0][0] = 0;
    for (int sum = 1; sum <= x; sum++) dp[0][sum] = inf;
    for (int i = 1; i <= n; i++) {
        for (int sum = 0; sum <= x; sum++) {
            dp[i][sum] = dp[i - 1][sum];
            if (sum >= a[i]) dp[i][sum] = min(dp[i][sum], dp[i][sum - a[i]]
+1); }}
 cout << (dp[n][x] >= inf ? -1 : dp[n][x]) << '\n';
 return 0;
}
/* Your task is to produce a sum of money x using the available coins
in such a way that the number of coins is minimal.
For example, if the coins are {1,5,7} and the desired sum is 11,
an optimal solution is 5 + 5 + 1 which requires 3 coins.*/
```

2.13 Money Sum

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 105, MX_SUM = 1e5 + 9;
int a[N], n;
bool dp[N][MX_SUM], vis[N][MX_SUM];
int f(int i, int sum) {
    if (i == n + 1) return sum == 0;
    if (vis[i][sum]) return dp[i][sum];
    bool is_possible = f(i + 1, sum);
    if (sum >= a[i]) is_possible |= f(i + 1, sum - a[i]);
    vis[i][sum] = true;
    return dp[i][sum] = is_possible;
}
int32_t main() {
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> a[i];
    vector<int> ans;
    for (int sum = 1; sum < MX_SUM; sum++) {
        if (f(1, sum)) ans.push_back(sum);
    }
    cout << ans.size() << "\n";
    for (auto it : ans) {
        cout << it << " ";
    }
}
/*You have n coins with certain values.
Your task is to find all money sums you can create using these coins.*/
```

2.14 Removal Game

```cpp
#include <bits/stdc++.h>
using namespace std;
#define int long long
const int N = 5005;
int a[N], n;
int dp[N][N], dp1[N][N];
int s(int i, int j);
int f(int i, int j) {
    if (i > j) return 0;
    if (dp[i][j] != -1) return dp[i][j];
    int u = a[i] + s(i + 1, j);
    int v = a[j] + s(i, j - 1);
    return dp[i][j] = max(u, v);}
int s(int i, int j) {
    if (i > j) return 0;
    if (dp1[i][j] != -1) return dp1[i][j];
    int u = f(i + 1, j);
    int v = f(i, j - 1);
    return dp1[i][j] = min(u, v);}
int32_t main() {
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> a[i];
    memset(dp, -1, sizeof dp);
    memset(dp1, -1, sizeof dp1);
    cout << f(1, n) << "\n";}
/*There is a list of n numbers and two players who move alternately.
On each move, a player removes either the first or last number from the list,
and their score increases by that number.Both players play optimally.
What is the maximum possible score for the first player?*/
```

2.15 Removing Digits

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1e6 + 9, inf = 1e9;
int dp[N];
int RDX(int n) {
    if (n < 0) return inf;
    if (n == 0) return 0;
    if (dp[n] != -1) return dp[n];
    int ans = inf;
    int f = n;
    while (n != 0) {
        int r = n % 10;
        if (r != 0) ans = min(ans, RDX(f - r) + 1);
        n /= 10;
    }
    return dp[f] = ans;
}
int32_t main() {
    int n; cin >> n;
    memset(dp, -1, sizeof dp);
    cout << RDX(n) << "\n";
}
/*You are given an integer n. On each step, you may subtract one of its
digits from the number. How many steps are needed to make the number equal
to 0?*/
```

2.16 Two Sets Equal Sum

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 505, M = 125005, mod = 1e9 + 7;
long long n, dp[N][M], total;
int f (int i, long long s) {
    if(i == n) {
        if(2 * s == total) return 1;
        else return 0;
    }
    long long &ans = dp[i][s];
    if(ans != -1) return ans;
    ans = (f(i + 1, s + i) % mod);
    ans += (f(i + 1, s) % mod);
    return ans;
}
int32_t main() {
    cin >> n;
    total = n * (n + 1) / 2;
    if(total % 2) {
        cout << 0 << "\n";
        return 0;
    }
    memset(dp, -1, sizeof dp);
    cout << (f(1, 0) % mod) << "\n";
    return 0;
}
/*count the number of ways numbers
1,2,....,n can be two sets of equal sum.
EX, if n=7, there are four solutions:
{1,3,4,6} and {2,5,7}
{1,2,5,6} and {3,4,7}
{1,2,4,7} and {3,5,6}
{1,6,7} and {2,3,4,5}*/
```

Longest Increasing Subsequence

```cpp
vector<long long> lis;
for (auto x : a) {
    auto it = lower_bound(lis.begin(), lis.end(), x);
    if (it == lis.end()) lis.push_back(x);
    else *it = x;
}
```

2.17 Vaction

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 5;
int dp[N][5];
int n, a[N], b[N], c[N];
int f(int i, int last) {
    if(i == n + 1) return 0;
    if(dp[i][last] != -1) return dp[i][last];
    int ans = 0;
    for(int j = 1; j <= 3; j++) {
        if(j != last && j == 1) ans = max(ans, f(i + 1, j) + a[i]);
        if(j != last && j == 2) ans = max(ans, f(i + 1, j) + b[i]);
        if(j != last && j == 3) ans = max(ans, f(i + 1, j) + c[i]);
    }
    return dp[i][last] = ans;
}
int32_t main() {
    cin >> n;
    for(int i = 1; i <= n; i++) cin >> a[i] >> b[i] >> c[i];
    memset(dp, -1, sizeof dp);
    cout << f(1, 0) << '\n';
}
/* The vacation consists of N days. For each day, Taro chooses one of 3
activities.
Find the maximum total happiness when he cannot do the same activity on two
consecutive days. */
```

3 Graphs

3.1 BFS

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 10;
int dist[N], par[N];
vector<int> adj[N];
queue<int> q;
void bfs(int s) {
    memset(dist, 63, sizeof(dist));
    dist[s] = 0;
    q.push(s);
    while (!q.empty()) {
        int u = q.front();
        q.pop();

        for (auto v : adj[u]) {
            if (dist[v] > dist[u] + 1) {
                par[v] = u;
                dist[v] = dist[u] + 1;
                q.push(v);
            }
        }
    }
}
int32_t main() {
    int n, m; cin >> n >> m; // n = nodes, m = edges
    for(int i = 0; i < m; i++) {
        int u, v; cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    int start_node; cin >> start_node;
    bfs(start_node);
    // Example: print distances from start_node
    for(int i = 1; i <= n; i++) {
        cout << "Distance" << start_node << "to" << i << "=" << dist[i];
    }
}
```

3.2 Bellman Ford

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 3e5 + 9;
struct st {
    int a, b, cost;
} e[N];
const int INF = 2e9;
int32_t main() {
    int n, m; cin >> n >> m;
    for (int i = 0; i < m; i++)  cin >> e[i].a >> e[i].b >> e[i].cost;
    int s; cin >> s; // starting node
    // distance vector
    vector<int> d(n, INF); // for finding any cycle set d[i] = 0 for all i
    d[s] = 0;
    vector<int> p(n, -1); // parent vector
    int x;
    for (int i = 0; i < n; ++i) {
        x = -1;
        for (int j = 0; j < m; ++j) {
            if (d[e[j].a] < INF) {
                if (d[e[j].b] > d[e[j].a] + e[j].cost) {
                    d[e[j].b] = max(-INF, d[e[j].a] + e[j].cost);
                    p[e[j].b] = e[j].a;
                    x = e[j].b;
                }
            }
        }
    }
    if (x == -1) {
        cout << "No negative cycle from " << s;
    } else {
        int y = x; // x can be on any cycle or reachable from some cycle
        for (int i = 0; i < n; ++i) y = p[y];
        vector<int> path;
        for (int cur = y;; cur = p[cur]) {
            path.push_back(cur);
            if (cur == y && path.size() > 1) break;
        }
        reverse(path.begin(), path.end());
        cout << "Negative cycle: ";
        for (int i = 0; i < path.size(); ++i) cout << path[i] << ' ';
    }
}
```

3.3 Bipartite Garphs

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 9;
vector<int> g[N];
bool vis[N];
int col[N];
bool ok;
void dfs(int u) {
    vis[u] = true;
    for (auto v : g[u]) {
        if (!vis[v]) {
            col[v] = col[u] ^ 1;
            dfs(v);
        } else {
            if (col[u] == col[v]) {
                ok = false;
            }
        }
    }
}

int32_t main() {
    int n, m; cin >> n >> m;
    while (m--) {
        int u, v; cin >> u >> v;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    ok = true;
    for (int i = 1; i <= n; i++)
    {
        if(!vis[i]) dfs(i);
    }
    if(ok) cout << "YES\n";
    else cout << "NO\n";
}
```

3.4 Cycle Detection

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 5e5 + 9;
vector<pair<int, int>> g[N];
int vis[N], par[N], e_id[N];
vector<int> cycle; // simple cycle, contains edge ids
bool dfs(int u) {
    if(!cycle.empty()) return 1;
    vis[u] = 1;
    for (auto [v, id] : g[u]) {
        if (v != par[u]) {
            if (vis[v] == 0) {
                par[v] = u;
                e_id[v] = id;
                if (dfs(v)) return 1;
            }
            else if (vis[v] == 1) { // cycle found
                cycle.push_back(id);
                for (int x = u; x != v; x = par[x]) {
                    cycle.push_back(e_id[x]);
                }
                return 1;
            }
        }
    }
    vis[u] = 2;
    return 0;
}
int32_t main() {
    int n, m; cin >> n >> m;
    for (int i = 1; i <= m; i++) {
        int u, v;
        cin >> u >> v;
        ++u; ++v; // 0-indexed to 1-indexed adjustment?
        g[u].push_back({v, i});
        g[v].push_back({u, i}); } // add for undirected graph

    for (int u = 1; u <= n; u++) {
        if (vis[u] == 0 && dfs(u)) {
            cout << cycle.size() << '\n';
            for (auto x : cycle) cout << x - 1 << '\n';
            return 0; }
    }
    cout << -1 << '\n';
}  //How to check if an undirected graph has a cycle or not?
```

3.5 DFS

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 9;
vector<int> g[N];
bool vis[N];
void dfs(int u) {
    vis[u] = true;
    for (auto v : g[u]) {
        if(!vis[v]) dfs(v); }
}
int32_t main() {
    int n, m; cin >> n >> m;
    while (m--) {
        int u, v; cin >> u >> v;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    for (int i = 1; i <= n; i++) {
        if(!vis[i]) dfs(i); }}
```

3.6 Dijkstra

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 3e5 + 9, mod = 998244353;
int n, m;
vector<pair<int,int>> g[N], gr[N];// gr = reverse graph
vector<long long> dijkstra(int s, vector<int> &cnt,
                           vector<pair<int,int>> graph[]) {
    const long long inf = 1e18;
    priority_queue<pair<long long,int>, vector<pair<long long,int>>,
                           greater<>> pq;
    vector<long long> d(n + 1, inf);
    vector<bool> vis(n + 1, false);
    pq.push({0, s});
    d[s] = 0;
    cnt.assign(n + 1, 0);
    cnt[s] = 1;
    while(!pq.empty()) {
        auto [dist, u] = pq.top(); pq.pop();
        if(vis[u]) continue;
        vis[u] = true;
        for(auto [v, w] : graph[u]) {
            if(d[u] + w < d[v]) {
                d[v] = d[u] + w;
                pq.push({d[v], v});
                cnt[v] = cnt[u];
            } else if(d[u] + w == d[v]) {
                cnt[v] = (cnt[v] + cnt[u]) % mod;
            }
        }
    }
    return d;
}
int u[N], v[N], w[N];
int32_t main() {
    int s, t; cin >> n >> m >> s >> t;
    for(int i = 1; i <= m; i++) {
        cin >> u[i] >> v[i] >> w[i];
        g[u[i]].push_back({v[i], w[i]});
        gr[v[i]].push_back({u[i], w[i]}); // reverse graph
    }
    vector<int> cnt_s, cnt_t;
    auto d1 = dijkstra(s, cnt_s, g);  // distances from s
    auto d2 = dijkstra(t, cnt_t, gr); // distances to t via reverse graph
    long long ans = d1[t];
    for(int i = 1; i <= m; i++) {
        int x = u[i], y = v[i];
        long long nw = d1[x] + w[i] + d2[y];
        if(nw == ans && 1LL * cnt_s[x] * cnt_t[y] % mod == cnt_s[t]) {
            cout << "YES\n";
        } else if(nw - ans + 1 < w[i]) {
            cout << "CAN " << nw - ans + 1 << "\n";
        } else {
            cout << "NO\n"; /*
        }                    Example Input:    Output:
    }                        4 4 1 4            CAN 1
}                            1 2 1              CAN 1
                             2 4 2               NO
                             1 3 2               NO
                             3 4 2
                             Explanation:
                             - Shortest path length from 1->4: 3
                             - Edge 1->2: nw=3, not unique shortest → CAN 1
                             - Edge 2->4: nw=3, not unique shortest → CAN 1
                             - Edge 1->3: nw=4 → NO
                             - Edge 3->4: nw=4 → NO
                             */
```

3.7 Floyd Warshall

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 105;
int d[N][N];
int main() {
    int n = 10;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            if (i != j) d[i][j] = 1e9;
            else d[i][j] = 0;
        }
    }
    for (int k = 1; k <= n; ++k) {
        for (int i = 1; i <= n; ++i) {
            for (int j = 1; j <= n; ++j) {
                d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
            }
        }
    }
}
```

3.8 Krushkals MST

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 3e5 + 9, mod = 1e9;
struct dsu {
    vector<int> par, rnk, size;
    int c; // connected components
    dsu(int n) : par(n+1), rnk(n+1,0), size(n+1,1), c(n) {
        for (int i = 1; i <= n; ++i) par[i] = i;
    }
    int find(int i) {
        return (par[i] == i ? i : (par[i] = find(par[i])));
    }
    bool same(int i, int j) { return find(i) == find(j); }
    int get_size(int i) { return size[find(i)]; }
    int count() { return c; } // number of connected components
    int merge(int i, int j) {
        if ((i = find(i)) == (j = find(j))) return -1; // already connected
        --c; // reduce connected component count
        if (rnk[i] > rnk[j]) swap(i, j);
        par[i] = j;
        size[j] += size[i];
        if (rnk[i] == rnk[j]) rnk[j]++;
        return j;
    }
};
int32_t main() {
    int n, m;  cin >> n >> m;
    vector<array<int, 3>> ed; // edges: {weight, u, v}
    for(int i = 1; i <= m; i++){
        int u, v, w;
        cin >> u >> v >> w;
        ed.push_back({w, u, v});
    }
    sort(ed.begin(), ed.end()); // sort edges by weight
    long long ans = 0;
    dsu d(n);
    for (auto e : ed) {
        int w = e[0], u = e[1], v = e[2];
        if (d.same(u, v)) continue; // already connected, skip
        ans += w;                   // add weight to MST
        d.merge(u, v);              // merge components
    }
    cout << ans << '\n'; // total weight of MST
}
```

3.9 LCA

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 3e5 + 9, LG = 18;
vector<int> g[N];
int par[N][LG + 1], dep[N], sz[N];
void dfs(int u, int p = 0) {
    par[u][0] = p;
    dep[u] = dep[p] + 1;
    sz[u] = 1;
    for (int i = 1; i <= LG; i++)
        par[u][i] = par[par[u][i - 1]][i - 1];

    for (auto v : g[u]) if (v != p) {
        dfs(v, u);
        sz[u] += sz[v];
    }
}
int lca(int u, int v) {
    if (dep[u] < dep[v]) swap(u, v);
    for (int k = LG; k >= 0; k--) {
        if (dep[par[u][k]] >= dep[v]) u = par[u][k];
    }
    if (u == v) return u;
    for (int k = LG; k >= 0; k--) {
        if (par[u][k] != par[v][k]) u = par[u][k], v = par[v][k];
    }
    return par[u][0];
}
int kth(int u, int k) {
    assert(k >= 0);
    for (int i = 0; i <= LG; i++) if (k & (1 << i)) u = par[u][i];
    return u;
}
int dist(int u, int v) {
    int l = lca(u, v);
    return dep[u] + dep[v] - 2 * dep[l];
}
int go(int u, int v, int k) {
    int l = lca(u, v);
    int d = dep[u] + dep[v] - 2 * dep[l];
    assert(k <= d);
    if (dep[l] + k <= dep[u]) return kth(u, k);
    k -= dep[u] - dep[l];
    return kth(v, dep[v] - dep[l] - k);
}
int32_t main() {
    int n; cin >> n;
    for (int i = 1; i < n; i++) {
        int u, v;
        cin >> u >> v;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    dfs(1);
    int q;
    cin >> q;
    while (q--) {
        int u, v;
        cin >> u >> v;
        cout << dist(u, v) << '\n';
    }
    return 0;
}
```

3.10 Prims MST

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 2020;
int g[N][N], w[N], to[N], selected[N];
long long Prims(int n, vector<pair<int,int>> &edges) {
    long long ans = 0;
    for(int i = 1; i <= n; i++) w[i] = 1e9, selected[i] = 0, to[i] = -1;
    w[1] = 0;
    for(int i = 1; i <= n; i++) {
        int u = -1;
        for(int j = 1; j <= n; j++) {
            if(!selected[j] && (u == -1 || w[j] < w[u])) u = j;
        }
        if (w[u] == 1e9) return -1;
        selected[u] = 1;
        ans += w[u];
        if(to[u] != -1) edges.emplace_back(u, to[u]);
        for(int v = 1; v <= n; v++) {
            if(g[u][v] < w[v]) w[v] = g[u][v], to[v] = u;
        }
    }
    return ans;
}
string s[N];
int main() {
    int n, m; cin >> n >> m;
    for(int i = 1; i <= n; i++) for(int j = 1; j <= n; j++) g[i][j] = 1e9;
    for(int i = 1; i <= n; i++) cin >> s[i];
    for(int i = 1; i <= n; i++){
        for(int j = i + 1; j <= n; j++){
            int w = 0;
            for(int k = 0; k < m; k++) {
                w = max(w, (int)abs(s[i][k] - s[j][k]));
            }
            g[i][j] = min(g[i][j], w);
            g[j][i] = min(g[j][i], w);
        }
    }
    vector<pair<int,int>> ed;
    long long ans = Prims(n, ed);
    int res = 0; for(auto e: ed) res = max(res, g[e.first][e.second]);
    cout << res << '\n';
}
```

3.11 Strongly Connected Components

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 3e5 + 9;
bool vis[N];
vector<int> g[N], r[N], G[N], vec;
void dfs1(int u) {
    vis[u] = 1;
    for(auto v: g[u]) if(!vis[v]) dfs1(v);
    vec.push_back(u);
}
vector<int> comp;
void dfs2(int u) {
    comp.push_back(u);
    vis[u] = 1;
    for(auto v: r[u]) if(!vis[v]) dfs2(v);
}
int idx[N], in[N], out[N];
int32_t main() {
    int n, m; cin >> n >> m;
    for(int i = 1; i <= m; i++) {
        int u, v; cin >> u >> v;
        g[u].push_back(v);
        r[v].push_back(u);
    }
    // Kosaraju step 1: order vertices by finish time
    for(int i = 1; i <= n; i++) if(!vis[i]) dfs1(i);
    reverse(vec.begin(), vec.end());
    // Kosaraju step 2: process reversed graph
    memset(vis, 0, sizeof vis);
    int scc = 0;
    for(auto u: vec) {
        if(!vis[u]) {
            comp.clear();
            dfs2(u);
            scc++;
            for(auto x: comp) idx[x] = scc;
        }
    }
    // Build DAG of SCCs
    for(int u = 1; u <= n; u++) {
        for(auto v: g[u]) {
            if(idx[u] != idx[v]) {
                in[idx[v]]++;
                out[idx[u]]++;
                G[idx[u]].push_back(idx[v]);
            }
        }
    }
    int needed_in = 0, needed_out = 0;
    for(int i = 1; i <= scc; i++) {
        if(!in[i]) needed_in++;
        if(!out[i]) needed_out++;
    }
    int ans = max(needed_in, needed_out);
    if(scc == 1) ans = 0;
    cout << ans << '\n';
}
```

3.12 Topological Sorting

```cpp
const int N = 1e5 + 9;
vector<int> g[N];
bool vis[N];
vector<int> ord;
void dfs(int u) {
    vis[u] = true;
    for (auto v : g[u]) if (!vis[v]) dfs(v);
    ord.push_back(u);}
int32_t main() {
    int n, m; cin >> n >> m;
    while (m--) {
        int u, v; cin >> u >> v;
        g[u].push_back(v);}
    for (int i = 1; i <= n; i++) if (!vis[i]) dfs(i);
    reverse(ord.begin(), ord.end());
    vector<int> pos(n + 1);
    for (int i = 0; i < (int)ord.size(); i++) pos[ord[i]] = i;
    for (int u = 1; u <= n; u++) {
        for (auto v : g[u]) {
            if (pos[u] > pos[v]) {
                cout << "IMPOSSIBLE\n";
                return 0;}
        }
    }
    for (auto u : ord) cout << u << ' ';
}
```

3.13 Tree Diameter

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 2e5 + 9;
vector<int> g[N];
int farthest(int s, int n, vector<int> &d) {
    static const int inf = N;
    d.assign(n + 1, inf);
    d[s] = 0;
    vector<bool> vis(n + 1);
    queue<int> q;
    q.push(s);
    vis[s] = 1;
    int last = s;
    while (!q.empty()) {
        int u = q.front(); q.pop();
        for (int v : g[u]) {
            if (vis[v]) continue;
            d[v] = d[u] + 1;
            q.push(v);
            vis[v] = 1;
        }
        last = u;
    }
    return last;
}
int32_t main() {
    int n; cin >> n;
    for (int i = 1; i < n; i++) {
        int u, v; cin >> u >> v;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    vector<int> dx, dy;
    int x = farthest(1, n, dx);
    int y = farthest(x, n, dx);
    farthest(y, n, dy);
    for (int i = 1; i <= n; i++) {
        cout << max(dx[i], dy[i]) << ' ';
    }
}
```

3.14 Zero One BFS

```cpp
const int N = 1e5 + 5;
using pii = pair<int,int>;
int dist[N];
vector<pii> adj[N];
deque<pii> dq;
void zero_one_bfs(int x) {
    fill(dist, dist + N, INT_MAX);
    dist[x] = 0;
    dq.push_back({x, 0});
    while(!dq.empty()) {
        int u = dq.front().first;
        int ud = dq.front().second;
        dq.pop_front();
        if(dist[u] < ud) continue;
        for(auto p : adj[u]) {
            int v = p.first;
            int w = p.second;
            if(dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w;
                if(w) dq.push_back({v, dist[v]});
                else dq.push_front({v, dist[v]});
            }
        }
    }
}
```

4 Number Theory

4.1 BIGMOD

```
// Big Mod
long long binpow(long long a, long long b, long long m) {
    a %= m;
    long long res = 1;
    while (b > 0) {
        if (b & 1) res = res * a % m;
        a = a * a % m;
        b >>= 1;
    }
    return res;
}
// Binary Multiplication with Mod
long long binmul(long long a, long long b, long long m) {
    long long res = 0LL;
    a %= m;
    while (b > 0) {
        if (b & 1) res = (res + a) % m;
        a = (a + a) % m;
        b >>= 1;
    }
    return res;
}
```

4.2 Basics

```
// Greatest Common Divisor & Lowest Common Multiple
ll gcd(ll a, ll b) { return b ? gcd(b, a % b) : a; }
ll lcm(ll a, ll b) { return a / gcd(a, b) * b; }
// Multiply caring overflow
ll mulmod(ll a, ll b, ll m = MOD) {
    ll r = 0;
    for (a %= m; b; b >>= 1, a = (a * 2) % m) if(b & 1) r = (r + a) % m;
    return r;
}
// Another option for mulmod using long double
ull mulmod(ull a, ull b, ull m = MOD) {
    ull q = (ld)a * (ld)b / (ld)m;
    ull r = a * b - q * m;
    return (r + m) % m;
}
// Fast exponential
ll fexp(ll a, ll b, ll m = MOD) {
    ll r = 1;
    for (a %= m; b; b >>= 1, a = (a * a) % m) if(b & 1) r = (r * a) % m;
    return r;
}
```

4.3 Combinatorics Basics

```
const int N = 2e5 + 9, mod = 998244353; // change this if needed
long long fact[N], finv[N], inv[N];
// Precalculate factorials and inverse factorials modulo mod
void precal_factorial(int n) {
    inv[0] = inv[1] = finv[0] = finv[1] = fact[0] = fact[1] = 1;
    for (int i = 2; i <= n; i++) {
        inv[i] = inv[mod % i] * (mod - mod / i) % mod;
        finv[i] = (finv[i - 1] * inv[i]) % mod;
        fact[i] = (fact[i - 1] * i) % mod;
    }
}
long long ncr(long long n, long long r) {
    if (r > n) return 0;
    long long a = (finv[r] * finv[n - r]) % mod;
    return (a * fact[n]) % mod;
}
```

4.4 Counting Digits

```
#include <bits/stdc++.h>
using namespace std;
// Function to find number of digits in n!
int findDigits(int n) {
    if (n < 0) return 0;
    if (n <= 1) return 1;
    double digits = 0;
    for (int i = 2; i <= n; i++) digits += log10(i);
    return floor(digits) + 1;
}
int main() {
    cout << findDigits(100) << endl;
    return 0;
}
```

4.5 Derangement

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1e6 + 9, mod = 1e9 + 7;
int d[N];
int32_t main() {
    d[0] = 1; d[1] = 0;
    for (int i = 2; i < N; i++) {
        d[i] = 1LL * (i - 1) * (d[i - 1] + d[i - 2]) % mod;
    }
    int n;
    cin >> n;
    cout << d[n] << '\n';
}
/*There are n children at a Christmas party,
and each of them has brought a gift.
Everybody must receive a gift that they did not bring themselves.
In how many ways can the gifts be distributed?*/
```

4.6 Euler Phi

```
// Euler's Totient Function for a single number n using prime factorization
int ind = 0, pf = primes[0], ans = n;
while (1LL * pf * pf <= n) {
    if (n % pf == 0) ans -= ans / pf;
    while (n % pf == 0) n /= pf;
    pf = primes[++ind];
}
if (n != 1) ans -= ans / n;

// Euler Totient Function from 1 to N in O(N log log N)
const int N = 1e5 + 9;
int phi[N];
void totient() {
    for (int i = 1; i < N; i++) phi[i] = i;
    for (int i = 2; i < N; i++) {
        if (phi[i] == i) { // i is prime
            for (int j = i; j < N; j += i)
                phi[j] -= phi[j] / i;
        }
    }
}

// Example 1: Single number totient
// n = 12 → prime factors = 2,3 → φ(12) = 12*(1-1/2)*(1-1/3) = 4

// Example 2: Totient from 1 to N (sieve)
// φ(1..10) = [1,1,2,2,4,2,6,4,6,4]
```

4.7 Fibonacci Number Faster

```
#include <bits/stdc++.h>
using namespace std;
int fib(long long n, int mod) {
    assert(n >= 0);
    if (n <= 1) return n;
    int a = 0, b = 1;
    long long i = 1ll << (63 - __builtin_clzll(n) - 1);
    for (; i; i >>= 1) {
        int na = (a * (long long)a + b * (long long)b) % mod;
        int nb = (2ll * a + b) * b % mod;
        a = na;
        b = nb;
        if (n & i) {
            int c = a + b;
            if (c >= mod) c -= mod;
            a = b;
            b = c;
        }
    }
    return b;
}
int32_t main() {
    cout << fib(10, 1e9 + 7) << '\n';
    return 0;
}
```

4.8 GCD LCM

```
int gcd(int a, int b) {
    if (a == 0) return b;
    return gcd(b % a, a);
}
long long lcm(long long a, long long b) {
    return (a / __gcd(a, b)) * b;
}
int array_gcd(const vector<int>& v) {
    int g = v[0];
    for (size_t i = 1; i < v.size(); i++) {
        g = gcd(g, v[i]);
        if (g == 1) break;
    }
    return g;
}
long long array_lcm(const vector<int>& v) {
    long long l = v[0];
    for (size_t i = 1; i < v.size(); i++) {
        l = (l / gcd(l, (long long)v[i])) * v[i];
    }
    return l;
}
// GCD of Differences
int gcd_of_differences(const vector<int> &v) {
    int g = 0;
    for(int i=1;i<v.size();i++){
        g = gcd(g, abs(v[i]-v[i-1]));
    }
    return g;
}
// Count divisible using LCM
long long count_divisible(long long N, const vector<int>& v) {
 long long L = array_lcm(v);
 return N / L;
}
```

```cpp
// Maximum GCD of any two numbers in an array
int main() {
    int n; cin >> n;
    const int MAXA = 1e6 + 1;
    vector<int> freq(MAXA, 0);
    for (int i = 0; i < n; i++) {
        int x; cin >> x; freq[x]++;
    }
    for (int d = MAXA - 1; d >= 1; d--) {
        int count = 0;
        for (int multiple = d; multiple < MAXA; multiple += d) {
            count += freq[multiple];
            if (count >= 2)
            {
                cout << d << "\n"; return 0;
            }}}
    return 0;
}
// Diophantine Check
bool has_solution(int a, int b, int c) {
    return (c % gcd(a,b) == 0);
}
// Binary / Stein's GCD
int binary_gcd(int a, int b) {
    if(a==0) return b;
    if(b==0) return a;
    int k = __builtin_ctz(a|b); // count trailing zeros
    a >>= __builtin_ctz(a);
    while(b){
        b >>= __builtin_ctz(b);
        if(a>b) swap(a,b);
        b -= a;
    }
    return a<<k;
}

/*
GCD & LCM – CP Techniques / Ideas

- Minimize LCM → pick co-prime numbers
- Maximize GCD → pick multiples of same number
- Array GCD = 1 → must have a co-prime element
- LCM divisible by X → subset LCM % X == 0
- Count numbers divisible → use LCM + inclusion-exclusion
- Reduce fractions → divide numerator & denominator by GCD
- Subset pattern → co-prime → low LCM, multiples → high GCD
- Linear Diophantine → solution exists if c % gcd(a,b) == 0
- Co-prime check → gcd(a,b) == 1
- GCD reduction → replace numbers by gcd(x, y)
- LCM growth → adding numbers increases LCM
- Equation scaling → divide coefficients by GCD
- Pairwise GCD/LCM → compute sequentially
- Prime factor → think in prime powers
- Co-prime product → LCM(a,b) = a*b if gcd(a,b)=1
- Multiples → GCD of multiples = multiple
- GCD in rotation → equals GCD of differences
- Divisibility → number divisible → divisible by array LCM
- Binary / bitwise GCD → use Stein's algorithm
- Extended Euclidean → solve ax+by=gcd(a,b)
- GCD in sequences → gcd(a1, a2-a1, ..., an-a1)
- Subset optimization → co-prime reduces LCM, multiples increase GCD
- Modulo tricks → GCD(a % m, b % m) = GCD(a,b) % m
- Co-prime array → array GCD = 1, pairwise not required
- Max product under LCM → pick numbers with minimal pairwise LCM
- GCD frequency → count multiples of each GCD candidate
- Sum divisible by GCD → sum % GCD(array) == 0
- LCM pruning → stop if LCM exceeds limit
- GCD for periodicity → GCD of differences = period
- LCM & prime factors → union for LCM, intersection for GCD
- Subset co-prime → minimize LCM
*/
```

4.9 Josephus

```cpp
// Function to find the last survivor
int Josephus_Any_K(int N, int k){
    int i = 1, ans = 0;
    while (i <= N) {
        ans = (ans + k) % i;
        i++;}
    return ans + 1;
}
int Josephus(int n, int k){
    if(k * 2 <= n) return k * 2;
    int c = n / 2;
    if(n % 2) return 2 * Josephus(n / 2, k - c - 1) + 1;
    else  return 2 * Josephus(n - c, k - c) - 1;
}
int main(){
    int q;cin >> q;
    while(q--){
        int N, k; // N = number of people, K = k-th removed
        cin >> N >> k;
        cout << Josephus(N, 2) << endl;
        cout << Josephus_Any_K(N, k) << endl;
    }
}
```

4.10 Large Number GCD

```cpp
#include <bits/stdc++.h>
using namespace std;
ll gcd(ll a, ll b) {
    if (!a) return b;
    return gcd(b % a, a);
}
ll reduceB(ll a, char b[]) {
    ll mod = 0;
    for (int i = 0; i < strlen(b); i++) mod = (mod * 10 + b[i] - '0') % a;
    return mod;
}
ll gcdLarge(ll a, char b[]) {
    ll num = reduceB(a, b);
    return gcd(a, num);
}
int main() {
    ll a = 1221;
    char b[] = "12345678910111213141516171819202122232425262728 29";
    if (a == 0) cout << b << endl;
    else cout << gcdLarge(a, b) << endl;
}
```

4.11 Lengenders Formula

```cpp
// n and a prime number p, find the largest x such that p^x
// divides n! (factorial) in O(log n).
//Number of times p appears in n --> n/p + n/p² + n/p³ + ...
#include <bits/stdc++.h>
using namespace std;
int legendre(long long n, long long p) {
    int ans = 0;
    while (n) {
        ans += n / p;
        n /= p;
    }
    return ans;
}
```

4.12 Miller Rabin

```cpp
#include <bits/stdc++.h>
using namespace std;
#define ll long long
namespace MillerRabin {
mt19937 rnd(chrono::steady_clock::now().time_since_epoch().count());
const int P = 1e6 + 9;
int primes[P], spf[P];
inline ll mul_mod(ll x, ll y, ll m) {
    ll res = (__int128)x * y % m;
    return res;
}
inline ll pow_mod(ll x, ll n, ll m) {
    ll res = 1 % m;
    for (; n; n >>= 1) {
        if (n & 1) res = mul_mod(res, x, m);
        x = mul_mod(x, x, m);
    }
    return res;
}
inline bool miller_rabin(ll n) {
    if (n <= 2 || (n & 1 ^ 1)) return (n == 2);
    if (n < P) return spf[n] == n;
    ll c, d, s = 0, r = n - 1;
    for (; !(r & 1); r >>= 1, s++) {}
    for (int i = 0; primes[i] < n && primes[i] < 32; i++) {
        c = pow_mod(primes[i], r, n);
        for (int j = 0; j < s; j++) {
            d = mul_mod(c, c, n);
            if (d == 1 && c != 1 && c != (n - 1)) return false;
            c = d;
        }
        if (c != 1) return false;
    }
    return true;
}
void init() {
    int cnt = 0;
    for (int i = 2; i < P; i++) {
        if (!spf[i]) primes[cnt++] = spf[i] = i;
        for (int j = 0, k; (k = i * primes[j]) < P; j++) {
            spf[k] = primes[j];
            if (spf[i] == spf[k]) break;
        }
    }
}
} // namespace MillerRabin
int32_t main() {
    MillerRabin::init();
    int t; cin >> t;
    while (t--) {
        ll n; cin >> n;
        if (MillerRabin::miller_rabin(n)) cout << "Yes\n";
        else cout << "No\n";
    }
}
```

4.13 Modular Arithmetic

```cpp
int main() {
    int a, b, m;
    cin >> a >> b >> m;
    int vagsesh = (a % m - b % m + m) % m;
    // Modular multiplication (a^n % m)
    int n; cin >> n;
    long long res = 1;
    for (int i = 1; i <= n; i++) res = (res * a) % m;
    cout << "Modular Exponentiation (a^n % m): " << res << endl;
}
```

4.14 Number of divisors

```cpp
#include <bits/stdc++.h>
using namespace std;
// Function to calculate the number of divisors of n
long long numberOfDivisors(long long num) {
    long long total = 1;
    for (long long i = 2; i * i <= num; i++) {
        if (num % i == 0) {
            int e = 0;
            while (num % i == 0) {
                e++;
                num /= i;
            }
            total *= (e + 1); // divisor formula
        }
    }
    // If remaining num > 1, it's a prime factor
    if (num > 1) total *= 2;
    return total;
}
int main() {
    long long n; cin >> n;
    cout << "Number of divisors: " << numberOfDivisors(n) << endl;
    return 0;
}
```

4.15 Optimized Sieve

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1e8 + 9;
bitset<N> f; // false = prime, true = not prime
int32_t main() {
    int n = N - 9;
    vector<int> primes;
    for(int i = 3; i <= n; i += 2) f[i] = false;
    f[2] = false;
    for (int i = 3; i * i <= n; i += 2) {
        for (int j = i * i; j <= n; j += 2*i) {
                f[j] = true;
        }
    }
    for (int i = 2; i <= n; i++) {
        if (!f[i]) primes.push_back(i);
    }
    cout << primes.size() << '\n';
}
```

4.16 Prime Factorization Spf

```cpp
const int N = 1e6 + 9;
int spf[N]; // smallest prime factor
int32_t main() {
    for (int i = 2; i < N; i++) spf[i] = i;
    for (int i = 2; i < N; i++) {
        for (int j = i; j < N; j += i) {
            spf[j] = min(spf[j], i);
        }
    }
    int q; cin >> q;
    while (q--) {
        int n; cin >> n; // n <= 1e6
        vector<int> ans;
        while (n > 1) {
            ans.push_back(spf[n]);
            n /= spf[n];
        }
        for (auto x : ans) cout << x << ' ';
    }
}
```

4.17 Prime Factors

```cpp
vector<ll> primes;
vector<ll> prime_factors(ll n) {
    vector<ll> factors;
    int ind = 0;
    ll pf = primes[ind];
    while (pf * pf <= n) {
        while (n % pf == 0) {
            factors.push_back(pf);
            n /= pf;
        }
        ind++;
        if (ind >= primes.size()) break;
        pf = primes[ind];
    }
    if (n != 1) factors.push_back(n);
    return factors;
}
int main() {
    const int N = 1e6 + 9;
    vector<bool> isPrime(N, true);
    isPrime[0] = isPrime[1] = false;
    for (int i = 2; i * i < N; i++) {
        if (isPrime[i]) {
            for (int j = i * i; j < N; j += i) isPrime[j] = false;
        }
    }
    for (int i = 2; i < N; i++) if (isPrime[i]) primes.push_back(i);
    ll n;
    cin >> n; // n <= ~9*10^13
    vector<ll> factors = prime_factors(n);
    for (ll x : factors) cout << x << ' ';
}
```

4.18 Sum of divisors(n)

```cpp
long long SumOfDivisors(long long num) {
    long long total = 1;
    for (int i = 2; (long long)i * i <= num; i++) {
        if (num % i == 0) {
            int e = 0;
            do {
                e++;
                num /= i;
            } while (num % i == 0);
            long long sum = 0, pow = 1;
            do {
                sum += pow;
                pow *= i;
            } while (e-- > 0);
            total *= sum;
        }
    }
    if (num > 1) total *= (1 + num);
    return total;
}
```

4.19 n! Trailing Zero

```cpp
int findTrailingZeros(int n) {
    if (n < 0) return -1;
    int count = 0;
    for (int i = 5; n / i >= 1; i *= 5) count += n / i;
    return count;
}
int main() {
    int n; cin >> n;
    int zeros = findTrailingZeros(n);
    cout << findTrailingZeros(n);
}
```

4.20 Sum of divisors(1 to n)

```cpp
// sum of all divisors of numbers 1..n --> n = 5 then sum of divisors
(1,2,3,4,5) = 21
long long mod = 1e9 + 7;
int main(){
    long long n; cin >> n;
    long long ans = 0;
    long long k = 1;
    while(k <= n) {
        long long q = n / k;
        long long L = n / (q + 1) + 1;
        long long R = n / q;
        long long count = R - L + 1;
        long long sum_d = ((L + R) % mod * (count % mod) % mod * ((mod + 1)
                          / 2)) % mod;
        ans = (ans + sum_d * (q % mod) % mod) % mod;
        k = R + 1;
    }
    cout << ans << endl;
}
```

4.21 Upto n NOD

```cpp
int main() {
    int n = 100;
    vector<int> d(n + 1, 0); // d[i] stores number of divisors of i
    for (int i = 1; i <= n; i++) {
        for (int j = i; j <= n; j += i) {
            d[j]++;      // Count of divisors
            // d[j] += i; // calculate sum of divisors
        }
    }
    for (int i = 1; i <= n; i++) cout << d[i] << ' ';
}
```

Extra Formula

```cpp
const ll MOD = 1e9 + 7;
ll mod_pow(ll a, ll b, ll mod) {
    ll res = 1;
    a %= mod;
    while (b > 0) {
        if (b & 1) res = (res * a) % mod;
        a = (a * a) % mod;
        b >>= 1;
    }
    return res;
}
ll mod_inverse(ll a) {
    return mod_pow(a, MOD - 2, MOD);
}
int main() {
    ll a; cin >> a;
    cout << mod_inverse(a) << '\n';
}
/*
- PRIME FACTORIZATION: n = p1^x1 * p2^x2 * ... * pk^xk
- NUM_DIVISORS: (x1 + 1) * (x2 + 1) * ... * (xk + 1)
- SUM_DIVISORS: (p1^(x1+1)-1)/(p1-1) * ... * (pk^(xk+1)-1)/(pk-1)
- PRODUCT_DIVISORS: n^(num_divisors/2)
- MOD_INV: a^-1 % MOD = pow(a, MOD-2, MOD) if MOD prime
- FERMAT LITTLE THEOREM: a^(p-1) ≡ 1 (mod p) if p prime
- EULER TOTIENT FUNCTION φ(n): count of numbers ≤ n coprime to n
- DIVISIBILITY RULES: quick checks for small primes (2,3,5,7,11,...)
- CHINESE REMAINDER THEOREM: Solve x ≡ a_i (mod m_i) for pairwise coprime m_i
*/
```

5 Miscellaneous

5.1 Base Conberstion

```
string base_convert(int n, int b)
{
    string s = "";
    while (n > 0)
    {
        s = to_string(n % b) + s;
        n /= b;
    }
    return s;
}
int convert_to_decimal(string s, int base)
{
    int n = 0, power = 1;
    for (int i = (int)s.size() - 1; i >= 0; i--)
    {
        n += power * (s[i] - '0');
        power *= base;
    }
    return n;
}
```

5.2 Bitset

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int x; cin >> x;
    int b = 0;
    do
    {
        cout << bitset<3>(b) << "  (" << b << ")" << endl;
    }
    while (b = (b - x) & x);
    return 0;
}
```

5.3 Bitwise Property

```
#include <bits/stdc++.h>
using namespace std;
const int inf = numeric_limits<int>::max() - 5;
#define int long long

// Basic bitwise operations
int AND(int a, int b) { return (a & b); }
int OR(int a, int b) { return (a | b); }
int XOR(int a, int b) { return (a ^ b); }
int right_shift(int a, int b) { return (a >> b); }
int left_shift(int a, int b) { return (a << b); }
int bitwise_not(int a) { return (~a); }

// Bit counts and properties
int ON_BIT(int a) { return __builtin_popcount(a); }// number of 1s
int leading_zero(int a) { return __builtin_clz(a); }// leading zeros 32-bit
int tailing_zero(int a) { return __builtin_ctz(a); }// trailing zeros

// Specific bit checks
int LSB(int a) { return (a & 1); }
bool kth_bit(int a, int k) { return (a & (1 << k)); }// check k-th bit
int msb(int a) { return a ? 32 - __builtin_clz(a) : 0; }
```

```
int main() {
 bitset<100005> bs; // all bits 0 initially

 bs.set(); // set all bits to 1
 bs.set(3); // set 3rd bit to 1
 bs.reset(3); // reset 3rd bit to 0
 bs.reset(); // reset all bits to 0
 bs.flip(5); // flip 5th bit
 bs.flip(); // flip entire bitset
 bool f = bs.test(5); // check if 5th bit is set
 int cnt = bs.count(); // count number of 1s
 bool ok1 = bs.all(); // check if all bits are 1
 bool ok2 = bs.any(); // check if any bit is 1
 bool ok3 = bs.none(); // check if no bits are 1
 unsigned long long x = bs.to_ullong(); // convert to number
 string s = bs.to_string(); // convert to string
}
```

5.4 Builtin

```
// Count number of set bits (1s)
int pc = __builtin_popcount(x);        // for int
int pcLL = __builtin_popcountll(y);    // for long long

// Count trailing zeros (least significant zeros)
int tz = __builtin_ctz(x);             // for int
int tzLL = __builtin_ctzll(y);         // for long long

// Count leading zeros (most significant zeros)
int lz = __builtin_clz(x);             // for int (32-bit)
int lzLL = __builtin_clzll(y);         // for long long (64-bit)

// Check if number is power of two
bool isP2 = (x > 0) && ((x & (x - 1)) == 0);

// Find position of least significant 1-bit (1-indexed)
int ffs_index = __builtin_ffs(x);      // returns 0 if x == 0
```

5.5 Merge Sort

```
int n;
long long inv = 0;  // inversion count
vector<int> v, ans; // original array & temporary array
void mergesort(int l, int r, vector<int> &v) {
    if (l >= r) return; // base case
    int mid = (l + r) / 2;
    mergesort(l, mid, v);  // left half
    mergesort(mid + 1, r, v);  // right half
    int i = l, j = mid + 1, k = l;
    while (i <= mid || j <= r) {
        if (i <= mid && (j > r || v[i] <= v[j])) {
            ans[k++] = v[i++];
        } else {
            ans[k++] = v[j++];
            inv += (mid - i + 1); // count inversions
        }
    }
    for (int i = l; i <= r; i++) v[i] = ans[i]; // copy original array
}
int main() {
    v = {2, 3, 8, 6, 1};
    n = v.size();
    ans.resize(n);
    mergesort(0, n - 1, v);
    cout << "Sorted array: ";
    for (auto x : v) cout << x << " ";
    cout << endl;
    cout << "Number of inversions: " << inv << endl;
}
```

5.6 STL

```
// ---------- tuple ----------
tuple<int, int, int> t;
get<0>(t); get<1>(t); get<2>(t);
auto [x1, x2, x3] = t;
// ---------- deque ----------
deque<int> dq;
dq.push_front(1); dq.push_back(3); dq.pop_front(); dq.pop_back();
dq.size(); dq.empty(); dq.front(); dq.back();
// ---------- stack ----------
stack<int> st;
st.push(2); st.pop(); st.top(); st.size(); st.empty();
// ---------- queue ----------
queue<int> q;
q.push(5); q.pop(); q.front(); q.back(); q.size(); q.empty();
// ---------- priority queue ----------
// Max priority first (largest value removed first)
priority_queue<long long> pq_max;
// Min priority first (smallest value removed first)
priority_queue<long long, vector<long long>, greater<long long>> pq_min;
// Operations
pq_max.push(x); pq_max.top(); pq_max.pop(); // Time Complexity = O(log n)
// ---------- map ----------
map<int, int> mp1;
map<int, pair<int, int>> mp2;
mp2[0].first; mp2[0].second;
// Lower bound & upper bound in vector
int index_lb = lower_bound(v.begin(), v.end(), val) - v.begin();
int index_ub = upper_bound(v.begin(), v.end(), val) - v.begin();
// Lower bound & upper bound in set
auto it_lb = s.lower_bound(6); *it_lb; it_lb--;
auto it_ub = s.upper_bound(6); *it_ub; it_ub--;
// ---------- multiset ----------
multiset<int> ms;
ms.insert(x); ms.erase(ms.find(x)); ms.begin(); ms.end(); ms.size();
ms.count(x); ms.empty(); ms.lower_bound(x); ms.upper_bound(x);
// ---------- unordered_map / unordered_set ----------
unordered_map<int, int> ump;
ump[x] = 5; ump.find(x); ump.erase(x); ump.count(x);

unordered_set<int> us;
us.insert(x); us.erase(x); us.find(x); us.count(x); us.size(); us.empty();
// ---------- vector ----------
vector<int> vec;
vec.push_back(1); vec.pop_back(); vec.size(); vec.empty();
vec.begin(); vec.end(); sort(vec.begin(), vec.end());
// ---------- string ----------
string str = "hello";
str.size(); str.empty(); str.push_back('a'); str.pop_back();
str.substr(0, 2); str.find('l');
// ---------- set ----------
set<int> s_set;
s_set.insert(5); s_set.erase(5); s_set.find(5); s_set.count(5);
s_set.lower_bound(4); s_set.upper_bound(6); s_set.size(); s_set.empty();
// ---------- unordered_multiset / unordered_multimap ----------
unordered_multiset<int> ums;
ums.insert(5); ums.erase(5); ums.count(5);

unordered_multimap<int,int> umm;
umm.insert({1,2}); umm.erase(1); umm.find(1);
// ---------- numeric / algorithm utilities ----------
iota(vec.begin(), vec.end(), 0);
reverse(vec.begin(), vec.end());
rotate(vec.begin(), vec.begin()+1, vec.end());
sort(vec.begin(), vec.end());
unique(vec.begin(), vec.end());
next_permutation(vec.begin(), vec.end());
prev_permutation(vec.begin(), vec.end());
accumulate(vec.begin(), vec.end(), 0);
```

5.7 Sqrt Decomposition

```cpp
#include <bits/stdc++.h>
using namespace std;
// range distinct elemnet
const int N = 1e5 + 1;
const int SQ = 500;
int n, m, v[N], freq[N];
int current_ans = 0;
void add(int p) {
    freq[v[p]]++;
    if (freq[v[p]] == 1) current_ans++; // new distinct element
}
void rem(int p) {
    freq[v[p]]--;
    if (freq[v[p]] == 0) current_ans--; // element no longer present
}
struct query {
    int i, l, r, ans;
};
bool c1(query a, query b) {
    if (a.l / SQ != b.l / SQ) return a.l < b.l;
    return (a.l / SQ & 1) ? (a.r > b.r) : (a.r < b.r);
}
bool c2(query a, query b) {
    return a.i < b.i;
}
int main() {
    cin >> n;
    for(int i = 0; i < n; i++) cin >> v[i];
    cin >> m;
    query qs[m];
    cout << "Enter queries (l r) 0-based indices:\n";
    for(int i = 0; i < m; i++) {
        cin >> qs[i].l >> qs[i].r;
        qs[i].i = i; // store original index
    }
    sort(qs, qs + m, c1);
    int l = 0, r = -1;
    for(int i = 0; i < m; i++) {
        query &q = qs[i];
        while(r < q.r) add(++r);
        while(r > q.r) rem(r--);
        while(l < q.l) rem(l++);
        while(l > q.l) add(--l);
        q.ans = current_ans;
    }
    sort(qs, qs + m, c2);
    for(int i = 0; i < m; i++) {
        cout << "Query " << i+1 << "[" << qs[i].l << "," << qs[i].r << "] ="
             << qs[i].ans << "\n";
    }
}
```

5.8 Binary Search

```cpp
ll isok(ll n)
{
    if(conditon True) return 0;
    else return 1;
}

ll lo = 0, hi  = n;
while (lo < hi) {
    // ll mid = lo + (hi - lo) / 2;
    ll mid = (hi + lo) / 2;
    if (isok(mid) == 0) lo = mid + 1;
    else hi = mid;
}
cout << lo << endl;
```

5.9 Ternary Search

```cpp
#include <bits/stdc++.h>
using namespace std;
#define ll long long
const double EPS = 1e-9;
ll f(ll x) {
    return -x*x + 4*x; // integer version
}
double f_double(double x) {
    return -x*x + 4*x; // double version
}
ll ternary_search_int(ll l, ll r, bool find_max = true) {
    while(r - l > 3) {
        ll m1 = l + (r - l)/3;
        ll m2 = r - (r - l)/3;
        ll f1 = f(m1), f2 = f(m2);
        if(find_max) {
            if(f1 < f2) l = m1;
            else r = m2;
        } else { // find minimum
            if(f1 > f2) l = m1;
            else r = m2;
        }
    }
    ll ans = f(l);
    for(int i = l+1; i <= r; i++) {
        ll tmp = f(i);
        if(find_max) ans = max(ans, tmp);
        else ans = min(ans, tmp);
    }
    return ans;
}
double ternary_search_double(double l, double r, bool find_max = true, int
iterations = 300) {
    for(int i = 0; i < iterations; i++) {
        double m1 = l + (r - l)/3.0;
        double m2 = r - (r - l)/3.0;
        double f1 = f_double(m1), f2 = f_double(m2);
        if(find_max) {
            if(f1 < f2) l = m1;
            else r = m2;
        } else { // find minimum
            if(f1 > f2) l = m1;
            else r = m2;
        }
    }
    return f_double(l);
}
int main() {
    ll l = 0, r = 4;
    cout << "Max (int) = " << ternary_search_int(l, r) << endl;
    cout << "Min (int) = " << ternary_search_int(l, r, false) << endl;
    double ld = 0.0, rd = 4.0;
    cout << "Max (double) = " << ternary_search_double(ld, rd) << endl;
    cout << "Min (double) = " << ternary_search_double(ld, rd, false) <<
endl;
    return 0;
}
```

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
```

5.10 Custom Sorting

```cpp
#include <bits/stdc++.h>
using namespace std;
bool customSort(const pair<int, pair<int,int>> &a,
                const pair<int, pair<int,int>> &b) {
    if (a.first != b.first)
        return a.first < b.first;               // 1st: ascending
    if (a.second.first != b.second.first)
        return a.second.first > b.second.first; // 2nd: descending
    return a.second.second < b.second.second;  // 3rd: ascending
}
int main() {
    vector<pair<int, pair<int,int>>> v1 = {
        {1, {5, 10}},
        {2, {4, 7}},
        {1, {6, 5}},
        {1, {5, 8}}
    };
    sort(v1.begin(), v1.end(), customSort);
    sort(v1.begin(), v1.end(), [](const auto &a, const auto &b) {
        if(a.first != b.first) return a.first < b.first; // 1st: ascending
        if(a.second.first != b.second.first) return a.second.first >
b.second.first; // 2nd: descending
        return a.second.second < b.second.second; // 3rd: ascending
    });
}
```

5.11 python & Run Code Technique

```python
# Optional file I/O
# import sys
# sys.stdout = open('out', 'w')
# sys.stdin = open('in', 'r')
R = lambda: map(int, input().split())
n, k = R()
v, t = [], [0]*n
p_list = list(R())
c_list = list(R())
for p, c, i in sorted(zip(p_list, c_list, range(n))):
    t[i] = sum(v) + c
    v.append(c)
    v.sort(reverse=True)
    if len(v) > k:
        v.pop()
print(' '.join(map(str, t)))
```

```
:: Compile and run C++ program
g++ -std=c++17 -O2 file_name.cpp -o a
.\a
g++ test.cpp -o test && ./test < input > output
g++ test.cpp && ./a.out < input > output
:: Run Python script
python -u "g:\foldername\filename.py"

// 1. Lucas Numbers (L(0) = 2, L(1) = 1, L(n) = L(n-1) + L(n-2))
2, 1, 3, 4, 7, 11, 18, 29, 47, 76, 123, 199, 322, 521

// 2. Triangular Numbers (T(n) = n*(n+1)/2)
1, 3, 6, 10, 15, 21, 28, 36, 45, 55, 66, 78, 91, 105, 120

// 3. Catalan Numbers (C(n) = C(0) = 1, Cn = (2n)! / ((n+1)! n!))
1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012

// 4. Bell Numbers (Bn = number of partitions of set of size n)
1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, 678570

// 5. Gray Codes (G(n) = G(n-1) + reversed(G(n-1) with MSB=1))
0, 1, 3, 2, 6, 7, 5, 4
```

6 Strings

6.1 Hashing

```
namespace FullStringHash {
    const int p1 = 137, mod1 = 127657753;
    const int p2 = 277, mod2 = 987654319;
    const int N = 1e5 + 5;
    int pw1[N], pw2[N];
    void prec() {
        pw1[0] = pw2[0] = 1;
        for (int i = 1; i < N; i++) {
            pw1[i] = 1LL * pw1[i-1] * p1 % mod1;
            pw2[i] = 1LL * pw2[i-1] * p2 % mod2;
        }
    }
    pair<int,int> getHash(const string &s) {
        int h1 = 0, h2 = 0;
        for (int i = 0; i < (int)s.size(); i++) {
            h1 = (h1 + 1LL * s[i] * pw1[i]) % mod1;
            h2 = (h2 + 1LL * s[i] * pw2[i]) % mod2;
        }
        return {h1, h2};
    }
}
struct SingleHash { //SINGLE HASH WITH PREFIX (SUBSTRING)
    const long long MOD = 1000000007;
    const long long P = 31;
    vector<long long> prefix, power;
    SingleHash(const string &s) {
        int n = s.size();
        prefix.assign(n+1, 0);
        power.assign(n+1, 1);
        for (int i = 1; i <= n; i++) {
            power[i] = (power[i-1] * P) % MOD;
            prefix[i] = (prefix[i-1] * P + (s[i-1] - 'a' + 1)) % MOD;
        }
    }
    long long getHash(int l, int r) {
        return (prefix[r+1] - (prefix[l] * power[r-l+1]) % MOD + MOD) % MOD;
    }
};
struct DoubleHash { //DOUBLE HASH WITH PREFIX
    const long long MOD1 = 1000000007;
    const long long MOD2 = 1000000009;
    const long long P1 = 31;
    const long long P2 = 37;
    vector<long long> prefix1, prefix2, power1, power2;
    DoubleHash(const string &s) {
        int n = s.size();
        prefix1.assign(n+1, 0);
        prefix2.assign(n+1, 0);
        power1.assign(n+1, 1);
        power2.assign(n+1, 1);
        for (int i = 1; i <= n; i++) {
            int val = s[i-1];
            power1[i] = (power1[i-1] * P1) % MOD1;
            power2[i] = (power2[i-1] * P2) % MOD2;
            prefix1[i] = (prefix1[i-1] * P1 + val) % MOD1;
            prefix2[i] = (prefix2[i-1] * P2 + val) % MOD2;
        }
    }
    pair<long long,long long> getHash(int l, int r) {
        long long h1 = (prefix1[r+1] - (prefix1[l] * power1[r-l+1]) % MOD1 +
MOD1) % MOD1;
        long long h2 = (prefix2[r+1] - (prefix2[l] * power2[r-l+1]) % MOD2 +
MOD2) % MOD2;
        return {h1, h2};
    }
};
```

```
bool isPalindrome(DoubleHash &Hf, DoubleHash &Hr, int n, int l, int r) {
 auto h1 = Hf.getHash(l, r);
 int rl = n - 1 - r;
 int rr = n - 1 - l;
 auto h2 = Hr.getHash(rl, rr);
 return h1 == h2;
}
int LCP(DoubleHash &H1, DoubleHash &H2, int n1, int n2) {
 int low = 0, high = min(n1, n2), ans = 0;
 while (low <= high) {
 int mid = (low + high) / 2;
 if (mid == 0 || H1.getHash(0, mid-1) == H2.getHash(0, mid-1)) {
 ans = mid;
 low = mid + 1;
 } else high = mid - 1;
 }
 return ans;
}
int main() {
 FullStringHash::prec();
 string fs1, fs2; cin >> fs1 >> fs2;
 cout << (FullStringHash::getHash(fs1) == FullStringHash::getHash(fs2));

 string shstr; cin >> shstr;
 SingleHash SH(shstr);
 int l1,r1,l2,r2; cin >> l1 >> r1 >> l2 >> r2;
 cout << (SH.getHash(l1,r1) == SH.getHash(l2,r2));

 string dhs1,dhs2; cin >> dhs1 >> dhs2;
 DoubleHash DH1(dhs1), DH2(dhs2);
 cout << (DH1.getHash(0,dhs2.size()-1) == DH2.getHash(0,dhs2.size()-1));

 string pstr; cin >> pstr;
 string prstr = pstr;
 reverse(prstr.begin(), prstr.end());
 DoubleHash Hf(pstr), Hr(prstr);
 cout << isPalindrome(Hf,Hr,pstr.size(),0,pstr.size()-1);

 string lcp1,lcp2; cin >> lcp1 >> lcp2;
 DoubleHash LCPH1(lcp1), LCPH2(lcp2);
 cout << LCP(LCPH1,LCPH2,lcp1.size(),lcp2.size()) << "\n";

}
```

6.2 Largest Substring More than K

```
// implement DoubleHash first
int max_oc(DoubleHash &DH, int len, int n){
    map<pair<long long,long long>, int> mp;
    for(int i=0; i+len-1 < n; i++){
        mp[DH.getHash(i,i+len-1)]++;
    }
    int ans = 0;
    for(auto [x,y] : mp) ans = max(ans,y);
    return ans;
}
int32_t main(){
    string s; int k; cin >> s >> k;
    int n = s.size();
    DoubleHash DH(s);
    int l = 1, r = n, ans = -1;
    while(l <= r){
        int mid = (l+r)/2;
        if(max_oc(DH, mid, n) >= k){
            ans = mid; l = mid + 1; // try bigger
        } else {
            r = mid - 1; // try smaller
        }
    }
    cout << ans << '\n';
}
```

6.3 LCP Of Two Substrings

```
int lcp(int i, int j, int x, int y) {
    int l = 1;
    int r = min(j - i + 1, y - x + 1);
    int ans = 0;
    while(l <= r){
        int mid = (l + r) / 2;
        if(get_hash(i, i + mid - 1) == get_hash(x, x + mid - 1)){
            ans = mid;
            l = mid + 1;
        } else {
            r = mid - 1;
        }
    }
    return ans;
}
```

6.4 Longest Common Substring

```
const int N = 1e5 + 9, p1 = 137,mod1 = 127657753, p2 = 277,mod2 = 987654319;
int power(long long n, long long k, int mod) {
    int ans = 1 % mod; n %= mod;
    if (n < 0) n += mod;
    while (k) {
        if (k & 1) ans = (long long)ans * n % mod;
        n = (long long)n * n % mod;
        k >>= 1;
    }
    return ans;
}
int ip1, ip2; pair<int, int> pw[N], ipw[N];
void prec() {
    pw[0] = {1, 1};
    for (int i = 1; i < N; i++) {
        pw[i].first = 1LL * pw[i - 1].first * p1 % mod1;
        pw[i].second = 1LL * pw[i - 1].second * p2 % mod2;
    }
    ip1 = power(p1, mod1 - 2, mod1);
    ip2 = power(p2, mod2 - 2, mod2);
    ipw[0] = {1, 1};
    for (int i = 1; i < N; i++) {
        ipw[i].first = 1LL * ipw[i - 1].first * ip1 % mod1;
        ipw[i].second = 1LL * ipw[i - 1].second * ip2 % mod2;
    }
}
struct Hashing {
    pair<int, int> pref[N];
    void build(const string &s) {
        int n = s.size();
        for (int i = 0; i < n; i++) {
            pref[i].first = 1LL * s[i] * pw[i].first % mod1;
            if (i) pref[i].first = (pref[i].first + pref[i - 1].first)
                                   % mod1;
            pref[i].second = 1LL * s[i] * pw[i].second % mod2;
            if (i) pref[i].second = (pref[i].second + pref[i - 1].second)
                                   % mod2;
        }
    }
    pair<int, int> get_hash(int i, int j) {
        pair<int, int> hs({0, 0});
        hs.first = pref[j].first;
        if (i) hs.first = (hs.first - pref[i - 1].first + mod1) % mod1;
        hs.first = 1LL * hs.first * ipw[i].first % mod1;
        hs.second = pref[j].second;
        if (i) hs.second = (hs.second - pref[i - 1].second + mod2) % mod2;
        hs.second = 1LL * hs.second * ipw[i].second % mod2;
        return hs;
    }
} A, B;
```

```
string a, b, res;
bool ok(int k) {// is there a k length substring that occurs in both a and b
    set<pair<int, int>> substring_hashes_in_a;
    int len_a = a.size();
    int len_b = b.size();
    for (int i = 0; i + k - 1 < len_a; i++)
        substring_hashes_in_a.insert(A.get_hash(i, i + k - 1));
    for (int i = 0; i + k - 1 < len_b; i++) {
        auto substring_hash_in_b = B.get_hash(i, i + k - 1);
        if (substring_hashes_in_a.find(substring_hash_in_b) !=
            substring_hashes_in_a.end()) {
            res = b.substr(i, k);
            return true;
        }
    }
    return false;
}
int32_t main() {
    prec();
    cin >> a >> b;
    A.build(a);
    B.build(b);
    int l = 1, r = min(a.size(), b.size()), ans = 0; // use min size of
strings
    while (l <= r) {
        int mid = (l + r) / 2;
        if (ok(mid)) {
            ans = mid;
            l = mid + 1;
        } else {
            r = mid - 1;
        }
    }
    ok(ans); // set res
    cout << res << '\n'; // O(n log^2 n)
    return 0;
}
```

6.5 Manacher's Algo Longest palindrome

```
string longestPalindrome(string s) {
    int n = s.size();
    if(n == 0) return "";
    string t = "^";
    for(char c : s) t += "#" + string(1,c);
    t += "#$";
    int m = t.size();
    vector<int> p(m, 0);
    int center = 0, right = 0;
    for(int i=1; i < m-1; i++){
        int mirror = 2*center - i;
        if(i < right) p[i] = min(right - i, p[mirror]);
        while(t[i + (1 + p[i])] == t[i - (1 + p[i])]) p[i]++;
        if(i + p[i] > right){
            center = i;
            right = i + p[i];
        }
    }
    int maxLen = 0, centerIndex = 0;
    for(int i=1; i<m-1; i++){
        if(p[i] > maxLen){
            maxLen = p[i];
            centerIndex = i;
        }
    }
    int start = (centerIndex - maxLen)/2;
    return s.substr(start, maxLen);
}
```

6.6 Pattern Matching

```
const int N = 1e6 + 9,p1 = 137, mod1 = 127657753,p2 = 277, mod2 = 987654319;
int power(long long n, long long k, int mod) {
    int ans = 1 % mod;
    n %= mod;
    if (n < 0) n += mod;
    while (k) {
        if (k & 1) ans = (long long) ans * n % mod;
        n = (long long) n * n % mod;
        k >>= 1;
    }
    return ans;}
int ip1, ip2; pair<int,int> pw[N], ipw[N];
void prec() {
    pw[0] = {1, 1};
    for (int i = 1; i < N; i++) {
        pw[i].first = 1LL * pw[i-1].first * p1 % mod1;
        pw[i].second = 1LL * pw[i-1].second * p2 % mod2;
    }
    ip1 = power(p1, mod1 - 2, mod1);
    ip2 = power(p2, mod2 - 2, mod2);
    ipw[0] = {1, 1};
    for (int i = 1; i < N; i++) {
        ipw[i].first = 1LL * ipw[i-1].first * ip1 % mod1;
        ipw[i].second = 1LL * ipw[i-1].second * ip2 % mod2;
    }
}
pair<int,int> string_hash(const string &s) {
    int n = s.size();
    pair<int,int> hs({0, 0});
    for (int i = 0; i < n; i++) {
        hs.first += 1LL * s[i] * pw[i].first % mod1;
        hs.first %= mod1;
        hs.second += 1LL * s[i] * pw[i].second % mod2;
        hs.second %= mod2;
    }
    return hs;}
pair<int,int> pref[N];
void build(const string &s) {
    int n = s.size();
    for (int i = 0; i < n; i++) {
        pref[i].first = 1LL * s[i] * pw[i].first % mod1;
        if (i) pref[i].first = (pref[i].first + pref[i-1].first) % mod1;
        pref[i].second = 1LL * s[i] * pw[i].second % mod2;
        if (i) pref[i].second = (pref[i].second + pref[i-1].second) % mod2;
    }
}
pair<int,int> get_hash(int i, int j) {
    assert(i <= j);
    pair<int,int> hs({0, 0});
    hs.first = pref[j].first;
    if (i) hs.first = (hs.first - pref[i-1].first + mod1) % mod1;
    hs.first = 1LL * hs.first * ipw[i].first % mod1;
    hs.second = pref[j].second;
    if (i) hs.second = (hs.second - pref[i-1].second + mod2) % mod2;
    hs.second = 1LL * hs.second * ipw[i].second % mod2;
    return hs;
}
int32_t main() {
    prec();
    string a, b; cin >> a >> b; build(a);
    int ans = 0; int n = a.size(), m = b.size();
    auto hash_b = string_hash(b);
    for (int i = 0; i + m - 1 < n; i++) {
        if (get_hash(i, i + m - 1) == hash_b) ans++;}
    cout << ans << '\n';
}
/*
Input :
banana
ana
Output :
2
*/
```

6.7 Generate Number

```
#include <bits/stdc++.h>
using namespace std;
vector<long long> pattern_numbers;
void generate(long long num, const vector<int> &digits, long long limit) {
    if (num > limit) return;
    if (num != 0) pattern_numbers.push_back(num);
    for (int d : digits) {
        generate(num * 10 + d, digits, limit);
    }
}
int count_in_range(long long l, long long r) {
    return upper_bound(pattern_numbers.begin(), pattern_numbers.end(), r) -
           lower_bound(pattern_numbers.begin(), pattern_numbers.end(), l);
}
int main() {
    vector<int> digits = {4, 7};  // Nice numbers
    long long limit = 1e18;
    generate(0, digits, limit);
    sort(pattern_numbers.begin(), pattern_numbers.end());
    int t;
    cin >> t;
    while (t--) {
        long long l, r;
        cin >> l >> r;
        cout << count_in_range(l, r) << "\n";
    }
    return 0;
}
#include <bits/stdc++.h>
using namespace std;
vector<int> digits;
long long l, r;
long long dp[20][2][2];
long long solve(const string &num, int pos, bool tight, bool started) {
    if (pos == num.size()) return started;
    long long &res = dp[pos][tight][started];
    if (res != -1) return res;
    res = 0;
    int limit = tight ? (num[pos] - '0') : 9;
    for (int d : digits) {
        if (!started && d == 0) continue;
        if (d > limit) continue;
        res += solve(num, pos + 1, tight && (d == limit), true);
    }
    if (!started) res += solve(num, pos + 1, false, false);
    return res;
}
long long count(long long x) {
    string s = to_string(x);
    memset(dp, -1, sizeof(dp));
    return solve(s, 0, true, false);
}
int main() {
    int t;
    cin >> t;
    digits = {7, 9};
    while (t--) {
        cin >> l >> r;
        long long ans = count(r) - count(l - 1);
        cout << ans << "\n";
    }
    return 0;
}
```

7 geometry

7.1 Basics

```
#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", "
typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector<vi> vii;
const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9 + 7;
const int N = 1e5 + 5;
typedef long double type;
bool ge(type x, type y) { return x + EPS > y; }
bool le(type x, type y) { return x - EPS < y; }
bool eq(type x, type y) { return ge(x, y) && le(x, y); }
int sign(type x) { return ge(x, 0) - le(x, 0); }
struct point {
    type x, y;
    point() : x(0), y(0) {}
    point(type _x, type _y) : x(_x), y(_y) {}
    point operator -() { return point(-x, -y); }
    point operator +(point p) { return point(x + p.x, y + p.y); }
    point operator -(point p) { return point(x - p.x, y - p.y); }
    point operator *(type k) { return point(x * k, y * k); }
    point operator /(type k) { return point(x / k, y / k); }
    // inner product
    type operator *(point p) { return x * p.x + y * p.y; }
    // cross product
    type operator %(point p) { return x * p.y - y * p.x; }
    bool operator ==(const point &p) const {
        return x == p.x && y == p.y;
    }
    bool operator !=(const point &p) const {
        return x != p.x || y != p.y;
    }
    bool operator <(const point &p) const {
        return (x < p.x) || (x == p.x && y < p.y);
    }
    // 0 => same direction, 1 => p is on the left, -1 => p is on the right
    int dir(point o, point p) {
        type v = (*this - o) % (p - o);
        return ge(v, 0) - le(v, 0);
    }
    bool on_seg(point p, point q) {
        if (this->dir(p, q)) return false;
        return ge(x, min(p.x, q.x)) && le(x, max(p.x, q.x)) &&
               ge(y, min(p.y, q.y)) && le(y, max(p.y, q.y));
    }
    ld abs() { return sqrt(x * x + y * y); }
    type abs2() { return x * x + y * y; }
    ld dist(point q) { return (*this - q).abs(); }
    type dist2(point q) { return (*this - q).abs2(); }
    ld arg() { return atan2l(y, x); }
    // Project point on vector y
    point project(point y) {
        return y * ((*this * y) / (y * y));
    }
    // Project point on line generated by points x and y
    point project(point x, point y) {
        return x + (*this - x).project(y - x);
    }
    ld dist_line(point x, point y) {
        return dist(project(x, y));
    }
    ld dist_seg(point x, point y) {
        return project(x, y).on_seg(x, y) ? dist_line(x, y) : min(dist(x),
dist(y));
    }
    point rotate(ld sinv, ld cosv) {
        return point(cosv * x - sinv * y, sinv * x + cosv * y);
    }
    point rotate(ld a) {
        return rotate(sin(a), cos(a));
    }
    // rotate around the argument of vector p
    point rotate(point p) {
        return rotate(p.y / p.abs(), p.x / p.abs());
    }
};
int direction(point o, point p, point q) {
    return p.dir(o, q);
}
point rotate_ccw90(point p) { return point(-p.y, p.x); }
point rotate_cw90(point p) { return point(p.y, -p.x); }
// for reading purposes avoid using * and % operators
type dot(point p, point q) { return p.x * q.x + p.y * q.y; }
type cross(point p, point q) { return p.x * q.y - p.y * q.x; }
// double area
type area_2(point a, point b, point c) {
    return cross(a, b) + cross(b, c) + cross(c, a);
}
// angle comparison-- 1 : bigger, -1 : smaller, 0 : equal
int angle_less(const point& a1, const point& b1, const point& a2, const
point& b2) {
    point p1(dot(a1, b1), abs(cross(a1, b1)));
    point p2(dot(a2, b2), abs(cross(a2, b2)));
    if (cross(p1, p2) < 0) return 1;
    if (cross(p1, p2) > 0) return -1;
    return 0;
}
ostream& operator<<(ostream& os, const point& p) {
    os << "(" << p.x << "," << p.y << ")";
    return os;
}
```

7.2 Circle

```
#include "basics.cpp"
#include "lines.cpp"
struct circle {
    point c; ld r;
    circle() {
        c = point(); r = 0;
    }
    circle(point _c, ld _r) : c(_c), r(_r) {}
    ld area() {
        return acos(-1.0) * r * r;
    }
    ld chord(ld rad) {
        return 2 * r * sin(rad / 2.0);
    }
    ld sector(ld rad) {
        return 0.5 * rad * area() / acos(-1.0);
    }
    bool intersects(circle other) {
        return le(c.dist(other.c), r + other.r);
    }
    bool contains(point p) {
        return le(c.dist(p), r);
    }
    pair<point, point> getTangentPoint(point p) {
        ld d1 = c.dist(p);
        ld theta = asin(r / d1);
        point p1 = (c - p).rotate(-theta);
        point p2 = (c - p).rotate(theta);
        p1 = p1 * (sqrt(d1 * d1 - r * r) / d1) + p;
        p2 = p2 * (sqrt(d1 * d1 - r * r) / d1) + p;
        return make_pair(p1, p2);
    }
};
circle circumcircle(point a, point b, point c) {
    circle ans;
    point u = point((b - a).y, -(b - a).x);
    point v = point((c - a).y, -(c - a).x);
    point n = (c - b) * 0.5;
    ld t = cross(u, n) / cross(v, u);
    ans.c = ((a + c) * 0.5) + (v * t);
    ans.r = ans.c.dist(a);
    return ans;
}
point compute_circle_center(point a, point b, point c) {
    // circumcenter
    b = (a + b) / 2;
    c = (a + c) / 2;
    return compute_line_intersection(
        b, b + rotate_cw90(a - b),
        c, c + rotate_cw90(a - c)
    );
}
int inside_circle(point p, circle c) {
    if (fabs(p.dist(c.c) - c.r) < EPS) return 1;
    else if (p.dist(c.c) < c.r) return 0;
    else return 2;
} // 0 = inside / 1 = border / 2 = outside
circle incircle(point p1, point p2, point p3) {
    ld m1 = p2.dist(p3);
    ld m2 = p1.dist(p3);
    ld m3 = p1.dist(p2);
    point c = (p1 * m1 + p2 * m2 + p3 * m3) * (1 / (m1 + m2 + m3));
    ld s = 0.5 * (m1 + m2 + m3);
    ld r = sqrt(s * (s - m1) * (s - m2) * (s - m3)) / s;
    return circle(c, r);
}
circle minimum_circle(vector<point> p) {
    random_shuffle(p.begin(), p.end());
    circle C = circle(p[0], 0.0);
    for (int i = 0; i < (int)p.size(); i++) {
        if (C.contains(p[i])) continue;
        C = circle(p[i], 0.0);
        for (int j = 0; j < i; j++) {
            if (C.contains(p[j])) continue;
            C = circle((p[j] + p[i]) * 0.5, 0.5 * p[j].dist(p[i]));
            for (int k = 0; k < j; k++) {
                if (C.contains(p[k])) continue;
                C = circumcircle(p[j], p[i], p[k]);
            }
        }
    }
    return C;
}
```

```cpp
// compute intersection of line through points a and b
// with circle centered at c with radius r > 0
vector<point> circle_line_intersection(point a, point b, point c, ld r) {
    vector<point> ret;
    b = b - a;
    a = a - c;
    ld A = dot(b, b);
    ld B = dot(a, b);
    ld C = dot(a, a) - r * r;
    ld D = B * B - A * C;
    if (D < -EPS) return ret;
    ret.push_back(c + a + b * (sqrt(D + EPS) - B) / A);
    if (D > EPS) ret.push_back(c + a + b * (-B - sqrt(D)) / A);
    return ret;
}
vector<point> circle_circle_intersection(point a, point b, ld r, ld R) {
    vector<point> ret;
    ld d = sqrt(a.dist2(b));
    if (d > r + R || d + min(r, R) < max(r, R)) return ret;
    ld x = (d * d - R * R + r * r) / (2 * d);
    ld y = sqrt(r * r - x * x);
    point v = (b - a) / d;
    ret.push_back(a + v * x + rotate_ccw90(v) * y);
    if (y > 0) ret.push_back(a + v * x - rotate_ccw90(v) * y);
    return ret;
}
// GREAT CIRCLE
double gcTheta(double pLat, double pLong,double qLat, double qLong) {
    pLat  *= acos(-1.0) / 180.0;
    pLong *= acos(-1.0) / 180.0;
    qLat  *= acos(-1.0) / 180.0;
    qLong *= acos(-1.0) / 180.0;
    return acos(
        cos(pLat) * cos(pLong) * cos(qLat) * cos(qLong) +
        cos(pLat) * sin(pLong) * cos(qLat) * sin(qLong) +
        sin(pLat) * sin(qLat)
    );
}
double gcDistance(double pLat, double pLong,double qLat, double qLong,double
radius) {
    return radius * gcTheta(pLat, pLong, qLat, qLong);
}
```

7.3 Lines

```cpp
#include "basics.cpp"
// WARNING: all distance functions are not realizing sqrt operation
// Suggestion: for line intersections check line_line_intersection
// and then use compute_line_intersection
point project_point_line(point c, point a, point b) {
    ld r = dot(b - a, b - a);
    if (fabs(r) < EPS) return a;
    return a + (b - a) * dot(c - a, b - a) / dot(b - a, b - a);
}
point project_point_ray(point c, point a, point b) {
    ld r = dot(b - a, b - a);
    if (fabs(r) < EPS) return a;
    r = dot(c - a, b - a) / r;
    if (le(r, 0)) return a;
    return a + (b - a) * r;
}
point project_point_segment(point c, point a, point b) {
    ld r = dot(b - a, b - a);
    if (fabs(r) < EPS) return a;
    r = dot(c - a, b - a) / r;
    if (le(r, 0)) return a;
    if (ge(r, 1)) return b;
    return a + (b - a) * r;
}
ld distance_point_line(point c, point a, point b) {
    return c.dist2(project_point_line(c, a, b));
}
ld distance_point_ray(point c, point a, point b) {
    return c.dist2(project_point_ray(c, a, b));
}
ld distance_point_segment(point c, point a, point b) {
    return c.dist2(project_point_segment(c, a, b));
}
// not tested
ld distance_point_plane(ld x, ld y, ld z, ld a, ld b, ld c, ld d) {
    return fabs(a * x + b * y + c * z - d) / sqrt(a * a + b * b + c * c);
}
bool lines_parallel(point a, point b, point c, point d) {
    return fabs(cross(b - a, d - c)) < EPS;
}
bool lines_collinear(point a, point b, point c, point d) {
    return lines_parallel(a, b, c, d) && fabs(cross(a - b, a - c)) < EPS &&
           fabs(cross(c - d, c - a)) < EPS;
}
point lines_intersect(point p, point q, point a, point b) {
    point r = q - p;
    point s = b - a;
    point c(p % q, a % b);
    if (eq(r % s, 0)) return point(LINF, LINF);
    return point(
        point(r.x, s.x) % c,
        point(r.y, s.y) % c
    ) / (r % s);
}
// be careful: test line_line_intersection before using this function
point compute_line_intersection(point a, point b, point c, point d) {
    b = b - a;
    d = c - d;
    c = c - a;
    assert(dot(b, b) > EPS && dot(d, d) > EPS);
    return a + b * cross(c, d) / cross(b, d);
}
bool line_line_intersect(point a, point b, point c, point d) {
    if (!lines_parallel(a, b, c, d)) return true;
    if (lines_collinear(a, b, c, d)) return true;
    return false;
}
// rays in direction a -> b, c -> d
bool ray_ray_intersect(point a, point b, point c, point d) {
    if (a.dist2(c) < EPS || a.dist2(d) < EPS ||
        b.dist2(c) < EPS || b.dist2(d) < EPS)
        return true;
    if (lines_collinear(a, b, c, d)) {
        if (ge(dot(b - a, d - c), 0)) return true;
        if (ge(dot(a - c, d - c), 0)) return true;
        return false;
    }
    if (!line_line_intersect(a, b, c, d)) return false;
    point inters = lines_intersect(a, b, c, d);
    if (ge(dot(inters - c, d - c), 0) && ge(dot(inters - a, b - a), 0))
        return true;
    return false;
}
bool segment_segment_intersect(point a, point b, point c, point d) {
    if (a.dist2(c) < EPS || a.dist2(d) < EPS ||
        b.dist2(c) < EPS || b.dist2(d) < EPS)
        return true;
    int d1 = direction(a, b, c);
    int d2 = direction(a, b, d);
    int d3 = direction(c, d, a);
    int d4 = direction(c, d, b);
    if (d1 * d2 < 0 && d3 * d4 < 0) return true;
    return a.on_seg(c, d) || b.on_seg(c, d) ||
           c.on_seg(a, b) || d.on_seg(a, b);
}
bool segment_line_intersect(point a, point b, point c, point d) {
 if (!line_line_intersect(a, b, c, d)) return false;
 point inters = lines_intersect(a, b, c, d);
 return inters.on_seg(a, b);
}
// ray in direction c -> d
bool segment_ray_intersect(point a, point b, point c, point d) {
 if (a.dist2(c) < EPS || a.dist2(d) < EPS ||
 b.dist2(c) < EPS || b.dist2(d) < EPS)
 return true;
 if (lines_collinear(a, b, c, d)) {
 if (c.on_seg(a, b)) return true;
 if (ge(dot(d - c, a - c), 0)) return true;
 return false;
 }
 if (!line_line_intersect(a, b, c, d)) return false;
 point inters = lines_intersect(a, b, c, d);
 if (!inters.on_seg(a, b)) return false;
 if (ge(dot(inters - c, d - c), 0)) return true;
 return false;
}
// ray in direction a -> b
bool ray_line_intersect(point a, point b, point c, point d) {
 if (a.dist2(c) < EPS || a.dist2(d) < EPS ||
 b.dist2(c) < EPS || b.dist2(d) < EPS)
 return true;
 if (!line_line_intersect(a, b, c, d)) return false;
 point inters = lines_intersect(a, b, c, d);
 if (ge(dot(inters - a, b - a), 0)) return true;
 return false;
}
ld distance_segment_line(point a, point b, point c, point d) {
 if (segment_line_intersect(a, b, c, d)) return 0;
 return min(distance_point_line(a, c, d), distance_point_line(b, c, d));
}
ld distance_segment_ray(point a, point b, point c, point d) {
 if (segment_ray_intersect(a, b, c, d)) return 0;
 ld min1 = distance_point_segment(c, a, b);
 ld min2 = min(distance_point_ray(a, c, d), distance_point_ray(b, c, d));
 return min(min1, min2);
}
ld distance_segment_segment(point a, point b, point c, point d) {
 if (segment_segment_intersect(a, b, c, d)) return 0;
 ld min1= min(distance_point_segment(c,a,b),distance_point_segment(d,a, b));
 ld min2= min(distance_point_segment(a,c,d),distance_point_segment(b,c, d));
 return min(min1, min2);
}
ld distance_ray_line(point a, point b, point c, point d) {
 if (ray_line_intersect(a, b, c, d)) return 0;
 return distance_point_line(a, c, d);
}
ld distance_ray_ray(point a, point b, point c, point d) {
 if (ray_ray_intersect(a, b, c, d)) return 0;
 return min(distance_point_ray(c, a, b), distance_point_ray(a, c, d));
}
ld distance_line_line(point a, point b, point c, point d) {
 if (line_line_intersect(a, b, c, d)) return 0;
 return distance_point_line(a, c, d);
}
```

7.4 Polygons

```cpp
#include <bits/stdc++.h>
using namespace std;
using ld=long double;
using ll=long long;
struct P{
    ll x,y;
    P(ll a=0,ll b=0):x(a),y(b){}
    P operator+(const P&p)const{return {x+p.x,y+p.y};}
    P operator-(const P&p)const{return {x-p.x,y-p.y};}
    P operator*(ll k)const{return {x*k,y*k};}
    P operator/(ll k)const{return {x/k,y/k};}
    bool operator<(const P&p)const{return x==p.x?y<p.y:x<p.x;}
    ll dist2(const P&p)const{return (x-p.x)*(x-p.x)+(y-p.y)*(y-p.y);}
    ll operator*(const P&p)const{return x*p.x+y*p.y;} // dot
    ll operator%(const P&p)const{return x*p.y-y*p.x;} // cross
};
// Cross product helper
ll crs(const P &a,const P &b,const P &c){return (b-a)%(c-a);}
// Convex hull - Monotone Chain
vector<P> chull(vector<P> pts){
    sort(pts.begin(),pts.end());
    pts.erase(unique(pts.begin(),pts.end()),pts.end());
    vector<P> up,dn;
    for(auto &p:pts){
        while(up.size()>1 && crs(up[up.size()-2],up.back(),p)>0)
up.pop_back();
        while(dn.size()>1 && crs(dn[dn.size()-2],dn.back(),p)<0)
dn.pop_back();
        up.push_back(p); dn.push_back(p);
    }
    for(int i=(int)up.size()-2;i>=1;i--) dn.push_back(up[i]);
    return dn;
}
// Polygon area
ld poly_area(const vector<P>&p){
    ll a=0;
    for(int i=0;i<p.size();i++){
        int j=(i+1)%p.size();
        a+=p[i].x*p[j].y - p[j].x*p[i].y;
    }
    return abs(a)/2.0;
}
// Polygon perimeter
ld poly_perim(const vector<P>&p){
    ld s=0;
    for(int i=0;i<p.size();i++){
        int j=(i+1)%p.size();
        s+=sqrt((ld)p[i].dist2(p[j]));
    }
    return s;
}
// Polygon centroid
P poly_cent(const vector<P>&p){
    ll A=0,Cx=0,Cy=0;
    for(int i=0;i<p.size();i++){
        int j=(i+1)%p.size();
        ll c=p[i]%p[j];
        A+=c; Cx+=(p[i].x+p[j].x)*c; Cy+=(p[i].y+p[j].y)*c;
    }
    A*=0.5;
    return {Cx/(3*A),Cy/(3*A)};
}
```

```cpp
// Point in convex polygon (O(log n))
bool pin_conv(const vector<P>&h,P q){
 int n=h.size();
 if((q-h[0])%(h[1]-h[0])<0) return 0;
 if((q-h[0])%(h[n-1]-h[0])>0) return 0;
 int l = 1,r = n-1;
 while(r-l>1){
 int m = (l+r)/2;
 if((q-h[0])%(h[m]-h[0])>=0) l=m; else r=m;
 }
 ll t = abs((h[l]-q)%(h[l+1]-q)+(h[l+1]-q)%(h[0]-q)+(h[0]-q)%(h[l]-q));
 ll s = abs((h[l]-h[l+1])%(h[l]-h[0]));
 return t == s;
}
// Max scalar product on convex hull
int max_sc(const vector<P>&h,P v){
 int n = h.size(),r = 0;
 for(int i=0;i<n;i++) if(h[i]*v > h[r]*v) r = i;
 return r;
}
// Min and Max helpers
template<typename T> T cmin(T &a,const T &b){if(b < a) a = b; return a;}
template<typename T> T cmax(T &a,const T &b){if(b > a) a = b; return a;}
int main(){
 vector<P> pts = {{0,0},{2,0},{1,1},{0,2},{2,2}};
 auto h = chull(pts);
 cout<<"Convex Hull Points:\n";
 for(auto &p:h) cout << "("<< p.x << "," << p.y << ") ";
 cout << "\nArea: " << poly_area(h) << "\n";
 cout << "Perimeter: "<<poly_perim(h) << "\n";
 auto c = poly_cent(h);
 cout<<"Centroid: (" << c.x << "," << c.y << ")\n";
 P tp = {1,1};
 cout << "Point (1,1) inside convex hull? " << (pin_conv(h,tp)?"Yes":"No")
<< "\n";
 P v = {1,2};
 cout << "Max scalar index along vector (1,2): " << max_sc(h,v) << "\n";
}
```

7.5 Radial Sort

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
struct point {
    ll x, y;
    point(ll a=0,ll b=0):x(a),y(b){}
    point operator-(const point &p) const { return {x-p.x,y-p.y}; }
    ll operator%(const point &p) const { return x*p.y - y*p.x; }
    ll dir(const point &o,const point &q) const {
        return ( *this - o ) % ( q - o );
    }
};
point origin;
int above(point p){
    if(p.y == origin.y) return p.x > origin.x;
    return p.y > origin.y;
}
bool cmp(point p, point q){
    int tmp = above(q) - above(p);
    if(tmp) return tmp>0;
    return p.dir(origin,q) > 0;
}
int main(){
    vector<point> pts = {{1,1},{2,2},{3,0},{0,3},{1,2}};
    origin = {0,0};
    sort(pts.begin(), pts.end(), cmp);
    cout << "Points sorted by angle around origin:\n";
    for(auto &p: pts) cout << "(" << p.x << "," << p.y << ") ";
}
```

8 Extra Math Formula & Technique

Math Formula

Summation from a to b:
Σ a→b = ((b-a+1)*(a+b))/2

Sum of ai*aj for 1 ≤ i ≤ j ≤ n:
aiaj = ((Σ ai)^2 - Σ ai^2)/2
Example array [1,2,3,4,5]:
Σ squares = 1^2 + 2^2 + 3^2 + 4^2 + 5^2
Σ elements = 1 + 2 + 3 + 4 + 5
Σ ai*aj = ((Σ elements)^2 - Σ squares)/2

Arithmetic Progression (AP):
nth term tn = a + (n-1)*d
Sum of first n terms Sn = n/2 * (2a + (n-1)*d)
Sum using first and nth term: Sn = n*(a + tn)/2

Geometric Progression (GP):
nth term tn = a * r^(n-1)
Sum of first n terms (r ≠ 1): Sn = a*(r^n - 1)/(r - 1)
Sum of first n terms (r = 1): Sn = a*n

Sum of first n natural numbers:
1+2+3+...+n = n*(n+1)/2
Sum of first n squares:
1^2+2^2+...+n^2 = n*(n+1)*(2n+1)/6
Sum of first n cubes:
1^3+2^3+...+n^3 = (n*(n+1)/2)^2
Sum of first n odd numbers:
1+3+5+...+(2n-1) = n^2
Sum of first n even numbers:
2+4+6+...+2n = n*(n+1)
Sum of consecutive numbers in range i to j:
i + (i+1) + ... + j = ((j-i+1)*(i+j))/2
Pair sum shortcut for range i..j:
Σ ai*aj = ((Σ ai)^2 - Σ ai^2)/2
Difference of sums formula:
Σ 1→n - Σ 1→m = (n*(n+1)/2) - (m*(m+1)/2)

Binomial Tricks

1. Weighted sum of binomial coefficients:
0*C(n,0) + 1*C(n,1) + 2*C(n,2) + ... + n*C(n,n) = n*2^(n-1)
2. Sum over rows for fixed r:
0Cr + 1Cr + 2Cr + ... + nCr = C(n+1, r+1)
3. Sum of squares of binomial coefficients:
(C(n,0))^2 + (C(n,1))^2 + ... + (C(n,n))^2 = C(2*n, n)
4. Sum of all binomial coefficients:
C(n,0) + C(n,1) + ... + C(n,n) = 2^n
5. Alternating sum of binomial coefficients:
C(n,0) - C(n,1) + C(n,2) - ... + (-1)^n * C(n,n) = 0
6. Hockey-stick identity:
C(r,r) + C(r+1,r) + C(r+2,r) + ... + C(n,r) = C(n+1, r+1)
7. Sum of even-indexed coefficients:
C(n,0) + C(n,2) + C(n,4) + ... = 2^(n-1)
8. Sum of odd-indexed coefficients:
C(n,1) + C(n,3) + C(n,5) + ... = 2^(n-1)
9. Vandermonde's identity:
C(m+n, r) = C(m,0)*C(n,r) + C(m,1)*C(n,r-1) + ... + C(m,r)*C(n,0)
10. Sum of k*C(n,k):
Σ (k*C(n,k)) from k=0 to n = n*2^(n-1)
11. Sum of k^2*C(n,k):
Σ (k^2*C(n,k)) from k=0 to n = n*(n+1)*2^(n-2)

Geometry Theorem

1. Pythagoras Theorem:
In right triangle ΔABC with right angle at C:
AB^2 = AC^2 + BC^2
2. Triangle Midpoint Theorem:
Line joining midpoints of two sides is parallel to third side and half its length.
3. Similar Triangles Theorem:
Corresponding angles equal; corresponding sides proportional.
4. Triangle Inequality Theorem:
Sum of any two sides > third side; difference of any two sides < third side.
5. Exterior Angle Theorem:
Exterior angle = sum of two opposite interior angles
6. Circle Theorems:
- Angle subtended by diameter = 90°
- Angle at center = 2 * angle at circumference
- Tangent perpendicular to radius at point of contact
- Tangent segments from external point equal
7. Quadrilateral Theorems:
- Sum of interior angles = (n-2)*180°, for n-sided polygon
- Opposite angles of a parallelogram are equal
- Diagonals of rectangle are equal
- Diagonals of rhombus bisect each other at right angles
- Diagonals of square are equal and perpendicular bisectors of each other
8. Midpoint / Centroid Theorem:
Centroid divides medians in 2:1 ratio from vertex
9. Orientation / Collinearity Theorem:
Points a, b, c collinear if orientation = 0; otherwise clockwise or counterclockwise.
10. Basic Polygon Theorems:
- Sum of interior angles = (n-2)*180°
- Sum of exterior angles = 360°
- Each exterior angle of regular polygon = 360°/n
11. Diagonal Theorems:
- Number of diagonals in n-sided polygon = n(n-3)/2
- Sum of lengths of diagonals depends on shape
12. Tangent / Secant Theorems:
- Two tangents from external point are equal
- Angle between tangent and chord = angle in alternate segment
13. Chord Theorems:
- Perpendicular from center to chord bisects chord
- Equal chords equidistant from center
14. Concurrency Theorems:
- Medians meet at centroid
- Angle bisectors meet at incenter
- Perpendicular bisectors meet at circumcenter
- Altitudes meet at orthocenter
15. Basic 3D Geometry Theorems (Higher Math):
- Line joining midpoints of opposite edges of a cube=diagonalof face/sqrt(2)
- Plane containing three points is uniquely determined
- Diagonal of cube = sqrt(3)*edge

Geometry Formulas

1. Triangle
   Given: base b, height h, sides a,b,c
   Semiperimeter: s = (a+b+c)/2
   Area = (1/2)*b*h
   Perimeter = a+b+c
   Heron Formula Area = √[s*(s-a)*(s-b)*(s-c)]

2. Square
   Given: side s
   Area = s^2
   Perimeter = 4s
   Diagonal = s√2

3. Rectangle
   Given: length l, width w
   Area = l*w
   Perimeter = 2(l + w)
   Diagonal = √(l^2 + w^2)

4. Trapezoid (Trapezium)
   Given: bases a,b, height h
   Area = (1/2)*(a+b)*h
   Midsegment = (a+b)/2
   Perimeter = a+b+c+d

5. Parallelogram
   Given: base b, height h, sides a,b
   Area = b*h
   Perimeter = 2(a+b)
   Diagonal formulas:
     d1 = √(a^2 + b^2 + 2ab*cosθ)
     d2 = √(a^2 + b^2 - 2ab*cosθ)

6. Circle
   Given: radius r
   Area = π*r^2
   Circumference = 2*π*r
   Diameter = 2r
   Inner circle radius: r = area / s
   Outer circle area: A = (a*b*c) / (4*R)

7. Sphere
   Given: radius r
   Surface Area = 4*π*r^2
   Volume = (4/3)*π*r^3

8. Cylinder
   Given: radius r, height h
   Lateral Surface Area = 2*π*r*h
   Total Surface Area = 2*π*r*(h + r)
   Volume = π*r^2*h

9. Cone
   Given: radius r, height h, slant height l = √(r^2 + h^2)
   Lateral Surface Area = π*r*l
   Total Surface Area = π*r*(r + l)
   Volume = (1/3)*π*r^2*h

10. Pyramid
    Given: base area B, height h, slant height l
    Volume = (1/3)*B*h
    Lateral Surface Area (square base) = 2*a*l
    Total Surface Area (square base) = a^2 + 2*a*l

11. Rectangular Prism (Cuboid)
    Given: length l, width w, height h
    Surface Area = 2*(lw + lh + wh)
    Volume = l*w*h
    Diagonal = √(l^2 + w^2 + h^2)

12. Cube
    Given: side s
    Surface Area = 6*s^2
    Volume = s^3
    Diagonal = s√3

13. N-POINT POLYGON REGIONS
    R = E - V + 2
    V = n + nC4
    E = (n*(n-1) + 4*nC4) / 2

Distance, Dot, Cross, Orientation:
- Distance of two points: d = √[(x2 - x1)^2 + (y2 - y1)^2]
- Dot product: a·b = ax*bx + ay*by
- Cross product (2D): a × b = ax*by - ay*bx
- Orientation: orient(a,b,c) = (b-a) × (c-a)

Line through (x1,y1),(x2,y2):
- Ax + By + C = 0
 A = y1 - y2
 B = x2 - x1
 C = x1*y2 - x2*y1
- Distance from point (x0,y0) to line: d = |A*x0 + B*y0 + C| / √(A^2 + B^2)

Polygon Formulas:
- Area = (1/2) * Σ (xi*yi+1 - xi+1*yi)
- Perimeter = Σ distance(pi, pi+1)

Circle Geometry:
- Chord length = 2*√(r^2 - d^2)
- Sector area = (1/2)*r^2*θ
- Arc length: s = r*θ
- Triangle area using circumradius: A = (a*b*c)/(4*R)

Vectors:
- Angle between vectors: θ = cos^(-1)((a·b)/(|a||b|))
- Projection: proj_b(a) = (a·b)/|b|^2 * b

Matrices and Determinants

1. The number of rows or columns (Order) in a square matrix is called its order.
2. A matrix whose all elements are zero is called a null matrix.
3. Matrix multiplication AB is possible only when the number of columns in matrix A is equal to the number of rows in matrix B.
4. If A is a non-singular matrix, then the inverse is given by: A^(-1) = (1/|A|) * Adj(A)
5. Adjoint of A: Adj(A) = (Cofactor Matrix)^T
6. Determinant minors and cofactors (for 3x3 D):
   D = | a11 a12 a13 |
       | a21 a22 a23 |
       | a31 a32 a33 |
   Minor of a11: m11 = a22*a33 - a23*a32
   Cofactor of aij: Aij = (-1)^(i+j) * mij
   Example: A11 = a22*a33 - a23*a32, A12 = -m12
7. Important Properties of Determinants:
   i. If any two columns (or rows) are identical, its value is zero.
      | a a c |
      | b b f | = 0
      | d d g |
   ii. If all elements in any one row (or column) of a determinant are multiplied by a scalar k, the value is k times the value of the original determinant.
      | k*a k*b k*c | | a b c |
      | d   e   f | = k * | d e f |
      | g   h   i | | g h i |

   iii. The value of the determinant remains unchanged if a multiple of one row (or column) is added to another.
      | a1 b1 c1 | | a1 + m*b1 b1 c1 |
      | a2 b2 c2 | = | a2 + m*b2 b2 c2 |
      | a3 b3 c3 | | a3 + m*b3 b3 c3 |
   iv. If two adjacent rows (or columns) are interchanged, the sign of the determinant's value changes.

8. Cramer's Rule for solving simultaneous linear equations:
   Given: a1*x + b1*y + c1*z = d1
          a2*x + b2*y + c2*z = d2
          a3*x + b3*y + c3*z = d3
   D = | a1 b1 c1 |
        | a2 b2 c2 |
        | a3 b3 c3 | ≠ 0
   Dx = | d1 b1 c1 |
        | d2 b2 c2 |
        | d3 b3 c3 |
   Dy = | a1 d1 c1 |
        | a2 d2 c2 |
        | a3 d3 c3 |
   Dz = | a1 b1 d1 |
        | a2 b2 d2 |
        | a3 b3 d3 |
   Solution: x = Dx / D, y = Dy / D, z = Dz / D

Vectors

1. Magnitude of vector: If a = a1i + a2j + a3k,
   then |a| = sqrt(a1^2 + a2^2 + a3^2)
2. Position Vector: For point (x, y, z), r = xi + yj + zk
3. Vector Subtraction: If OP = x1i + y1j + z1k and OQ = x2i + y2j + z2k,
   then PQ = OQ - OP = (x2-x1)i + (y2-y1)j + (z2-z1)k
4. Section Formula:
   i. Internal Division: If R divides AB internally in ratio m:n,
      r = (m*b + n*a)/(m+n)
   ii. Midpoint: If C is the midpoint of AB, c = (a + b)/2
5. Unit vector parallel to a: a/|a|
6. Unit vector perpendicular to a and b: (a x b)/|(a x b)|
7. Unit vector perpendicular to plane a and b: (a x b)/|(a x b)|
8. Vector Addition: If a = a1i + a2j + a3k and b = b1i + b2j + b3k,
   then a + b = (a1+b1)i + (a2+b2)j + (a3+b3)k
9. Scalar Product (Dot Product): a . b = |a||b|cosθ = a1b1 + a2b2 + a3b3
10. Unit Vector Dot Products: i.i = j.j = k.k = 1; i.j = j.k = k.i = 0
11. Vector Product (Cross Product):
    i. Magnitude: |a x b| = |a||b|sinθ
    ii. Cartesian Form: a x b = | i j k|
                                |a1 a2 a3|
                                |b1 b2 b3|
    iii. Property: a x b = - (b x a)
12. Unit Vector Cross Products: i x i = 0;
    Cyclic: i x j = k; j x k = i; k x i = j
13. Condition for Perpendicularity: If a ⊥ b, then a . b = 0
14. Scalar projection of a onto b: a cosθ = (a . b)/|b|
15. Vector projection of a onto b: ((a . b)/|b|^2) * b
16. Scalar projection of b onto a: (a . b)/|a|
17. Volume of parallelepiped formed by a, b, c: |det([a, b, c])|
18. Condition for Coplanarity: If a, b, c are coplanar, then a . (b x c) = 0
19. Area Formulas:
    i. Area of parallelogram formed by a and b: |a x b| sq units
    ii. Area of triangle formed by a and b: (1/2)|a x b| sq units
20. Collinear Vectors: a, b collinear → a x b = 0
21. Coplanar Vectors: a, b, c coplanar → a . (b x c) = 0
22. Triangle Using Vectors: If AB = b - a, AC = c - a,
    Area ΔABC = (1/2)|AB x AC|
23. Parallelogram Using Vectors: Sides a, b → Area = |a x b|
24. Triangle Inequality: |a + b| ≤ |a| + |b|
25. Vector collinearity ratio: a = k*b, k ∈ R → vectors are collinear
26. Mixed Product (Scalar Triple Product) Property:
    i. a . (b x c) = determinant |a, b, c|
    ii. |a . (b x c)| = volume of parallelepiped
    iii. Permutation: a . (b x c) = b . (c x a) = c . (a x b)
    iv. Sign changes if two vectors swap: a . (b x c) = - b . (a x c)
27. Coplanar vectors check using scalar triple product: a . (b x c) = 0

Straight Lines

1. Coordinates on axes:
   - Point on x-axis: (x, 0)
   - Point on y-axis: (0, y)

2. Distance of point (x, y):
   - From x-axis: |y|
   - From y-axis: |x|

3. Cartesian (x, y) and Polar (r, θ) relation:
   - x = r*cosθ
   - y = r*sinθ
   - r = sqrt(x^2 + y^2)
   - θ = tan^(-1)(y/x)

4. Polar coordinates from Cartesian:
   i. 1st quadrant (x>0, y>0): θ = tan^(-1)(y/x)
   ii. 2nd quadrant (-x, y): θ = 180° - tan^(-1)(y/|x|)
   iii. 3rd quadrant (-x, -y): θ = 180° + tan^(-1)(|y|/|x|)
   iv. 4th quadrant (x, -y): θ = 360° - tan^(-1)(|y|/x)
   v. Alternative: θ = tan^(-1)(y/x) if x>0
      θ = π + tan^(-1)(y/x) if x<0

5. Distance between points (x1, y1) and (x2, y2):
   - d = sqrt((x2-x1)^2 + (y2-y1)^2)

6. Distance for same y (x1, b) and (x2, b):
   - d = |x2 - x1|

7. Distance for same x (a, y1) and (a, y2):
   - d = |y2 - y1|

8. Distance between Polar Points (r1, θ1) and (r2, θ2):
   - d = sqrt(r1^2 + r2^2 - 2*r1*r2*cos(θ1 - θ2))

9. Division Formula (Internal) for (x1, y1) and (x2, y2) in ratio m1:m2:
   - x = (m1*x2 + m2*x1)/(m1+m2)
   - y = (m1*y2 + m2*y1)/(m1+m2)

10. Division Formula (External) for (x1, y1) and (x2, y2) in ratio m1:m2:
    - x = (m1*x2 - m2*x1)/(m1-m2)
    - y = (m1*y2 - m2*y1)/(m1-m2)

11. Midpoint of line segment between (x1, y1) and (x2, y2):
    - M = ((x1+x2)/2, (y1+y2)/2)

12. Centroid of triangle with vertices (x1, y1), (x2, y2), (x3, y3):
    - G = ((x1+x2+x3)/3, (y1+y2+y3)/3)

13. Area of Triangle ABC:
    - Area = (1/2) * |det([[x1,y1,1],[x2,y2,1],[x3,y3,1]])| sq units

14. Area of Triangle with origin (0,0):
    - Area = (1/2) * |x1*y2 - x2*y1| sq units

15. Collinearity of points A, B, C:
    - Collinear if Area = 0
    - (1/2) * |x1(y2-y3) + x2(y3-y1) + x3(y1-y2)| = 0

16. Area of Quadrilateral ABCD:
    - Area = (1/2) * |det([[x1,y1],[x2,y2],[x3,y3],[x4,y4],[x1,y1]])|

17. Slope (m) of a line:
    i. m = tan θ (angle with x-axis)
    ii. x-axis slope: m = 0
    iii. y-axis slope: undefined
    iv. Line through (x1, y1), (x2, y2): m = (y2-y1)/(x2-x1)
    v. Line ax + by + c = 0: m = -a/b

18. Equation of x-axis: y = 0
19. Equation of y-axis: x = 0
20. Line parallel to x-axis: y = b
21. Line parallel to y-axis: x = a
22. Line passing through origin: y = m*x

23. Forms of line equation:
 i. Point-Slope: y - y1 = m*(x - x1)
 ii. Slope-Intercept: y = m*x + c
 iii. Two-Point: (y-y1)/(y2-y1) = (x-x1)/(x2-x1)
 iv. Intercept: x/a + y/b = 1
 v. General: ax + by + c = 0

24. Area of triangle formed by x/a + y/b = 1 and axes:
 - Area = (1/2) * |a*b| sq units

25. Perpendicular distance from point (x1, y1) to line ax + by + c = 0:
 - d = |a*x1 + b*y1 + c| / sqrt(a^2 + b^2)

Combinatorics

- NON-NEGATIVE INTEGERS x1 + x2 + ... + xk = n , xi >= 0
  Formula: C(n+k-1, k-1)
- POSITIVE INTEGERS x1 + x2 + ... + xk = n , xi > 0
  Formula: C(n-1, k-1)

- AT MOST ONE BALL PER BOX k balls into n boxes
  Formula: C(n, k)

- MULTIPLE BALLS ALLOWED xi >= 0
  Formula: C(n+k-1, k)

- EACH BOX HAS AT LEAST ONE BALL xi > 0
  Formula: C(k-1, n-1)

- NO TWO BALLS ADJACENT Choose k from n with no adjacency
  Formula: C(n-k+1, k)

- EXACTLY r BOXES NON-EMPTY
  Formula: C(n, r) * C(k-1, r-1)

- DISTINCT BALLS → ANY BOX
  Formula: n^k

- DISTINCT BALLS → NO EMPTY BOX Onto functions
  Formula: n! * S(k, n)

- COMBINATION WITH REPETITION Multiset of size r from n types
  Formula: C(n+r-1, r)

- PERMUTATION WITH REPETITION n objects, n1, n2, ..., nk identical
  Formula: n! / (n1! n2! ... nk!)

- MULTINOMIAL COEFFICIENT Split n into n1, n2, ..., nk
  Formula: n! / (n1! n2! ... nk!)

- TOTAL FUNCTIONS (n → k)
  Formula: k^n

- STIRLING NUMBER (2nd KIND)
  S(n, k) = S(n-1, k-1) + k*S(n-1, k)

- CIRCULAR PERMUTATION
  (n-1)!

- CIRCULAR WITH REPETITION
  (n-1)! / (n1! n2! ...)

- BINARY STRINGS WITH k ONES
  C(n, k)