IME ACM-ICPC Team Notebook

Contents

1 Template + vimrc

1.1 Template

```cpp
#include <iostream>
#include <vector>
#include <chrono>
#include <random>
using namespace std;

mt19937_64 llrand((int) chrono::steady_clock::now().time_since_epoch().count());

#define st first
#define nd second

#ifndef ONLINE_JUDGE
  #define db(x) cerr << #x << " == " << x << endl
  #define dbs(x) cerr << x << endl
  #define _ << ", " <<
#else
  #define db(x) ((void)0)
  #define dbs(x) ((void)0)
#endif

using ll = long long;
using ld = long double;

const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 1e5+5;

int main() {
  ios_base::sync_with_stdio(0), cin.tie(0);
}
```

1.2 vimrc

```vim
syntax on
set et ts=2 sw=0 sts=-1 ai nu hls cindent
set noswapfile
set mouse=a
nnoremap ; :
vnoremap ; :
noremap <c-j> 15gj
noremap <c-k> 15gk
nnoremap <s-k> i<CR><ESC>
inoremap ,. <esc>
vnoremap ,. <esc>
nnoremap ,. <esc>
```

2 Graphs

2.1 DFS

```cpp
// Depth First Search O(V+E)

const int N = 1e5 + 5;

int vis[N];
vector<int> adj[N];

void dfs(int x){
    vis[x] = 1;
    for(auto u : adj[x]) if(!vis[u]) dfs(u);
}
```

2.2 BFS

```
// Breadth First Search O(V+E)

const int N = 1e5 + 5;

int vis[N];
vector<int> adj[N];
queue<int> q;

void bfs(int x){
    vis[x] = 1;
    q.push(x);
    while(!q.empty()){
        int u = q.front(); q.pop();
        for(auto v : adj[u]) if(!vis[v]) {
          vis[v] = 1, q.push(v);
        }
    }
}
```

2.3 Zero-One-BFS

```
// 0-1 BFS - O(V+E)

const int N = 1e5 + 5;

int dist[N];
vector<pii> adj[N];
deque<pii> dq;

void zero_one_bfs (int x){
    cl(dist, 63);
    dist[x] = 0;
    dq.push_back({x, 0});
    while(!dq.empty()){
        int u = dq.front().st;
        int ud = dq.front().nd;
        dq.pop_front();
        if(dist[u] < ud) continue;
        for(auto x : adj[u]){
            int v = x.st;
            int w = x.nd;
            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]});
            }
        }
    }
}
```

2.4 Toposort

```
// Kahn - Topological Sort O(V + E)

const int N = 1e5+5;

vector<int> adj[N];
int n, in[N];

// For directed graph: in[x] == 0
// For undirected graph: in[x] <= 1

void kahn(){
  queue<int> q;
  for(int i = 1; i <= n; i++) if(!in[i]) q.push(i);

  while(q.size()){
    int u = q.front(); q.pop();
    for(auto x : adj[u]) if(in[x] and --in[x] == 0) q.push(x);
  }
}
```

2.5 MST (Kruskal)

```
// Kruskal - MST O(ElogE)

// + Union-Find
/////////////////////////////////////////////
int par[N], sz[N];

//Path Compression
int find(int a) { return a == par[a] ? a : par[a] = find(par[a]); }

//Ranking
void unite(int a, int b){
    if(find(a) == find(b)) return;
    a = find(a), b = find(b);
    if(sz[a] < sz[b]) swap(a, b);
    sz[a] += sz[b], par[b] = a;
}

//in main
for(int i = 0; i < N; i++) par[i] = i, sz[i] = 1;
/////////////////////////////////////////////

vector<piii> edges;
// dist, node1, node2

sort(edges.begin(), edges.end());
int cost = 0;
for(auto e : edges) if(find(e.nd.st) != find(e.nd.nd)){
    cost += e.st, unite(e.nd.st, e.nd.nd);
}
```

2.6 MST (Prim)

```
// Prim - MST O(ElogE)

int cost, vis[N];
vector<pii> adj[N];
priority_queue<pii> pq;

void prim(int s = 1){
    pq.push({0, s});
    while(!pq.empty()){
        int ud = -pq.top().st;
        int u = pq.top().nd;
        pq.pop();
        if(vis[u]) continue;
        vis[u] = 1;
        cost += ud;
        for(auto x : adj[u]){
            int v = x.st;
            int w = x.nd;
            if(!vis[v]) pq.push({-w, v});
        }
    }
}
```

2.7 Shortest Path (Dijkstra)

```
// Dijkstra - O((V+E)logE)

int dist[N];
vector<pii> adj[N];
priority_queue<pii> pq;

void dijkstra(int s){
    cl(dist, 63);
    dist[s] = 0;
    pq.push({0, s});
    while(!pq.empty()){
        int ud = -pq.top().st;
        int u = pq.top().nd;
        pq.pop();
        if(dist[u] < ud) continue;
        for(auto x : adj[u]){
            int v = x.st;
            int w = x.nd;
            if(dist[u] + w < dist[v]){
                dist[v] = dist[u] + w;
                pq.push({-dist[v], v});
            }
        }
    }
}
```

2.8 Shortest Path (SPFA)

```
// Shortest Path Faster Algoritm O(VE)

int n, inq[N], dist[N];
vector<pair<int, int>> adj[N];

void spfa(int s) {
  memset(dist, 63, sizeof dist);
  queue<int> q;
  dist[s] = 0;
  q.push(s), inq[s] = 1;
  while(q.size()) {
    int u = q.front(); q.pop(); inq[u] = 0;
    for(auto x : adj[u]) {
      int v = x.st;
      int w = x.nd;
      if(dist[u] + w < dist[v]) {
          dist[v] = dist[u] + w;
          if(!inq[v]) q.push(v), inq[v] = 1;
      }
    }
  }
}
```

2.9 Max Flow

```
// Dinic - O(n^2 * m)
// Max flow


const int N = 1e5 + 5;
const int INF = 0x3f3f3f3f;

struct edge { int v, c, f; };

int n, s, t, h[N], st[N];
vector<edge> edgs;
vector<int> g[N];

// directed from u to v with cap(u, v) = c
void add_edge(int u, int v, int c) {
  int k = edgs.size();
  edgs.push_back({v, c, 0});
  edgs.push_back({u, 0, 0});
  g[u].push_back(k);
  g[v].push_back(k+1);
}

int bfs() {
  memset(h, 0, sizeof h);
  h[s] = 1;
  queue<int> q;
  q.push(s);
  while(q.size()) {
    int u = q.front(); q.pop();
    for(auto i : g[u]) {
      int v = edgs[i].v;
      if(!h[v] and edgs[i].f < edgs[i].c)
        h[v] = h[u] + 1, q.push(v);
    }
  }
  return h[t];
}

int dfs(int u, int flow) {
  if(!flow or u == t) return flow;
  for(int &i = st[u]; i < g[u].size(); i++) {
    edge &dir = edgs[g[u][i]], &rev = edgs[g[u][i]^1];
    int v = dir.v;
    if(h[v] != h[u] + 1) continue;
    int inc = min(flow, dir.c - dir.f);
    inc = dfs(v, inc);
    if(inc) {
      dir.f += inc, rev.f -= inc;
      return inc;
    }
  }
  return 0;
}

int dinic() {
  int flow = 0;
  while(bfs()) {
    memset(st, 0, sizeof st);
    while(int inc = dfs(s, INF)) flow += inc;
  }
  return flow;
}
```

2.10 Min Cost Max Flow

```
// Min Cost Max Flow - O(n^2 * m^2)

struct edge { int v, f, c, w; };

vector<int> g[N];
vector<edge> edgs;
int s, t, inq[N], p[N], dist[N];

void add_edge(int u, int v, int c, int w) {
  int k = edgs.size();
  g[u].push_back(k);
  g[v].push_back(k+1);
  edgs.push_back({v, 0, c, w});
  edgs.push_back({u, 0, 0, -w});
}

int spfa() {
  memset(dist, 63, sizeof dist);
  queue<int> q;
  dist[s] = 0;
  q.push(s), inq[s] = 1;
  while(q.size()) {
    int u = q.front(); q.pop(); inq[u] = 0;
    for(auto i : g[u]) {
      edge dir = edgs[i];
      int v = dir.v;
      int w = dir.w;
      if(dir.f < dir.c and dist[u] + w < dist[v]) {
        dist[v] = dist[u] + w;
        p[v] = i;
        if(!inq[v]) q.push(v), inq[v] = 1;
      }
    }
  }

  if(dist[t] == INF) return 0;

  int inc = INF;
  for(int u = t; u != s; u = edgs[p[u]^1].v) {
    edge &dir = edgs[p[u]];
    inc = min(inc, dir.c - dir.f);
  }

  int aux = 0;
  for(int u = t; u != s; u = edgs[p[u]^1].v) {
    edge &dir = edgs[p[u]], &rev = edgs[p[u]^1];
    dir.f += inc;
    rev.f -= inc;
    aux += inc*dir.w;
  }

  return aux;
}

int mcmf() {
  int cost = 0;
  while(int inc = spfa()) cost += inc;
  return cost;
}
```

2.11 Max Bipartite Cardinality Matching (Kuhn)

```
// Khun (Maximum Bipartite Matching) - O(VE)

int n, cnt, vis[N], match[N], ans;
vector<int> adj[N];

bool find(int u){
  if(vis[u] == cnt) return false;
  vis[u] = cnt;
  for(auto v : adj[u]) if(!match[v] or find(match[v])) return match[v] = u;
  return false;
}

// Maximum Independent Set on bipartite graph
// MIS  = V - MATCH

// Minimum Vertex Cover
// MVC = MATCH

// Minimum Path Cover on DAG
// MPC = V - MATCH

// TIP: If you don't know the sides of the bipartite graph,
// run kuhn for all nodes and match = ans/2;

// in main (only for one of the sides)
for(int i = 1; i <= n; i++) ++cnt, ans += find(i);
```

2.12 Lowest Common Ancestor

```
// Lowest Common Ancestor - <O(nlog n), O(log n)>

const int N = 1e6;
const int M = 25; //m = log N

int anc[M][N], h[N], rt;
vector<int> adj[N];

void dfs(int x = rt, int p = -1, int ht = 0){
  anc[0][x] = p, h[x] = ht;
  for(auto v : adj[x]) if(v != p) dfs(v, x, ht+1);
}

void build(){
  dfs(), anc[0][rt] = rt;

  for(int j = 1; j < M; j++)
    for(int i = 1; i <= n; i++) // 1-indexed
      anc[j][i] = anc[j-1][anc[j-1][i]];
}

int lca(int u, int v){
  if(h[u] < h[v]) swap(u, v);
  for(int i = M-1; i >= 0; i--) if(h[u] - (1<<i) >= h[v]) u = anc[i][u];

  if(u == v) return u;
  for(int i = M-1; i >= 0; i--) if(anc[i][u] != anc[i][v])
    u = anc[i][u], v = anc[i][v];

  return anc[0][u];
}
```

2.13 2-SAT

```
// 2-SAT - O(V+E)

int n, vis[2*N], ord[2*N], ordn, cnt, cmp[2*N], val[N];
vector<int> adj[2*N], adjt[2*N];

// for a variable u with idx i
// u is 2*i and !u is 2*i+1
// (a v b) == !a -> b ^ !b -> a

int v(int x) { return 2*x; }
int nv(int x) { return 2*x+1; }

// add a -> b
void add(int a, int b) {
  adj[a].push_back(b);
  adj[b^1].push_back(a^1);
  adjt[b].push_back(a);
  adjt[a^1].push_back(b^1);
}

// add clause (a v b)
void add_or(int a, int b){
  adj[a^1].push_back(b);
  adj[b^1].push_back(a);
  adjt[b].push_back(a^1);
  adjt[a].push_back(b^1);
}

void dfs(int x){
  vis[x] = 1;
  for(auto v : adj[x]) if(!vis[v]) dfs(v);
  ord[ordn++] = x;
}

void dfst(int x){
  cmp[x] = cnt, vis[x] = 0;
  for(auto v : adjt[x]) if(vis[v]) dfst(v);
}

bool run2sat(){
  for(int i = 1; i <= n; i++) {
    if(!vis[v(i)]) dfs(v(i));
    if(!vis[nv(i)]) dfs(nv(i));
  }
  for(int i = ordn-1; i >= 0; i--)
    if(vis[ord[i]]) cnt++, dfst(ord[i]);
  for(int i = 1; i <= n; i ++){
    if(cmp[v(i)] == cmp[nv(i)]) return false;
    val[i] = cmp[v(i)] > cmp[nv(i)];
  }
  return true;
}
```

2.14 Assignment Problem

```
// Hungarian - O(n^2 * m)
template<class T, bool is_max = false, bool is_zero_indexed = false>
struct Hungarian {
  bool swap_coord = false;
  int lines, cols;
  T _INF = numeric_limits<T>::max() / 2, ans;

  vector<int> pairV, way;
  vector<bool> used;
  vector<T> pu, pv, minv;
  vector<vector<T>> cost;

  Hungarian(int _n, int _m) {
    if (_n > _m) swap(_n, _m), swap_coord = true;

    lines = _n + 1, cols = _m + 1;

    clear();
    cost.assign(lines, vector<T>(cols, _INF));
  }

  void clear() {
    pairV.assign(cols, 0);
    way.assign(cols, 0);
    pv.assign(cols, 0);
    pu.assign(lines, 0);
  }

  void update(int i, int j, T val) {
    if (is_zero_indexed) i++, j++;
    if (is_max) val = -val;
    if (swap_coord) swap(i, j);

    assert(i < lines);
    assert(j < cols);

    cost[i][j] = val;
  }

  bool solution_exists() {
    for (int i = 1; i < lines; i++) {
      bool has_val = false;
      for (int j = 1; j < cols; j++) if (cost[i][j] < _INF) { has_val = true; break; }
      if (!has_val) return false;
    }
    return true;
  }

  vector<int>& get_assignment() { return pairV; }

  // Only run this if solution exists
  T run() {
    for (int i = 1, j0 = 0; i < lines; i++) {
      pairV[0] = i;
      minv.assign(cols, _INF);
      used.assign(cols, 0);
      do {
        used[j0] = 1;
        int i0 = pairV[j0], j1;
        T delta = _INF;
        for (int j = 1; j < cols; j++) {
          if (used[j]) continue;
          T cur = cost[i0][j] - pu[i0] - pv[j];
          if (cur < minv[j]) minv[j] = cur, way[j] = j0;
          if (minv[j] < delta) delta = minv[j], j1 = j;
        }

        for (int j = 0; j < cols; j++) {
          if (used[j]) pu[pairV[j]] += delta, pv[j] -= delta;
          else minv[j] -= delta;
        }
        j0 = j1;
      } while (pairV[j0]);

      do {
        int j1 = way[j0];
        pairV[j0] = pairV[j1];
        j0 = j1;
      } while (j0);
    }

    ans = 0;
    for (int j = 1; j < cols; j++) if (pairV[j]) ans += cost[pairV[j]][j];

    if (is_max) ans = -ans;
    if (is_zero_indexed) {
      for (int j = 0; j + 1 < cols; j++) pairV[j] = pairV[j + 1], pairV[j]--;
      pairV[cols - 1] = -1;
    }
    if (swap_coord) {
      vector<int> pairV_sub(lines, 0);
      for (int j = 0; j < cols; j++) if (pairV[j] >= 0) pairV_sub[pairV[j]] = j;
      swap(pairV, pairV_sub);
    }

    return ans;
  }
};

template <bool is_max = false, bool is_zero_indexed = false>
struct HungarianMult : Hungarian<long double, is_max, is_zero_indexed> {
  using super = Hungarian<long double, is_max, is_zero_indexed>;

  HungarianMult(int _n, int _m) : super(_n, _m) {}

  void update(int i, int j, long double x) {
    super::update(i, j, log2(x));
  }
};
```

3 Mathematics

3.1 Fast Exponential

```
// Fast Exponential - O(log b)
ll fexp (ll b, ll e, ll mod) {
  ll ans = 1;
  while (e) {
    if (e&1) ans = (ans*b) % mod;
    b = (b*b) % mod;
    e = e/2;
  }
  return ans;
```

```
}
```

3.2 Fast Fourier Transform

```
// FFT - Polynomial Multiplication in O(n log n)
// Made by tourist

// p(x)^k -> remeber to use fast exponentiation
// mod multiplication -> every coefficient in range [0, mod-1]
// be careful with overflow!

namespace fft {
  typedef double dbl;

  struct num {
    dbl x, y;
    num() { x = y = 0; }
    num(dbl x, dbl y) : x(x), y(y) {}
  };

  inline num operator+ (num a, num b) { return num(a.x + b.x, a.y + b.y); }
  inline num operator- (num a, num b) { return num(a.x - b.x, a.y - b.y); }
  inline num operator* (num a, num b) { return num(a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x); }
  inline num conj(num a) { return num(a.x, -a.y); }

  int base = 1;
  vector<num> roots = {{0, 0}, {1, 0}};
  vector<int> rev = {0, 1};

  const dbl PI = acosl(-1.0);

  void ensure_base(int nbase) {
    if(nbase <= base) return;

    rev.resize(1 << nbase);
    for(int i=0; i < (1 << nbase); i++) {
      rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (nbase - 1));
    }
    roots.resize(1 << nbase);

    while(base < nbase) {
      dbl angle = 2*PI / (1 << (base + 1));
      for(int i = 1 << (base - 1); i < (1 << base); i++) {
        roots[i << 1] = roots[i];
        dbl angle_i = angle * (2 * i + 1 - (1 << base));
        roots[(i << 1) + 1] = num(cos(angle_i), sin(angle_i));
      }
      base++;
    }
  }

  void fft(vector<num> &a, int n = -1) {
    if(n == -1) {
      n = a.size();
    }
    assert((n & (n-1)) == 0);
    int zeros = __builtin_ctz(n);
    ensure_base(zeros);
    int shift = base - zeros;
    for(int i = 0; i < n; i++) {
      if(i < (rev[i] >> shift)) {
        swap(a[i], a[rev[i] >> shift]);
      }
    }
    for(int k = 1; k < n; k <<= 1) {
      for(int i = 0; i < n; i += 2 * k) {
        for(int j = 0; j < k; j++) {
          num z = a[i+j+k] * roots[j+k];
          a[i+j+k] = a[i+j] - z;
          a[i+j] = a[i+j] + z;
        }
      }
    }
  }

  vector<num> fa, fb;
  vector<int> multiply(vector<int> &a, vector<int> &b) {
    int need = a.size() + b.size() - 1;
    int nbase = 0;
    while((1 << nbase) < need) nbase++;
    ensure_base(nbase);
    int sz = 1 << nbase;
    if(sz > (int) fa.size()) {
      fa.resize(sz);
    }
    for(int i = 0; i < sz; i++) {
      int x = (i < (int) a.size() ? a[i] : 0);
      int y = (i < (int) b.size() ? b[i] : 0);
      fa[i] = num(x, y);
    }
    fft(fa, sz);
    num r(0, -0.25 / sz);
    for(int i = 0; i <= (sz >> 1); i++) {
      int j = (sz - i) & (sz - 1);
      num z = (fa[j] * fa[j] - conj(fa[i] * fa[i])) * r;
      if(i != j) {
        fa[j] = (fa[i] * fa[i] - conj(fa[j] * fa[j])) * r;
      }
      fa[i] = z;
    }
    fft(fa, sz);
    vector<int> res(need);
    for(int i = 0; i < need; i++) {
      res[i] = fa[i].x + 0.5;
    }
    return res;
  }

  vector<int> multiply_mod(vector<int> &a, vector<int> &b, int m, int eq = 0) {
    int need = a.size() + b.size() - 1;
    int nbase = 0;
    while ((1 << nbase) < need) nbase++;
    ensure_base(nbase);
    int sz = 1 << nbase;
    if (sz > (int) fa.size()) {
      fa.resize(sz);
    }
    for (int i = 0; i < (int) a.size(); i++) {
      int x = (a[i] % m + m) % m;
      fa[i] = num(x & ((1 << 15) - 1), x >> 15);
    }
    fill(fa.begin() + a.size(), fa.begin() + sz, num {0, 0});
    fft(fa, sz);
    if (sz > (int) fb.size()) {
      fb.resize(sz);
    }
    if (eq) {
      copy(fa.begin(), fa.begin() + sz, fb.begin());
    } else {
      for (int i = 0; i < (int) b.size(); i++) {
        int x = (b[i] % m + m) % m;
        fb[i] = num(x & ((1 << 15) - 1), x >> 15);
      }
      fill(fb.begin() + b.size(), fb.begin() + sz, num {0, 0});
      fft(fb, sz);
    }
    dbl ratio = 0.25 / sz;
    num r2(0, -1);
    num r3(ratio, 0);
    num r4(0, -ratio);
    num r5(0, 1);
    for (int i = 0; i <= (sz >> 1); i++) {
      int j = (sz - i) & (sz - 1);
      num a1 = (fa[i] + conj(fa[j]));
      num a2 = (fa[i] - conj(fa[j])) * r2;
      num b1 = (fb[i] + conj(fb[j])) * r3;
      num b2 = (fb[i] - conj(fb[j])) * r4;
      if (i != j) {
        num c1 = (fa[j] + conj(fa[i]));
        num c2 = (fa[j] - conj(fa[i])) * r2;
        num d1 = (fb[j] + conj(fb[i])) * r3;
        num d2 = (fb[j] - conj(fb[i])) * r4;
        fa[i] = c1 * d1 + c2 * d2 * r5;
        fb[i] = c1 * d2 + c2 * d1;
      }
      fa[j] = a1 * b1 + a2 * b2 * r5;
      fb[j] = a1 * b2 + a2 * b1;
    }
    fft(fa, sz);
    fft(fb, sz);
    vector<int> res(need);
    for (int i = 0; i < need; i++) {
      long long aa = fa[i].x + 0.5;
      long long bb = fb[i].x + 0.5;
      long long cc = fa[i].y + 0.5;
      res[i] = (aa + ((bb % m) << 15) + ((cc % m) << 30)) % m;
    }
    return res;
  }

  vector<int> square_mod(vector<int> &a, int m) {
    return multiply_mod(a, a, m, 1);
  }
}
```

3.3 Mobius Function

```
// Mobius Function
// u(1) = 1
// u(p) = -1
// u(p^k) = 0, k >= 2
// u(a*b) = u(a)*u(b), for a,b co-primes

// Sum for d|n of u(d) = [n == 1]

// Calculate Mobius all integers - O(n log(log n))

int cmp[N], mob[N];
void mobius() {
  for(int i = 1; i < N; i++) mob[i] = 1;
  for(ll i = 2; i < N; i++) if(!cmp[i]) {
    for(ll j = i; j < N; j += i) cmp[j] = 1, mob[j] *= -1;
    for(ll j = i*i; j < N; j += i*i) mob[j] = 0;
  }
}

// Calculate Mobius for 1 integer - O(sqrt(n))
int mobius(int n) {
  if(n == 1) return 1;
  int p = 0;
  for(int i = 2; i*i <= n; i++) {
    if(n % i == 0) {
      n = n/i, p++;
      if(n % i == 0) return 0;
    }
  }
  if(n > 1) p++;
  return p&1 ? -1 : 1;
}
```

3.4 Sieve

```
// Sieve - O(n log(log n))

int cmp[N];
vector<int> p;

void sieve() {
  for(ll i = 2; i < N; i++) if(!cmp[i]) {
    p.push_back(i);
    for(ll j = i*i; j < N; j += i) cmp[j] = 1;
  }
}
```

4 Strings

4.1 Aho-Corasick

```
// Aho-Corasick

// Build: O(sum size of patterns)
// Find total number of matches: O(size of input string)
// Find number of matches for each pattern: O(num of patterns + size of input string)

// ids start from 0 by default!

template <int NUM_OF_PATTERNS = 1000, int ALPHA_SIZE = 62>
struct Aho {
  struct Node {
    int p, char_p, link = -1, str_idx = -1;
    vector<int> nxt;
    Node(int _p = -1, int _char_p = -1) : p(_p), char_p(_char_p), nxt(ALPHA_SIZE, -1) {}
  };

  vector<Node> nodes = { Node() };
  vector<int> ord;
  int cnt = 0;
  int ans;
  bool build_done = false;
  vector<pair<int, int>> rep;
  // how many times the string had a matching
  vector<int> occur;

  // change this if different alphabet
  int remap(char c) {
    if (islower(c)) return c - 'a';
    if (isalpha(c)) return c - 'A' + 26;
    return c - '0' + 52;
  }

  void add(string &p, int id = -1) {
    int u = 0;
    if (id == -1) id = cnt;

    for (char ch : p) {
      int c = remap(ch);
      if (nodes[u].nxt[c] == -1) {
        nodes[u].nxt[c] = (int)nodes.size();
        nodes.push_back(Node(u, c));
      }

      u = nodes[u].nxt[c];
    }

    assert(id < NUM_OF_PATTERNS);

    if (nodes[u].str_idx != -1) rep.push_back({ id, nodes[u].str_idx });
    else nodes[u].str_idx = id;
    cnt++;
  }

  void build() {
    build_done = true;
    queue<int> q;

    for (int i = 0; i < ALPHA_SIZE; i++) {
      if (nodes[0].nxt[i] != -1) q.push(nodes[0].nxt[i]);
      else nodes[0].nxt[i] = 0;
    }

    while(q.size()) {
      int u = q.front();
      if (nodes[u].str_idx != -1) ord.push_back(u);
      q.pop();

      int j = nodes[nodes[u].p].link;
      if (j == -1) nodes[u].link = 0;
      else nodes[u].link = nodes[j].nxt[nodes[u].char_p];

      for (int i = 0; i < ALPHA_SIZE; i++) {
        if (nodes[u].nxt[i] != -1) q.push(nodes[u].nxt[i]);
        else nodes[u].nxt[i] = nodes[nodes[u].link].nxt[i];
      }
    }
  }

  void match(string &s) {
    if (!build_done) build();

    ans = 0;
    occur = vector<int>(NUM_OF_PATTERNS, 0);

    int u = 0;
    for (char ch : s) {
      int c = remap(ch);
      u = nodes[u].nxt[c];

      if (nodes[u].str_idx != -1) occur[nodes[u].str_idx]++;
    }

    for (int i = (int)ord.size() - 1; i >= 0; i--) {
      int v = ord[i];
      int fv = nodes[v].link;
      ans += occur[nodes[v].str_idx];

      if (nodes[fv].str_idx != -1) occur[nodes[fv].str_idx] += occur[nodes[v].str_idx];
    }

    for (pair<int, int> x : rep) occur[x.first] = occur[x.second];
  }
};
```

4.2 Rabin-Karp

```
// Rabin-Karp (String Matching + Hashing)

const int MOD = 1e9+9;
const int B = 313;
char s[N], p[N];
int n, m; // n = strlen(s), m = strlen(p)

// Chance of collision for k generated values and N possible hash values
// e^(-k*(k-1)/2*N)

int rabin(){
  if(n < m) return 0;
  ull hp = 0, hs = 0, E = 1, oc = 0;
  for(int i = 0; i < m; i++){
    hp = ((hp*B)%MOD + p[i])%MOD;
    hs = ((hs*B)%MOD + s[i])%MOD;
    E = (E*B)%MOD;
  }

  if(hs == hp) oc++; //match at 0
  for(int i = m; i < n; i++){
    hs = ((hs*B)%MOD + s[i])%MOD;
    hs = (hs - s[i-m]*E%MOD + MOD)%MOD;
    if(hs == hp) oc++; //match at i-m+1
  }
  return oc;
}
```

4.3 Manacher

```
// Manacher - O(n)

// d1 -> odd : size = 2*d1[i] - 1
// d2 -> even : size = 2*d2[i]


vector<int> d1, d2;

void manacher(string &s) {
  int n = s.size();
  d1.resize(n), d2.resize(n);
  for(int i = 0, l1 = 0, l2 = 0, r1 = -1, r2 = -1; i < n; i++) {
    if(i <= r1) d1[i] = min(d1[r1+l1-i], r1-i+1);
    if(i <= r2) d2[i] = min(d2[r2+l2-i+1], r2-i+1);

    while(i - d1[i] >= 0 and i + d1[i] < n and
        s[i - d1[i]] == s[i + d1[i]]) d1[i]++;

    while(i - d2[i] - 1 >= 0 and i + d2[i] < n and
        s[i - d2[i] - 1] == s[i + d2[i]]) d2[i]++;

    if(i + d1[i] - 1 > r1) l1 = i - d1[i] + 1, r1 = i + d1[i] - 1;
    if(i + d2[i] - 1 > r2) l2 = i - d2[i], r2 = i + d2[i] - 1;
  }
}
```

4.4 Suffix Automaton

```
#include <bits/stdc++.h>
using namespace std;

const int N = 1e5 + 5;

int len[2*N], link[2*N], nxt[2*N][26], szt = 1, last = 1;

void add(string &s) {
  for(auto c : s) {
    int cur = ++szt, p = last, j = c - 'a';
    len[cur] = len[last] + 1;
    while(p and !nxt[p][j]) {
      nxt[p][j] = cur;
      p = link[p];
    }
    if(!p) link[cur] = 1;
    else {
      int q = nxt[p][j];
      if(len[p] + 1 == len[q]) link[cur] = q;
      else {
        int clone = ++szt;
        len[clone] = len[p] + 1;
        link[clone] = link[q];
        for(int i = 0; i < 26; i++)
```

```
          if(nxt[q][i]) nxt[clone][i] = nxt[q][i];
        while(p and nxt[p][j] == q) {
          nxt[p][j] = clone;
          p = link[p];
        }
        link[q] = link[cur] = clone;
      }
    }
    last = cur;
  }
}


int main() {


}
```

4.5 Knuth-Morris-Pratt

```
// KMP - O(n + m)

// max size pattern
const int N = 1e5 + 5;

int lps[N], cnt;

void prekmp(string &p){
  for (int i = 1, j = 0; i < p.size(); i++) {
    while (j and p[j] != p[i]) j = lps[j-1];
    if (p[j] == p[i]) j++;
    lps[i] = j;
  }
}

void kmp(string &s, string &p) {
  for (int i = 0, j = 0; i < s.size(); i++) {
    while (j and p[j] != s[i]) j = lps[j-1];
    if (p[j] == s[i]) j++;
    if (j == p.size()) {
      // match i-j+1
      cnt++;
      j = lps[j-1];
    }
  }
}
```

4.6 Z Function

```
// Z-Function - O(n)

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

4.7 String Hashing

```
// String Hashing
// Rabin Karp - O(n + m)

// max size txt + 1
const int N = 1e6 + 5;

// lowercase letters p = 31 (remember to do s[i] - 'a' + 1)
// uppercase and lowercase letters p = 53 (remember to do s[i] - 'a' + 1)
// any character p = 313

const int MOD = 1e9+9;
ull h[N], p[N];
int cnt;

void build(string &s) {
  p[0] = 1, p[1] = 313;
  for(int i = 1; i <= s.size(); i++) {
    h[i] = ((p[1]*h[i-1]) % MOD + s[i-1]) % MOD;
    p[i] = (p[1]*p[i-1]) % MOD;
  }
}

// 1-indexed
ull fhash(int l, int r) {
  return (h[r] - ((h[l-1]*p[r-l+1]) % MOD) + MOD) % MOD;
}

ull shash(string &pt) {
  ull h = 0;
  for(int i = 0; i < pt.size(); i++)
    h = ((h*p[1]) % MOD + pt[i]) % MOD;
  return h;
}

void rabin_karp(string &s, string &pt) {
  build(s);
  ull hp = shash(pt);
  for(int i = 0, m = pt.size(); i + m <= s.size(); i++) {
    if(fhash(i+1, i+m) == hp) {
      // match at i
      cnt++;
    }
  }
}
```

5 Data Structures

5.1 BIT (Range Update, Point Query)

```
// Binary Indexed Tree
// Range Update and Point Query
// Update - O(log n)
// Query - O(log n)

int bit[N];

void add(int p, int v) {
    for (p+=2; p<N; p+=p&-p) bit[p] += v;
}

void update(int l, int r, int val) { add(l, val), add(r+1, -val); }

int query(int p) {
    int r = 0;
    for (p+=2; p; p-=p&-p) r += bit[p];
    return r;
}
```

5.2 Centroid Decomposition

```
// Centroid Decomposition - O(nlog n)
int n, m, sz[N], forb[N], par[N];

void dfs(int u, int p) {
  sz[u] = 1;
  for(auto v : adj[u]) {
    if(v != p and !forb[v]) {
      dfs(v, u);
      sz[u] += sz[v];
    }
  }
}

int cent(int u, int p, int amt) {
  for(auto v : adj[u]) {
    if(v == p or forb[v]) continue;
    if(sz[v] > amt/2) return cent(v, u, amt);
  }
  return u;
}

void decomp(int u, int p) {
  dfs(u, -1);
  int cen = cent(u, -1, sz[u]);
  forb[cen] = 1;
  if(p != -1) par[cen] = p;

  for(auto v : adj[u])
    if(!forb[v]) decomp(v, cen);
}

// in main
// decomp(1, -1);
```

5.3 Max/Min Queue

```
// Monotonic Queue (aka Max/Min Queue) - Operations in O(1)

template <class T, class C = greater_equal<T>>
struct MonoQueue {
  C cmp;
  T sum = 0;
  int l = 1, r = 0;
  deque<pair<T, int>> dq;

  void push(T x) {
    x -= sum;
    while (!dq.empty() and cmp(dq.back().first, x)) dq.pop_back();
    dq.push_back({x, ++r});
  }

  T query() {
    assert(size() > 0);
    T val = dq.front().first;
    return val + sum;
  }

  void pop() { if (!dq.empty() and dq.front().second == l++) dq.pop_front(); }
  void clear() { sum = 0, l = 1, r = 0, dq.clear(); }
  void add(T x) { sum += x; }
  int size() { return r - l + 1; }
};

template <class T> using MinQueue = MonoQueue<T>;
template <class T> using MaxQueue = MonoQueue<T, less_equal<T>>;
```

5.4 Persistent Segment Tree

```
//Persistent Segment Tree
//Update and Query - O(log n)
//Space - O(n log n + q * log n)

//M -> n log n + q * log n

const int N = 2e5 + 5;

const int M = 1e7 + 5;

int n;
int st[M], lc[M], rc[M];
int cnt, v[N];

void init(int p = 0, int l = 1, int r = n) {
    if (l == r) { st[p] = v[l]; return; }

    int mid = (l + r) / 2;

    lc[p] = ++cnt;
    rc[p] = ++cnt;

    init(lc[p], l, mid);
    init(rc[p], mid + 1, r);

    st[p] = st[lc[p]] + st[rc[p]];
}

int query(int i, int j, int p, int l = 1, int r = n) {
    if (r < i or l > j) return 0;
    if (i <= l and r <= j) return st[p];

    int mid = (l + r) / 2;
    return query(i, j, lc[p], l, mid) + query(i, j, rc[p], mid + 1, r);
}

int update(int idx, int val, int p, int l = 1, int r = n) {
    int u = ++cnt;
    if (l == r) st[u] = val;
    else {
      int mid = (l + r) / 2;
      if (idx <= mid) {
        lc[u] = update(idx, val, lc[p], l, mid);
        rc[u] = rc[p];
      } else {
        lc[u] = lc[p];
        rc[u] = update(idx, val, rc[p], mid + 1, r);
      }
    }

    st[u] = st[lc[u]] + st[rc[u]];
    return u;
}
```

5.5 Segment Tree with Lazy

```
//Segment Tree (Range Query and Point Update)
//Update and Query - O(log n)
//Space - O(n)

int n, v[N], lz[4*N], st[4*N];

// n - size of the array (up to N)
void build(int p = 1, int l = 1, int r = n){
    if(l == r) { st[p] = v[l]; return; }
    build(2*p, l, (l+r)/2), build(2*p + 1, (l+r)/2 + 1, r);
    st[p] = st[2*p] + st[2*p + 1];
}

void push(int p, int l, int r){
  if(lz[p]){
    st[p] = lz[p]*(r-l+1);
    if(l != r) lz[2*p] = lz[2*p + 1] = lz[p];
    lz[p] = 0;
  }
}

int query(int i, int j, int p = 1, int l = 1, int r = n){
    push(p, l, r);
    if(l > j or r < i) return 0;
    if(l >= i and j >= r) return st[p];
    return query(i, j, 2*p, l, (l+r)/2) + query(i, j, 2*p + 1, (l+r)/2 + 1, r);
}

void update(int i, int j, int v, int p = 1, int l = 1, int r = n){
    push(p, l, r);
    if(l > j or r < i) return;
    if(l >= i and j >= r) { lz[p] = v, push(p, l, r); return; }
    update(i, j, v, 2*p, l, (l+r)/2), update(i, j, v, 2*p+1, (l+r)/2+1, r);
    st[p] = st[2*p] + st[2*p + 1];
}
```

5.6 Segment Tree

```
//Segment Tree (Range Query and Point Update)
//NOT the Lazy Propagation version
//Update and Query - O(log n)
//Space - O(n)

int n, v[N], st[4*N];

// n - size of the array (up to N)
// You could do point update in all values of v, instead of using build
void build(int p = 1, int l = 1, int r = n){
    if(l == r) { st[p] = v[l]; return; }
    build(2*p, l, (l+r)/2), build(2*p + 1, (l+r)/2 + 1, r);
    st[p] = st[2*p] + st[2*p + 1];
}

int query(int i, int j, int p = 1, int l = 1, int r = n){
    if(l >= i and j >= r) return st[p];
    if(l > j or r < i) return 0;
    return query(i, j, 2*p, l, (l+r)/2) + query(i, j, 2*p + 1, (l+r)/2 + 1, r);
}

void update(int idx, int v, int p = 1, int l = 1, int r = n){
    if(l == r) { st[p] = v; return; }
    if(idx <= (l+r)/2) update(idx, v, 2*p, l, (l+r)/2);
    else update(idx, v, 2*p + 1, (l+r)/2 + 1, r);
    st[p] = st[2*p] + st[2*p + 1];
}
```

5.7 Treap

```
// Treap
// Operations in O (log n)

mt19937_64 llrand(random_device{}());

struct node{
    int val, cnt;
    node *r, *l;
    ll pri;
    node(int x) : val(x), cnt(1), pri(llrand()), l(0), r(0) {}
};

struct treap{
    node *root;

    int cnt(node *t) {return t ? t->cnt : 0;}

    void update(node *&t){
        if(!t) return;
        t->cnt = cnt(t->l) + cnt(t->l) + 1;
    }

    node *merge(node *l, node *r){
        if(!l and !r) return nullptr;
        if(!l or !r) return l ? l : r;
        node *t;
        if(l->pri > r->pri) t = l, t->r = merge(l->r, r);
        else t = r, t->l = merge(l, r->l);
        update(t);
        return t;
    }

    pair<node*, node*> split(node *t, int pos){
        if(!t) return {0, 0};
        if(cnt(t->l) < pos){
            auto x = split(t->r, pos - cnt(t->l) - 1);
            t->r = x.st;
            update(t);
            return {t, x.nd};
        }

        auto x = split(t->l, pos);
        t->l = x.nd;
        update(t);
        return {x.st, t};
    }
};
```

5.8 Union Find Simple

```
//Union-Find
//Union and Find - O(alpha n)

int par[N], sz[N];

//Path Compression
int find(int a) { return a == par[a] ? a : par[a] = find(par[a]); }

//Ranking
void unite(int a, int b){
    if(find(a) == find(b)) return;
    a = find(a), b = find(b);
    if(sz[a] < sz[b]) swap(a, b);
    sz[a] += sz[b], par[b] = a;
}

//in main
for(int i = 0; i < N; i++) par[i] = i, sz[i] = 1;
```

5.9 Union Find with Rollback

```
//Union-Find with Rollback
//Union and find - O(log n)

int par[N], sz[N];
vector<pii> old_par, old_sz;

int find(int a){ return par[a] == a ? a : find(par[a]); }

void unite(int a, int b){
    if(find(a) == find(b)) return;
    a = find(a), b = find(b);
    if(sz[a] < sz[b]) swap(a, b);
    old_par.pb({b, par[b]});
    old_par.pb({a, sz[a]});
    sz[a] += sz[b], par[b] = a;
}

void roolback(){
    par[old_par.top().st] = old_par.top().nd;
    sz[old_sz.top().st] = old_sz.top().nd;
    old_par.pop();
    old_sz.pop();
}

//in main
for(int i = 0; i < N; i++) par[i] = i, sz[i] = 1;
```

5.10 Union Find Partial Persistent

```
//Union-Find with Partial Persistence
//Union and Find - O(log n)

int t, par[N], sz[N], his[N];

int find(int a, int t){
    if(par[a] == a) return a;
    if(his[a] > t) return a;
    return find(par[a], t);
}

void unite(int a, int b){
    if(find(a, t) == find(b, t)) return;
    a = find(a, t), b = find(b, t), t++;
    if(sz[a] < sz[b]) swap(a, b);
    sz[a] += sz[b], par[b] = a, his[b] = t;
}

//in main
for(int i = 0; i < N; i++) par[i] = i, sz[i] = 1, his[i] = 0;
```

6 Dynamic Programming

6.1 Convex Hull Trick

```
// Convex Hull Trick

// max / min (a_i + m_j * x_i + b_j)
// max : m_j increasing, b_j decreasing
// min : m_j decreasing, b_j increasing

typedef long long type;
struct line { type b, m; };

int nh, pos;
line hull[N];

bool check(line s, line t, line u){
  //attention for overflow
  return ld (u.b - t.b)/(t.m - u.m) > ld (t.b - s.b)/(s.m - t.m);
}
```

```cpp
void update(line s){
  if(nh == 1 and hull[nh-1].b == s.b) nh--;
  if(nh > 0 and s.m >= hull[nh-1].m) return;
  while(nh >= 2 and !check(hull[nh-2], hull[nh-1], s)) nh--;
  pos = min(pos, nh);
  hull[nh++] = s;
}

type eval(int id, type x) { return hull[id].b + hull[id].m*x; }

// Linear Query
// queries always move to the right
/*
type query(type x){
  while(pos+1 < nh and eval(pos, x) > eval(pos+1, x)) pos++;
  // max: change '<' to '>'
  return eval(pos, x);
}
*/

type query(type x){
  int l = 0, r = nh-1, mid;
  while(r - l > 5){
    mid = (l+r)/2;
    // max change '<' to '>'
    if(eval(mid+1, x) < eval(mid, x)) l = mid;
    else r = mid+1;
  }
  type mn = LINF;
  // max change LINF to -LINF
  for(int i = l; i <= r; i++) mn = min(mn, eval(i, x));
  // max change min to max
  return mn;
}
```

6.2 Steiner Tree

```cpp
// Steiner-Tree O(2^t*n^2 + n*3^t + APSP)

// N - number of nodes
// T - number of terminals

int n, t, dp[N][(1 << T)], dist[N][N];

int steiner_tree() {
  // + APSP
  memset(dp, 63, sizeof dp);
  for(int i = 1; i <= n; i++) dp[i][1 << (i-1)] = 0;

  for(int msk = 0; msk < (1 << t); msk++) {
    for(int i = 1; i <= n; i++) {
      for(int ss = msk; ss > 0; ss = (ss - 1) & msk)
        dp[i][msk] = min(dp[i][msk], dp[i][ss] + dp[i][msk - ss]);

      if(dp[i][msk] != INF)
        for(int j = 1; j <= n; j++)
          dp[j][msk] = min(dp[j][msk], dp[i][msk] + dist[i][j]);
    }
  }

  int mn = INF;
  for(int i = 1; i <= n; i++) mn = min(mn, dp[i][(1 << t) - 1]);
  return mn;
}
```

7 Geometry

7.1 Convex Hull

```cpp
// Convex Hull Graham's Scan Algorithm O(nlogn)

struct Point{
    int x, y;
    Point(int x = 0, int y = 0):x(x), y(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); }
    int operator%(Point p){ return x*p.y - y*p.x; }
    int operator*(Point p){ return x*p.x + y*p.y; }
};

vector<Point> pts, hull;
Point ori;
//ori -> the highest leftmost point

bool cmp(Point a, Point b){
    if((b - ori) % (a - ori) > 0) return false;
    if((b - ori) % (a - ori) == 0 and (b - ori)*(b - ori) < (a - ori)*(a - ori)) return false;
    return true;
}

void convex_hull (vector<Point>& pts){
    sort(pts.begin() + 1, pts.end(), cmp);
    hull.pb(pts[0]);
    hull.pb(pts[1]);
    for(int i = 2; i < pts.size(); i++){
        while((hull[hull.size()-1] - hull[hull.size()-2]) % (pts[i] - hull[hull.size()-1]) <= 0) hull.
              pop_back();
        hull.pb(pts[i]);
    }
}
```

7.2 Minimum Enclosing Circle

```cpp
#include <bits/stdc++.h>
using namespace std;

const double EPS = 1e-9;
const double PI = acos(-1.);

struct point{
  double x, y;
  point() : x(0.0), y(0.0) {}
  point(double x, double y) : x(x), y(y) {}
  point operator +(point b){ return point(x+b.x, y+b.y); }
  point operator -(point b){ return point(x-b.x, y-b.y); }
  point operator *(double k) { return point(x*k, y*k); }
  point operator /(double k) { return point(x/k, y/k); }
  double operator %(point b) { return x*b.y - y*b.x; }
};

double dist(point p1, point p2){ return hypot(p1.x - p2.x, p1.y - p2.y); }

point rot90cw(point p1){ return point(p1.y, -p1.x); }

struct circle{
  point c;
  double r;
  circle() { c = point(); r = 0; }
  circle(point c, double r) : c(c), r(r) {}
  bool contains(point p) { return dist(c, p) <= r + EPS; }
};

circle circumcircle(point a, point b, point c) {
        point u = rot90cw(b-a);
  point v = rot90cw(c-a);
        point n = (c-b)/2;
  point ans = ((a+c)/2) + (v*((u%n)/(v%u)));
  return circle(ans, dist(ans, a));
}

// Welzl - Minimum Enclosing Circle O(n)
circle minimumCircle(vector<point> p){
  random_shuffle(p.begin(), p.end());
  circle C = circle(p[0], 0.0);
  for(int i = 0; i < 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])/2, dist(p[j], p[i])/2);
      for(int k = 0; k < j; k++){
        if(C.contains(p[k])) continue;
        C = circumcircle(p[j], p[i], p[k]);
      }
    }
  }
  return C;
}
```

8 Miscellaneous

8.1 Parallel Binary Search

```
// Parallel Binary Search - O(nlog n * cost to update data structure + qlog n * cost for bs condition)

struct query { int i, ans; /*+ query related info*/ };
vector<query> req;
int l_default, r_default, idx = l_default;

void pbs(vector<query>& qs, int l_pbs = l_default, int r_pbs = r_default){
    int mid = (l_pbs + r_pbs) / 2;
    // mid = (L+R+1)/2 if different from simple upper/lower bound
    if(qs.empty()) return;

    while(idx < mid) {
        idx++;
        //add value to data structure
    }
    while(idx > mid) {
        //remove value to data structure
        idx--;
    }

    if(l_pbs == r_pbs) {
        for(auto q : qs) req[q.i].ans = l_pbs;
        return;
    }

    vector<query> vl, vr;
    for (auto& q : qs) {
        if ( /* cond */ ) vl.pb(q);
        else vr.pb(q);
    }

    pbs(vl, l_pbs, mid);
    pbs(vr, mid + 1, r_pbs);
}
```

8.2 Ternary Search

```
// Ternary Search - O(log n)

int ternary_search_min (int l = 1, int r = N){
    int mid, mn = INF, idx;
    while(r - l > 5){
        mid = (l+r)/2;
        if(func(mid + 1) < func(mid)) l = mid;
        else r = mid + 1;
    }
    for(int i = l; i <= r; i++) if(func(i) < mn) mn = func(i), idx = i;
    return idx;
}

int ternary_search_max (int l = 1, int r = N){
    int mid, mx = -INF, idx;
    while(r - l > 5){
        mid = (l+r)/2;
        if(func(mid + 1) > func(mid)) l = mid;
        else r = mid + 1;
    }
    for(int i = l; i <= r; i++) if(func(i) > mx) mx = func(i), idx = i;
    return idx;
}
```

8.3 Ternary Search with Double

```
// Ternary Search - O(log n)

double ternary_search_min (double l, double r){
    double mid1, mid2;
    for(int i = 0; i < 100; i++){
        mid1 = (r-l)/3 + l;
        mid2 = mid1 + (r-l)/3;
        if(func(mid2) < func(mid1)) l = mid1;
        else r = mid2;
    }
    return l;
}

double ternary_search_max (double l, double r){
    double mid1, mid2;
    for(int i = 0; i < 100; i++){
        mid1 = (r-l)/3 + l;
        mid2 = mid1 + (r-l)/3;
        if(func(mid2) > func(mid1)) l = mid1;
        else r = mid2;
    }
    return l;
}
```

9 Math Extra

9.1 Combinatorial formulas

$\sum_{k=0}^{n} k^2 = n(n+1)(2n+1)/6$
$\sum_{k=0}^{n} k^3 = n^2(n+1)^2/4$
$\sum_{k=0}^{n} k^4 = (6n^5+15n^4+10n^3-n)/30$
$\sum_{k=0}^{n} k^5 = (2n^6+6n^5+5n^4-n^2)/12$
$\sum_{k=0}^{n} x^k = (x^{n+1}-1)/(x-1)$
$\sum_{k=0}^{n} kx^k = (x-(n+1)x^{n+1}+nx^{n+2})/(x-1)^2$
$\binom{n}{k} = \frac{n!}{(n-k)!k!}$
$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$
$\binom{n}{k} = \frac{n}{n-k}\binom{n-1}{k}$
$\binom{n}{k} = \frac{n-k+1}{k}\binom{n}{k-1}$
$\binom{n+1}{k} = \frac{n+1}{n-k+1}\binom{n}{k}$
$\binom{n}{k+1} = \frac{n-k}{k+1}\binom{n}{k}$
$\sum_{k=1}^{n} k\binom{n}{k} = n2^{n-1}$
$\sum_{k=1}^{n} k^2\binom{n}{k} = (n+n^2)2^{n-2}$
$\binom{m+n}{r} = \sum_{k=0}^{r}\binom{m}{k}\binom{n}{r-k}$
$\binom{n}{k} = \prod_{i=1}^{k}\frac{n-k+i}{i}$

9.2 Number theory identities

Lucas' Theorem: For non-negative integers m and n and a prime p,

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

where

$$m = m_k p^k + m_{k-1}p^{k-1} + \cdots + m_1 p + m_0$$

is the base p representation of m, and similarly for n.

9.3 Stirling Numbers of the second kind

Number of ways to partition a set of n numbers into k non-empty subsets.

$$\left\{ {n \atop k} \right\} = \frac{1}{k!} \sum_{j=0}^{k} (-1)^{(k-j)} \binom{k}{j} j^n$$

Recurrence relation:

$$\left\{ {0 \atop 0} \right\} = 1$$
$$\left\{ {n \atop 0} \right\} = \left\{ {0 \atop n} \right\} = 1$$
$$\left\{ {n+1 \atop k} \right\} = k \left\{ {n \atop k} \right\} + \left\{ {n \atop k-1} \right\}$$

9.4 Burnside's Lemma

Let G be a finite group that acts on a set X. For each g in G let X^g denote the set of elements in X that are fixed by g, which means $X^g = \{x \in X | g(x) = x\}$. Burnside's lemma assers the following formula for the number of orbits, denoted $|X/G|$:

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

9.5 Numerical integration

RK4: to integrate $\dot{y} = f(t, y)$ with $y_0 = y(t_0)$, compute

$$\begin{aligned}
k_1 &= f(t_n, y_n) \\
k_2 &= f(t_n + \frac{h}{2}, y_n + \frac{h}{2} k_1) \\
k_3 &= f(t_n + \frac{h}{2}, y_n + \frac{h}{2} k_2) \\
k_4 &= f(t_n + h, y_n + h k_3) \\
y_{n+1} &= y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)
\end{aligned}$$

	C	A	N	Assunto	Descricao	Diff
A						
B						
C						
D						
E						
F						
G						
H						
I						
J						
K						
L						