Mathreex ICPC Team Notebook 2024

Contents

1 Template

1.1 Template

```cpp
#include <bits/stdc++.h>

#define mp make_pair
#define pb push_back
#define ppb pop_back
#define all(a) (a).begin(), (a).end()
#define sz(a) (int)a.size()
#define f first
#define s second
#define forn(i, n) for (int i = 0; i < n; i++)
#define forx(i, x, n) for (int i = x; i < n; i++)
#define each(a, x) for (auto &(a) : (x))

using namespace std;

typedef long long ll;
typedef vector<int> vi;
typedef vector<ll> vl;

void solve() {
  // code here
}

int main()
{
  ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
  solve();
  return 0;
}
```

2 Graph

2.1 BFS Algorithm

```
vector<int> bfs(vector<vector<int>>& g , int v) {
    vector<int> dis(g.size(), -1);
    queue<int> q;
    dis[v] = 0;
    q.push(v);
    while(!q.empty()) {
        int node= q.front();
        q.pop();
        for(int x : g[node]) {
            if(dis[x] == -1) {
                dis[x] = dis[node] + 1;
                q.push(x);
            }
        }
    }
    return dis;
}
```

2.2 DFS Algorithm

```
vector<bool> vis(tam);

void dfs(int node){

    vis[node] = 1;
    for(int x : g[node])
        if(!vis[x])
            dfs(x);
}
```

2.3 FloodFill Algorithm

```
int n, m;
int dir[2][4] = {{0,0,1,-1}, {1,-1,0,0}};

vector<vector<int>> tab, visi;

int floodfill(int x, int y) {
    if(x < 0 || y < 0 || x >= n || y >= m || visi[x][y] || tab[x][y] == 0)
        return;
    visi[x][y] = 1;
    int ret = 1;
    for(int i = 0; i < 4; i++)
        ret += floodfill(x + dir[0][i], y + dir[1][i]);
    return ret;
}
```

2.4 Dijkstra's Algorithm

```
typedef long long ll;

const long long INF = 4e18;

vector<ll> dijkstra(vector<vector<pair<ll, ll>>> graph, int n, int initial_node)
{
  vector<ll> dis(n + 1, INF);
  dis[initial_node] = 0;

  priority_queue<pair<ll, ll>, vector<pair<ll, ll>>, greater<pair<ll, ll>>> pq;
  pq.push({0, initial_node});
  while (!pq.empty())
  {
    pll minor = pq.top();
    pq.pop();
    ll actual_cost = minor.f;
    int node = minor.s;
    if (dis[node] < actual_cost)
      continue;
    for (auto to : graph[node])
    {
      int neighbor = to.f;
      ll cost = to.s;
      if (dis[node] + cost < dis[neighbor])
      {
        dis[neighbor] = dis[node] + cost;
        pq.push({dis[neighbor], neighbor});
      }
    }
  }

  return dis;
}
```

2.5 Floyd Warshall's Algorithm

```
typedef long long ll;

vector<vector<ll>> floydWarshall(vector<vector<pair<ll, ll>>> graph, int n)
{
  vector<vector<ll>> dis(n + 1, vl(n + 1, INF));
  forn(i, n) dis[i][i] = 0;

  forn(u, n)
  {
    for (auto to : graph[u])
    {
      ll v = to.f, w = to.s;
      dis[u][v] = min(dis[u][v], w);
      dis[v][u] = min(dis[v][u], w);
    }
  }

  forn(k, n)
  {
    forn(u, n)
    {
      forn(v, n) dis[u][v] = min(dis[u][v], dis[u][k] + dis[k][v]);
    }
  }

  return dis;
}
```

2.6 MST (Kruskal's Algorithm)

```
typedef long long ll;

ll kruskal(vector<pair<ll, pair<int, int>>> edges, int n)
{
  sort(all(edges));
  UnionFind dsu(n + 1);
  int countEdges = 0;
  ll res = 0;
  for (auto edge : edges)
  {
    ll weight = edge.f;
    int u = edge.s.f;
    int v = edge.s.s;
    if (dsu.join(u, v))
    {
      countEdges++;
      res += weight;
    }

    if (countEdges == n - 1)
      return res;
  }

  if (countEdges < n - 1)
    return -1;

  return res;
}
```

2.7 Union Find Structure

```
struct UnionFind
{
  vector<int> p;
  UnionFind(int n) : p(n, -1) {}

  int find(int x)
  {
    if (p[x] == -1)
      return x;

    return p[x] = find(p[x]);
  }

  bool join(int x, int y)
  {
    x = find(x), y = find(y);
    if (x == y)
      return 0;

    p[y] = x;
    return 1;
  }
};
```

2.8 2-SAT Kosaraju

```
/**********************************************************************************************
* 2-SAT (TELL WHETHER A SERIES OF STATEMENTS CAN OR CANNOT BE FEASIBLE AT THE SAME TIME)        *
                                                                                              *
* Time complexity: O(V+E)                                                                     *
* Usage: n          -> number of variables, 1-indexed                                         *
*        p = v(i)   -> picks the "true" state for variable i                                  *
*        p = nv(i)  -> picks the "false" state for variable i, i.e. ~i                        *
*        add(p, q)  -> add clause (p v q) (which also means ~p => q, which also means ~q => p) *
*        run2sat()  -> true if possible, false if impossible                                  *
*        val[i]     -> tells if i has to be true or false for that solution                   *
**********************************************************************************************/

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 clause (a v b)
void add(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;
}

int main () {
                for (int i = 1; i <= n; i++) {
                                if (val[i]); // i-th variable is true
                                else         // i-th variable is false
}
```

2.9 2-SAT Tarjan

```
// 2-SAT - O(V+E)
// For each variable x, we create two nodes in the graph: u and !u
// If the variable has index i, the index of u and !u are: 2*i and 2*i+1
// Adds a statment u => v
void add(int u, int v){
        adj[u].pb(v);
        adj[v^1].pb(u^1);
}

//0-indexed variables; starts from var_0 and goes to var_n-1
for(int i = 0; i < n; i++){
        tarjan(2*i), tarjan(2*i + 1);
        //cmp is a tarjan variable that says the component from a certain node
        if(cmp[2*i] == cmp[2*i + 1]) //Invalid
        if(cmp[2*i] < cmp[2*i + 1]) //Var_i is true
        else //Var_i is false

        //its just a possible solution!
}
```

2.10 Bellman Ford

```
/**************************************************************************************
* BELLMAN-FORD ALGORITHM (SHORTEST PATH TO A VERTEX - WITH NEGATIVE COST)             *
* Time complexity: O(VE)                                                              *
* Usage: dist[node]                                                                   *
* Notation: m:           number of edges                                              *
*           n:           number of vertices                                           *
*           (a, b, w):   edge between a and b with weight w                           *
*           s:           starting node                                                *
**************************************************************************************/
const int N = 1e4+10; // Maximum number of nodes
vector<int> adj[N], adjw[N];
int dist[N], v, w;

memset(dist, 63, sizeof(dist));
dist[0] = 0;
for (int i = 0; i < n-1; ++i)
        for (int u = 0; u < n; ++u)
                for (int j = 0; j < adj[u].size(); ++j)
                        v = adj[u][j], w = adjw[u][j],
                        dist[v] = min(dist[v], dist[u]+w);
```

2.11 Block cut

```
// Tarjan for Block Cut Tree (Node Biconnected Componentes) - O(n + m)
#define pb push_back
#include <bits/stdc++.h>
using namespace std;

const int N = 1e5+5;

// Regular Tarjan stuff
int n, num[N], low[N], cnt, ch[N], art[N];
vector<int> adj[N], st;

int lb[N]; // Last block that node is contained
int bn; // Number of blocks
vector<int> blc[N]; // List of nodes from block

void dfs(int u, int p) {
        num[u] = low[u] = ++cnt;
        ch[u] = adj[u].size();
        st.pb(u);

        if (adj[u].size() == 1) blc[++bn].pb(u);

        for(int v : adj[u]) {
                if (!num[v]) {
                        dfs(v, u), low[u] = min(low[u], low[v]);
                        if (low[v] == num[u]) {
                                if (p != -1 or ch[u] > 1) art[u] = 1;
                                blc[++bn].pb(u);
                                while(blc[bn].back() != v)
                                        blc[bn].pb(st.back()), st.pop_back();
                        }
                }
                else if (v != p) low[u] = min(low[u], num[v]), ch[v]--;
        }
```

```
        if (low[u] == num[u]) st.pop_back();
}

// Nodes from 1 .. n are blocks
// Nodes from n+1 .. 2*n are articulations
vector<int> bct[2*N]; // Adj list for Block Cut Tree

void build_tree() {
        for(int u=1; u<=n; ++u) for(int v : adj[u]) if (num[u] > num[v]) {
                if (lb[u] == lb[v] or blc[lb[u]][0] == v) /* edge u-v belongs to block lb[u] */;
                else { /* edge u-v belongs to block cut tree */;
                        int x = (art[u] ? u + n : lb[u]), y = (art[v] ? v + n : lb[v]);
                        bct[x].pb(y), bct[y].pb(x);
                }
        }
}

void tarjan() {
        for(int u=1; u<=n; ++u) if (!num[u]) dfs(u, -1);
        for(int b=1; b<=bn; ++b) for(int u : blc[b]) lb[u] = b;
        build_tree();
}
```

2.12 Bridges and articulations

```
// Articulation points and Bridges O(V+E)
int par[N], art[N], low[N], num[N], ch[N], cnt;

void articulation(int u) {
        low[u] = num[u] = ++cnt;
        for (int v : adj[u]) {
                if (!num[v]) {
                        par[v] = u; ch[u]++;
                        articulation(v);
                        if (low[v] >= num[u]) art[u] = 1;
                        if (low[v] >  num[u]) { /* u-v bridge */ }
                        low[u] = min(low[u], low[v]);
                }
                else if (v != par[u]) low[u] = min(low[u], num[v]);
        }
}

for (int i = 0; i < n; ++i) if (!num[i])
        articulation(i), art[i] = ch[i]>1;
```

2.13 Dinic

```
// Dinic - O(V^2 * E)
// Bipartite graph or unit flow - O(sqrt(V) * E)
// Small flow - O(F * (V + E))
// USE INF = 1e9!

/**********************************************************************************
* DINIC (FIND MAX FLOW / BIPARTITE MATCHING)                                      *
* Time complexity: O(EV^2)                                                        *
* Usage: dinic()                                                                  *
*        add_edge(from, to, capacity)                                             *
* Testcase:                                                                       *
* add_edge(src, 1, 1);   add_edge(1, snk, 1);   add_edge(2, 3, INF);              *
* add_edge(src, 2, 1);   add_edge(2, snk, 1);   add_edge(3, 4, INF);              *
* add_edge(src, 2, 1);   add_edge(3, snk, 1);                                     *
* add_edge(src, 2, 1);   add_edge(4, snk, 1);   => dinic() = 4                    *
**********************************************************************************/

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

const int N = 1e5+1, INF = 1e9;
struct edge {int v, c, f;};

int n, src, snk, h[N], ptr[N];
vector<edge> edgs;
vector<int> g[N];

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);
}

void clear() {
                memset(h, 0, sizeof h);
                memset(ptr, 0, sizeof ptr);
                edgs.clear();
                for (int i = 0; i < N; i++) g[i].clear();
                src = 0;
                snk = N-1;
}

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

int dfs (int u, int flow) {
        if (!flow or u == snk) return flow;
        for (int &i = ptr[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(ptr, 0, sizeof ptr);
                while (int inc = dfs(src, INF)) flow += inc;
        }
        return flow;
}

//Recover Dinic
void recover(){
        for(int i = 0; i < edgs.size(); i += 2){
                //edge (u -> v) is being used with flow f
                if(edgs[i].f > 0) {
                        int v = edgs[i].v;
                        int u = edgs[i^1].v;
                }
        }
}

/**********************************************************************************************
* FLOW WITH DEMANDS                                                                           *
*                                                                                             *
* 1 - Finding an arbitrary flow                                                               *
* Assume a network with [L, R] on edges (some may have L = 0), let's call it old network.     *
* Create a New Source and New Sink (this will be the src and snk for Dinic).                  *
* Modelling Network:                                                                          *
* 1) Every edge from the old network will have cost R - L                                     *
* 2) Add an edge from New Source to every vertex v with cost:                                 *
*    Sum(L) for every (u, v). (sum all L that LEAVES v)                                       *
* 3) Add an edge from every vertex v to New Sink with cost:                                   *
*    Sum(L) for every (v, w). (sum all L that ARRIVES v)                                      *
* 4) Add an edge from Old Source to Old Sink with cost INF (circulation problem)              *
* The Network will be valid if and only if the flow saturates the network (max flow == sum(L)) *
*                                                                                             *
* 2 - Finding Min Flow                                                                        *
* To find min flow that satisfies just do a binary search in the (Old Sink -> Old Source) edge *
* The cost of this edge represents all the flow from old network                              *
* Min flow = Sum(L) that arrives in Old Sink + flow that leaves (Old Sink -> Old Source)      *
**********************************************************************************************/


int main () {
                clear();
                return 0;
}
```

2.14 Dominator tree

```cpp
// a node u is said to be dominating node v if, from every path from the entry point to v you have to
      pass through u
// so this code is able to find every dominator from a specific entry point (usually 1)
// for directed graphs obviously

const int N = 1e5 + 7;

vector<int> adj[N], radj[N], tree[N], bucket[N];
int sdom[N], par[N], dom[N], dsu[N], label[N], arr[N], rev[N], cnt;

void dfs(int u) {
        cnt++;
        arr[u] = cnt;
        rev[cnt] = u;
        label[cnt] = cnt;
        sdom[cnt] = cnt;
        dsu[cnt] = cnt;
        for(auto e : adj[u]) {
                if(!arr[e]) {
                        dfs(e);
                        par[arr[e]] = arr[u];
                }
                radj[arr[e]].push_back(arr[u]);
        }
}

int find(int u, int x = 0) {
        if(u == dsu[u]) {
                return (x ? -1 : u);
        }
        int v = find(dsu[u], x + 1);
        if(v == -1) {
                return u;
        }
        if(sdom[label[dsu[u]]] < sdom[label[u]]) {
                label[u] = label[dsu[u]];
        }
        dsu[u] = v;
        return (x ? v : label[u]);
}

void unite(int u, int v) {
        dsu[v] = u;
}

// in main

dfs(1);
for(int i = cnt; i >= 1; i--) {
        for(auto e : radj[i]) {
                sdom[i] = min(sdom[i], sdom[find(e)]);
        }
        if(i > 1) {
                bucket[sdom[i]].push_back(i);
        }
        for(auto e : bucket[i]) {
                int v = find(e);
                if(sdom[e] == sdom[v]) {
                        dom[e] = sdom[e];
                } else {
                        dom[e] = v;
                }
        }
        if(i > 1) {
                unite(par[i], i);
        }
}
for(int i = 2; i <= cnt; i++) {
        if(dom[i] != sdom[i]) {
                dom[i] = dom[dom[i]];
        }
        tree[rev[i]].push_back(rev[dom[i]]);
        tree[rev[dom[i]]].push_back(rev[i]);
}
```

2.15 Erdos gallai

```cpp
// Erdos-Gallai - O(nlogn)
// check if it's possible to create a simple graph (undirected edges) from
// a sequence of vertice's degrees
bool gallai(vector<int> v) {
        vector<ll> sum;
        sum.resize(v.size());

        sort(v.begin(), v.end(), greater<int>());
        sum[0] = v[0];
        for (int i = 1; i < v.size(); i++) sum[i] = sum[i-1] + v[i];
        if (sum.back() % 2) return 0;

        for (int k = 1; k < v.size(); k++) {
                int p = lower_bound(v.begin(), v.end(), k, greater<int>()) - v.begin();
                if (p < k) p = k;
                if (sum[k-1] > 1ll*k*(p-1) + sum.back() - sum[p-1]) return 0;
        }
        return 1;
}
```

2.16 Eulerian path

```cpp
vector<int> ans, adj[N];
int in[N];

void dfs(int v){
        while(adj[v].size()){
                int x = adj[v].back();
                adj[v].pop_back();
                dfs(x);
        }
        ans.pb(v);
}

// Verify if there is an eulerian path or circuit
vector<int> v;
for(int i = 0; i < n; i++) if(adj[i].size() != in[i]){
        if(abs((int)adj[i].size() - in[i]) != 1) //-> There is no valid eulerian circuit/path
        v.pb(i);
}

if(v.size()){
        if(v.size() != 2) //-> There is no valid eulerian path
        if(in[v[0]] > adj[v[0]].size()) swap(v[0], v[1]);
        if(in[v[0]] > adj[v[0]].size()) //-> There is no valid eulerian path
        adj[v[1]].pb(v[0]); // Turn the eulerian path into a eulerian circuit
}

dfs(0);
for(int i = 0; i < cnt; i++)
        if(adj[i].size()) //-> There is no valid eulerian circuit/path in this case because the graph
              is not conected

ans.pop_back(); // Since it's a curcuit, the first and the last are repeated
reverse(ans.begin(), ans.end());

int bg = 0; // Is used to mark where the eulerian path begins
if(v.size()){
        for(int i = 0; i < ans.size(); i++)
                if(ans[i] == v[1] and ans[(i + 1)%ans.size()] == v[0]){
                        bg = i + 1;
                        break;
                }
}
```

2.17 Fast Kuhn

```cpp
const int N = 1e5+5;

int x, marcB[N], matchB[N], matchA[N], ans, n, m, p;
vector<int> adj[N];

bool dfs(int v){
        for(int i = 0; i < adj[v].size(); i++){
                int viz = adj[v][i];
                if(marcB[viz] == 1 ) continue;
                marcB[viz] = 1;

                if((matchB[viz] == -1) || dfs(matchB[viz])){
                        matchB[viz] = v;
                        matchA[v] = viz;
                        return true;
                }

        }
        return false;
}

int main(){
        //...
```

```
	for(int i = 0; i<=n; i++) matchA[i] = -1;
	for(int j = 0; j<=m; j++) matchB[j] = -1;

	bool aux = true;
	while(aux){
		for(int j=1; j<=m; j++) marcB[j] = 0;
		aux = false;
		for(int i=1; i<=n; i++){
			if(matchA[i] != -1) continue;
			if(dfs(i)){
				ans++;
				aux = true;
			}
		}
	}
	//...
}
```

2.18 Find cycl 3 4

```
#include <bits/stdc++.h>

using lint = int64_t;

constexpr int MOD = int(1e9) + 7;
constexpr int INF = 0x3f3f3f3f;
constexpr int NINF = 0xcfcfcfcf;
constexpr lint LINF = 0x3f3f3f3f3f3f3f3f;

#define endl '\n'

const long double PI = acosl(-1.0);

int cmp_double(double a, double b = 0, double eps = 1e-9) {
	return a + eps > b ? b + eps > a ? 0 : 1 : -1;
}

using namespace std;

#define P 1000000007
#define N 330000

int n, m;
vector<int> go[N], lk[N];
int w[N], deg[N], pos[N], id[N];

bool circle3() {
	int ans = 0;
	for(int i = 1; i <= n; i++) w[i] = 0;
	for(int x = 1; x <= n; x++)  {
		for(int y : lk[x]) w[y] = 1;
		for(int y : lk[x]) for(int z:lk[y]) if(w[z]) {
			ans=(ans+go[x].size()+go[y].size()+go[z].size() - 6);
			if(ans) return true;
		}
		for(int y:lk[x]) w[y] = 0;
	}
	return false;
}

bool circle4() {
	for(int i = 1; i <= n; i++) w[i] = 0;
	int ans = 0;
	for(int x = 1; x <= n; x++) {
		for(int y:go[x]) for(int z:lk[y]) if(pos[z] > pos[x]) {
			ans = (ans+w[z]);
			w[z]++;
			if(ans) return true;
		}
		for(int y:go[x]) for(int z : lk[y]) w[z] = 0;
	}
	return false;
}

inline bool cmp(const int &x, const int &y) {
	return deg[x] < deg[y];
}

int main() {
	cin.tie(nullptr)->sync_with_stdio(false);
	cin >> n >> m;

	int x, y;
	for(int i = 0; i < n; i++) {
		cin >> x >> y;
	}
	for(int i = 1; i <= n; i++) {
		deg[i] = 0, go[i].clear(), lk[i].clear();
	}
	while (m--){
		int a, b;
		cin >> a >> b;
		deg[a]++, deg[b]++;
		go[a].push_back(b);
		go[b].push_back(a);
	}

	for(int i = 1; i <= n; i++) id[i]= i;
	sort(id+1, id+1+n, cmp);
	for(int i = 1; i<= n; i++) pos[id[i]]=i;
	for(int x = 1; x<= n; x++) {
		for(int y:go[x]) {
			if(pos[y]>pos[x]) lk[x].push_back(y);
		}
	};

	if(circle3()) {
		cout << "3" << endl;
		return 0;
	};

	if(circle4()) {
		cout << "4" << endl;
		return 0;
	};

	cout << "5" << endl;
	return 0;
}
```

2.19 Hungarian

```
// Hungarian - O(m*n^2)
// Assignment Problem

int n, m;
int pu[N], pv[N], cost[N][M];
int pairV[N], way[M], minv[M], used[M];

void hungarian() {
	for(int i = 1, j0 = 0; i <= n; i++) {
		pairV[0] = i;
		memset(minv, 63, sizeof minv);
		memset(used, 0, sizeof used);
		do {
			used[j0] = 1;
			int i0 = pairV[j0], delta = INF, j1;
			for(int j = 1; j <= m; j++) {
				if(used[j]) continue;
				int 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 <= m; 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);
	}
}

// in main
// for(int j = 1; j <= m; j++)
//  if(pairV[j]) ans += cost[pairV[j]][j];
//
```

2.20 Hungarian navarro

```
// Hungarian - O(n^2 * m)
template<bool is_max = false, class T = int, bool is_zero_indexed = false>
struct Hungarian {
```

```cpp
	bool swap_coord = false;
	int lines, cols;
	T 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.resize(lines);
		for (auto& line : cost) line.assign(cols, 0);
	}

	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;
	}

	T run() {
		T _INF = numeric_limits<T>::max();
		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 : public Hungarian<is_max, long double, is_zero_indexed> {
	using super = Hungarian<is_max, long double, 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));
	}
};
```

2.21 Kahn

```cpp
/********************************************************************************
* KAHN'S ALGORITHM (TOPOLOGICAL SORTING)                                        *
                                                             *
* Time complexity: O(V+E)                                                       *
* Notation: adj[i]:  adjacency matrix for node i                                *
*           n:       number of vertices                                         *
*           e:       number of edges                                            *
*           a, b:    edge between a and b                                       *
*           inc:     number of incoming arcs/edges                              *
*           q:       queue with the independent vertices                        *
*           tsort:   final topo sort, i.e. possible order to traverse graph     *
********************************************************************************/

vector <int> adj[N];
int inc[N]; // number of incoming arcs/edges

// undirected graph: inc[v] <= 1
// directed graph:   inc[v] == 0

queue<int> q;
for (int i = 1; i <= n; ++i) if (inc[i] <= 1) q.push(i);

while (!q.empty()) {
	int u = q.front(); q.pop();
	for (int v : adj[u])
		if (inc[v] > 1 and --inc[v] <= 1)
			q.push(v);
}
```

2.22 Kosaraju

```cpp
/********************************************************************************
* KOSARAJU'S ALGORITHM (GET EVERY STRONGLY CONNECTED COMPONENTS (SCC))          *
* Description: Given a directed graph, the algorithm generates a list of every  *
* strongly connected components. A SCC is a set of points in which you can reach *
* every point regardless of where you start from. For instance, cycles can be  *
* a SCC themselves or part of a greater SCC.                                    *
* This algorithm starts with a DFS and generates an array called "ord" which    *
* stores vertices according to the finish times (i.e. when it reaches "return"). *
* Then, it makes a reversed DFS according to "ord" list. The set of points      *
* visited by the reversed DFS defines a new SCC.                                *
* One of the uses of getting all SCC is that you can generate a new DAG (Directed *
* Acyclic Graph), easier to work with, in which each SCC being a "supernode" of *
* the DAG.                                                                      *
* Time complexity: O(V+E)                                                       *
* Notation: adj[i]:   adjacency list for node i                                 *
*           adjt[i]:  reversed adjacency list for node i                        *
*           ord:      array of vertices according to their finish time          *
*           ordn:     ord counter                                               *
*           scc[i]:   supernode assigned to i                                   *
*           scc_cnt:  amount of supernodes in the graph                         *
********************************************************************************/
const int N = 2e5 + 5;

vector<int> adj[N], adjt[N];
int n, ordn, scc_cnt, vis[N], ord[N], scc[N];

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

void dfst(int u) {
	scc[u] = scc_cnt, vis[u] = 0;
	for (auto v : adjt[u]) if (vis[v]) dfst(v);
}

// add edge: u -> v
void add_edge(int u, int v){
	adj[u].push_back(v);
```

```
        adjt[v].push_back(u);
}

//Undirected version:
/*
        int par[N];

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

        void dfst(int u) {
                scc[u] = scc_cnt, vis[u] = 0;
                for (auto v : adj[u]) if(vis[v] and u != par[v]) dfst(v);
        }

        // add edge: u -> v
        void add_edge(int u, int v){
                adj[u].push_back(v);
                adj[v].push_back(u);
        }

*/

// run kosaraju
void kosaraju(){
        for (int i = 1; i <= n; ++i) if (!vis[i]) dfs(i);
        for (int i = ordn - 1; i >= 0; --i) if (vis[ord[i]]) scc_cnt++, dfst(ord[i]);
}
```

2.23 Kuhn

```
/********************************************************************************
* KUHN'S ALGORITHM (FIND GREATEST NUMBER OF MATCHINGS - BIPARTITE GRAPH)          *
* Time complexity: O(VE)                                                         *
* Notation: ans:      number of matchings                                        *
*           b[j]:     matching edge b[j] <-> j                                   *
*           adj[i]:   adjacency list for node i                                  *
*           vis:      visited nodes                                              *
*           x:        counter to help reuse vis list                             *
********************************************************************************/

// TIP: If too slow, shuffle nodes and try again.
int x, vis[N], b[N], ans;

bool match(int u) {
        if (vis[u] == x) return 0;
        vis[u] = x;
        for (int v : adj[u])
                if (!b[v] or match(b[v])) return b[v]=u;
        return 0;
}

for (int i = 1; i <= n; ++i) ++x, ans += match(i);

// Maximum Independent Set on bipartite graph
MIS + MCBM = V

// Minimum Vertex Cover on bipartite graph
MVC = MCBM
```

2.24 LCA

```
// Lowest Common Ancestor <O(nlogn), O(logn)>
const int N = 1e6, M = 25;
int anc[M][N], h[N], rt;

// TODO: Calculate h[u] and set anc[0][u] = parent of node u for each u

// build (sparse table)
anc[0][rt] = rt; // set parent of the root to itself
for (int i = 1; i < M; ++i)
        for (int j = 1; j <= n; ++j)
                anc[i][j] = anc[i-1][anc[i-1][j]];

// query
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];
}
```

2.25 Max weight LCA

```
// Using LCA to find max edge weight between (u, v)

const int N = 1e5+5;  // Max number of vertices
const int K = 20;     // Each 1e3 requires ~ 10 K
const int M = K+5;
int n;                // Number of vertices
vector <pii> adj[N];
int vis[N], h[N], anc[N][M], mx[N][M];

void dfs (int u) {
        vis[u] = 1;
        for (auto p : adj[u]) {
                int v = p.st;
                int w = p.nd;
                if (!vis[v]) {
                        h[v] = h[u]+1;
                        anc[v][0] = u;
                        mx[v][0] = w;
                        dfs(v);
                }
        }
}

void build () {
        // cl(mn, 63) -- Don't forget to initialize with INF if min edge!
        anc[1][0] = 1;
        dfs(1);
        for (int j = 1; j <= K; j++) for (int i = 1; i <= n; i++) {
                anc[i][j] = anc[anc[i][j-1]][j-1];
                mx[i][j] = max(mx[i][j-1], mx[anc[i][j-1]][j-1]);
        }
}

int mxedge (int u, int v) {
        int ans = 0;

        if (h[u] < h[v]) swap(u, v);
        for (int j = K; j >= 0; j--) if (h[anc[u][j]] >= h[v]) {
                ans = max(ans, mx[u][j]);
                u = anc[u][j];
        }
        if (u == v) return ans;
        for (int j = K; j >= 0; j--) if (anc[u][j] != anc[v][j]) {
                ans = max(ans, mx[u][j]);
                ans = max(ans, mx[v][j]);
                u = anc[u][j];
                v = anc[v][j];
        }
        return max({ans, mx[u][0], mx[v][0]});
}
```

2.26 Min cost max flow

```
// USE INF = 1e9!

/*********************************************************************************
* MIN COST MAX FLOW (MINIMUM COST TO ACHIEVE MAXIMUM FLOW)                        *
* Description: Given a graph which represents a flow network where every edge has *
* a capacity and a cost per unit, find the minimum cost to establish the maximum  *
* possible flow from s to t.                                                      *
* Note: When adding edge (a, b), it is a directed edge!                           *
* Usage: min_cost_max_flow()                                                      *
*        add_edge(from, to, cost, capacity)                                       *
* Notation: flw: max flow                                                         *
*           cst: min cost to achieve flw                                          *
* Testcase:                                                                       *
* add_edge(src, 1, 0, 1);   add_edge(1, snk, 0, 1);   add_edge(2, 3, 1, INF);     *
* add_edge(src, 2, 0, 1);   add_edge(2, snk, 0, 1);   add_edge(3, 4, 1, INF);     *
* add_edge(src, 2, 0, 1);   add_edge(3, snk, 0, 1);                               *
* add_edge(src, 2, 0, 1);   add_edge(4, snk, 0, 1);   => flw = 4, cst = 3         *
*********************************************************************************/

// w: weight or cost, c : capacity
struct edge {int v, f, w, c; };
```

```
int n, flw_lmt=INF, src, snk, flw, cst, p[N], d[N], et[N];
vector<edge> e;
vector<int> g[N];

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

void clear() {
	flw_lmt = INF;
	for(int i=0; i<=n; ++i) g[i].clear();
	e.clear();
}

void min_cost_max_flow() {
	flw = 0, cst = 0;
	while (flw < flw_lmt) {
		memset(et, 0, (n+1) * sizeof(int));
		memset(d, 63, (n+1) * sizeof(int));
		deque<int> q;
		q.push_back(src), d[src] = 0;

		while (!q.empty()) {
			int u = q.front(); q.pop_front();
			et[u] = 2;

			for(int i : g[u]) {
				edge &dir = e[i];
				int v = dir.v;
				if (dir.f < dir.c and d[u] + dir.w < d[v]) {
					d[v] = d[u] + dir.w;
					if (et[v] == 0) q.push_back(v);
					else if (et[v] == 2) q.push_front(v);
					et[v] = 1;
					p[v] = i;
				}
			}
		}

		if (d[snk] > INF) break;

		int inc = flw_lmt - flw;
		for (int u=snk; u != src; u = e[p[u]^1].v) {
			edge &dir = e[p[u]];
			inc = min(inc, dir.c - dir.f);
		}

		for (int u=snk; u != src; u = e[p[u]^1].v) {
			edge &dir = e[p[u]], &rev = e[p[u]^1];
			dir.f += inc;
			rev.f -= inc;
			cst += inc * dir.w;
		}

		if (!inc) break;
		flw += inc;
	}
}
```

2.27 Prim

```
// Prim - MST O(ElogE)
vi adj[N], adjw[N];
int vis[N];

priority_queue<pii> pq;
pq.push(mp(0, 0));

while (!pq.empty()) {
	int u = pq.top().nd;
	pq.pop();
	if (vis[u]) continue;
	vis[u]=1;
	for (int i = 0; i < adj[u].size(); ++i) {
		int v = adj[u][i];
		int w = adjw[u][i];
		if (!vis[v]) pq.push(mp(-w, v));
	}
}
```

2.28 Small to large

```
// Imagine you have a tree with colored vertices, and you want to do some type of query on every
    subtree about the colors inside
// complexity: O(nlogn)

vector<int> adj[N], vec[N];
int sz[N], color[N], cnt[N];

void dfs_size(int v = 1, int p = 0) {
	sz[v] = 1;
	for (auto u : adj[v]) {
		if (u != p) {
			dfs_size(u, v);
			sz[v] += sz[u];
		}
	}
}

void dfs(int v = 1, int p = 0, bool keep = false) {
	int Max = -1, bigchild = -1;
	for (auto u : adj[v]) {
		if (u != p && Max < sz[u]) {
			Max = sz[u];
			bigchild = u;
		}
	}
	for (auto u : adj[v]) {
		if (u != p && u != bigchild) {
			dfs(u, v, 0);
		}
	}
	if (bigchild != -1) {
		dfs(bigchild, v, 1);
		swap(vec[v], vec[bigchild]);
	}
	vec[v].push_back(v);
	cnt[color[v]]++;
	for (auto u : adj[v]) {
		if (u != p && u != bigchild) {
			for (auto x : vec[u]) {
				cnt[color[x]]++;
				vec[v].push_back(x);
			}
		}
	}
	// now here you can do what the query wants
	// there are cnt[c] vertex in subtree v color with c
	if (keep == 0) {
		for (auto u : vec[v]) {
			cnt[color[u]]--;
		}
	}
}
```

2.29 SPFA

```
// Shortest Path Faster Algoritm O(VE)
int dist[N], inq[N];

cl(dist,63);
queue<int> q;
q.push(0); dist[0] = 0; inq[0] = 1;

while (!q.empty()) {
	int u = q.front(); q.pop(); inq[u]=0;
	for (int i = 0; i < adj[u].size(); ++i) {
		int v = adj[u][i], w = adjw[u][i];
		if (dist[v] > dist[u] + w) {
			dist[v] = dist[u] + w;
			if (!inq[v]) q.push(v), inq[v] = 1;
		}
	}
}
```

2.30 Stanford Stoer Wagner

```
	// a is a N*N matrix storing the graph we use; a[i][j]=a[j][i]
	memset(use,0,sizeof(use));
	ans=maxlongint;
	for (int i=1;i<N;i++)
	{
		memcpy(visit,use,505*sizeof(int));
```

```
        memset(reach,0,sizeof(reach));
        memset(last,0,sizeof(last));
        t=0;
        for (int j=1;j<=N;j++)
                if (use[j]==0) {t=j;break;}
        for (int j=1;j<=N;j++)
                if (use[j]==0) reach[j]=a[t][j],last[j]=t;
        visit[t]=1;
        for (int j=1;j<=N-i;j++)
        {
                maxc=maxk=0;
                for (int k=1;k<=N;k++)
                        if ((visit[k]==0)&&(reach[k]>maxc)) maxc=reach[k],maxk=k;
                c2=maxk,visit[maxk]=1;
                for (int k=1;k<=N;k++)
                        if (visit[k]==0) reach[k]+=a[maxk][k],last[k]=maxk;
        }
        c1=last[c2];
        sum=0;
        for (int j=1;j<=N;j++)
                if (use[j]==0) sum+=a[j][c2];
        ans=min(ans,sum);
        use[c2]=1;
        for (int j=1;j<=N;j++)
                if ((c1!=j)&&(use[j]==0)) {a[j][c1]+=a[j][c2];a[c1][j]=a[j][c1];}
}
```

2.31 Tarjan

```
// Tarjan for SCC and Edge Biconnected Componentes - O(n + m)
vector<int> adj[N];
stack<int> st;
bool inSt[N];

int id[N], cmp[N];
int cnt, cmpCnt;

void clear(){
        memset(id, 0, sizeof id);
        cnt = cmpCnt = 0;
}

int tarjan(int n){
        int low;
        id[n] = low = ++cnt;
        st.push(n), inSt[n] = true;

        for(auto x : adj[n]){
                if(id[x] and inSt[x]) low = min(low, id[x]);
                else if(!id[x]) {
                        int lowx = tarjan(x);
                        if(inSt[x])
                                low = min(low, lowx);
                }
        }

        if(low == id[n]){
                while(st.size()){
                        int x = st.top();
                        inSt[x] = false;
                        cmp[x] = cmpCnt;

                        st.pop();
                        if(x == n) break;
                }
                cmpCnt++;
        }
        return low;
}
```

2.32 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]});
                        }
                }
        }
}
```

3 DFS

3.1 Coin Change

```
void solve() {
  ll n_coins, total;
  cin >> n_coins >> total;
  vl dp(total + 1, INT32_MAX - 1);
  vl coins(n_coins);
  forn(i, n_coins) cin >> coins[i];

  dp[0] = 0;
  for(i, n_coins) {
    each(coin, coins) {
      if (coin + i > x) continue;
      dp[coin + i] = min(dp[coin + i], dp[i] + 1);
    }
  }

  if (dp[total] + 1 == INT32_MAX) cout << "-1\n";
  else cout << dp[total] << '\n';
}
```

3.2 Knapsack

```
ll knapsack(ll W, vi weights, vi profits, int n) {
  vector<vi> dp(n + 1, vi(W + 1));
  forn(i, n + 1) {
    forn(w, W + 1) {
      if (i == 0 || w == 0) dp[i][w] = 0;
      else if (weights[i - 1] <= w)
        dp[i][w] = max(
          profit[i - 1] + dp[i - 1][w - weights[i - 1]],
          dp[i - 1][w]);
      else
        dp[i][w] = dp[i - 1][w];
    }
  }

  return dp[n][W];
}
```

3.3 Longest Common Subsequence

```
int lcs(string &s1, string &s2) {
  int m = sz(s1), n = sz(s2);

  vector<vi> dp(m + 1, vi(n + 1, 0));
  forx(i, 1, m + 1) {
    forx(j, 1, n + 1) {
      dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
      if (s1[i - 1] == s2[j - 1]) dp[i][j] = max(dp[i][j], dp[i - 1][j - 1] + 1);
    }
  }

  return dp[m][n];
}
```

3.4 Longest Increasing Subsequences

```cpp
int lis(vi &original) {
  vi aux;
  forn(i, sz(original)) {
    auto it = lower_bound(all(aux), original[i]);
    if (it == aux.end()) aux.pb(original[i]);
    else *it = original[i];
  }

  return sz(aux);
}
```

4 Query

4.1 Prefix sum

```cpp
void solve() {
  ll n, q, x, y;
  cin >> n >> q;

  vl nums(n), prefix(n + 1);
  forn(i, n) cin >> nums[i], prefix[i + 1] = prefix[i] + nums[i];

  forn(i, q) {
    cin >> x >> y;
    cout << prefix[y] - prefix[x - 1] << '\n';
  }
}
```

4.2 Prefix sum 2D

```cpp
void solve() {
  ll n, q;
  cin >> n >> q;
  vector<string> s(n); // 0-index

  vector<vl> prefix(n + 1, vl(n + 1)); // 1-index
  forn(i, n) {
    forn(j, n) {
      ll value = s[i][j] == '*';
      prefix[i + 1][j + 1] = (value
                              + prefix[i][j + 1]
                              + prefix[i + 1][j]
                              - prefix[i][j]);
    }
  }

  while (q--) {
    ll x1, y1, x2, y2;
    cin >> x1 >> y1 >> x2 >> y2;
    x1--, y1--, x2--, y2--;

    ll sum = (prefix[x2 + 1][y2 + 1]
              - prefix[x1][y2 + 1]
              - prefix[x2 + 1][y1]
              + prefix[x1][y1]); // 0-index query
    cout << sum << '\n';
  }
}
```

4.3 Fenwick Tree

```cpp
struct BIT { // 1-index
  vl bit;
  ll n;

  BIT(int n) : bit(n+1), n(n) {}

  ll lsb(int i) { return i & -i; }

  void add(int i, ll x) {
    for (; i <= n; i += lsb(i)) bit[i] += x;
  }

  ll sum(int r) {
    ll res = 0;
    for (; r > 0; r -= lsb(r)) res += bit[r];
    return res;
  }

  ll sum(int l, int r) {
    return sum(r) - sum(l-1);
  }

  void set(int i, ll x) {
    add(i, x - sum(i, i));
  }
};
```

4.4 Fenwick Tree 2D

```cpp
struct BIT2D {
  vector<vl> bit;
  ll n, m;

  BIT2D(ll n, ll m) : bit(n + 1, vector<ll>(m + 1)), n(n), m(m) {}

  ll lsb(ll i) {
    return i & -i;
  }

  void add(int row, int col, ll x) {
    for (int i = row; i <= n; i += lsb(i)) {
      for (int j = col; j <= m; j += lsb(j)) {
          bit[i][j] += x;
      }
    }
  }

  ll sum(int row, int col) {
    ll res = 0;
    for (int i = row; i > 0; i -= lsb(i)) {
      for (int j = col; j > 0; j -= lsb(j)) {
        res += bit[i][j];
      }
    }

    return res;
  }

  ll sum(int x1, int y1, int x2, int y2) {
    return (sum(x2, y2)
            - sum(x1 - 1, y2)
            - sum(x2, y1 - 1)
            + sum(x1 - 1, y1 - 1));
  }

  void set(int x, int y, ll val) {
    add(x, y, val - sum(x, y, x, y));
  }
};
```

4.5 General Segtree

```cpp
struct Node {
  ll a = 0;

  Node(ll val = 0) : a(val) {}
};

Node e() {
  Node node;
  return node;
}

Node op(Node a, Node b) {
  Node node;
  node.a = a.a ^ b.a;
  return node;
}

struct Segtree {
  vector<Node> nodes;
  ll n;
```

```
  void init(int n) {
    auto a = vector<Node>(n, e());
    init(a);
  }

  void init(vector<Node>& initial) {
    nodes.clear();
    n = initial.size();
    int size = 1;
    while (size < n) {
      size *= 2;
    }
    nodes.resize(size * 2);
    build(0, 0, n-1, initial);
  }

  void build(int i, int sl, int sr, vector<Node>& initial) {
    if (sl == sr) {
      nodes[i] = initial[sl];
    } else {
      ll mid = (sl + sr) >> 1;
      build(i*2+1, sl, mid, initial);
      build(i*2+2, mid+1,sr,initial);
      nodes[i] = op(nodes[i*2+1], nodes[i*2+2]);
    }
  }

  void update(int i, int sl, int sr, int pos, Node node) {
    if (sl <= pos && pos <= sr) {
      if (sl == sr) {
        nodes[i] = node;
      } else {
        int mid = (sl + sr) >> 1;
        update(i * 2 + 1, sl, mid, pos, node);
        update(i * 2 + 2, mid + 1, sr, pos, node);
        nodes[i] = op(nodes[i*2+1], nodes[i*2+2]);
      }
    }
  }

  void update(int pos, Node node) {
    update(0, 0, n - 1, pos, node);
  }

  Node query(int i, int sl, int sr, int l, int r) {
    if (l <= sl && sr <= r) {
      return nodes[i];
    } else if(sr < l || r < sl) {
      return e();
    } else {
      int mid = (sl + sr) / 2;
      auto a = query(i * 2 + 1, sl, mid, l, r);
      auto b = query(i * 2 + 2, mid + 1, sr, l, r);
      return op(a, b);
    }
  }

  Node query(int l, int r) {
    return query(0, 0, n - 1, l, r);
  }

  Node get(int i) {
    return query(i, i);
  }
};
```

4.6 Sum Lazytree

```
// 0-index
struct Lazytree {
	int n;
	vl sum;
	vl lazySum;

	void init(int nn) {
		sum.clear();
		n = nn;
		int size = 1;
		while (size < n)
		{
			size *= 2;
		}
		sum.resize(size * 2);
		lazySum.resize(size * 2);
	}

	void update(int i, int sl, int sr, int l, int r, ll diff) {
		if (lazySum[i]) {
			sum[i] += (sr - sl + 1) * lazySum[i];
			if (sl != sr) {
				lazySum[i * 2 + 1] += lazySum[i];
				lazySum[i * 2 + 2] += lazySum[i];
			}

			lazySum[i] = 0;
		}

		if (l <= sl && sr <= r) {
			sum[i] += (sr - sl + 1) * diff;
			if (sl != sr) {
				lazySum[i * 2 + 1] += diff;
				lazySum[i * 2 + 2] += diff;
			}
		} else if (sr < l || r < sl) {
		} else {
			int mid = (sl + sr) >> 1;
			update(i * 2 + 1, sl, mid, l, r, diff);
			update(i * 2 + 2, mid + 1, sr, l, r, diff);
			sum[i] = sum[i * 2 + 1] + sum[i * 2 + 2];
		}
	}

	void update(int l, int r, ll diff) {
		assert(l <= r);
		assert(r < n);
		update(0, 0, n - 1, l, r, diff);
	}

	ll query(int i, int sl, int sr, int l, int r) {
		if (lazySum[i]) {
			sum[i] += lazySum[i] * (sr - sl + 1);
			if (sl != sr) {
				lazySum[i * 2 + 1] += lazySum[i];
				lazySum[i * 2 + 2] += lazySum[i];
			}

			lazySum[i] = 0;
		}

		if (l <= sl && sr <= r) {
			return sum[i];
		} else if (sr < l || r < sl) {
			return 0;
		} else {
			int mid = (sl + sr) >> 1;
			return query(i * 2 + 1, sl, mid, l, r) + query(i * 2 + 2, mid + 1, sr, l, r);
		}
	}

	ll query(int l, int r)
	{
		assert(l <= r);
		assert(r < n);
		return query(0, 0, n - 1, l, r);
	}
};
```

5 Geometry

5.1 2D Library

```
typedef long double lf;
const lf EPS = 1e-8L;
const lf E0 = 0.0L;//Keep = 0 for integer coordinates, otherwise = EPS
const lf INF = 5e9;

enum {OUT,IN,ON};

struct pt {
  lf x,y;
  pt(){}
  pt(lf a , lf b): x(a), y(b){}

  pt operator - (const pt &q ) const {
    return {x - q.x , y - q.y };
  }

  pt operator + (const pt &q ) const {
    return {x + q.x , y + q.y };
  }
```

```
  pt operator * (const lf &t ) const {
    return {x * t , y * t };
  }

  pt operator / (const lf &t ) const {
    return {x / t , y / t };
  }

  bool operator < ( const pt & q ) const {
    if( fabsl( x - q.x ) > E0 ) return x < q.x;
    return y < q.y;
  }

  void normalize() {
    lf norm = hypotl( x, y );
    if( fabsl( norm ) > EPS )
      x /= norm, y /= norm;
  }
};

pt rot90( pt p ) { return { -p.y, p.x }; }
pt rot( pt p, lf w ) {
  return { cosl( w ) * p.x - sinl( w ) * p.y, sinl( w ) * p.x + cosl( w ) * p.y };
}

lf norm2(pt p) { return p.x * p.x + p.y * p.y; }
lf dis2(pt p, pt q) { return norm2(p-q); }

lf norm(pt p) { return hypotl ( p.x, p.y ); }
lf dis(pt p, pt q) { return norm( p - q ); }

lf dot(pt p, pt q) { return p.x * q.x + p.y * q.y; }
lf cross(pt p, pt q) { return p.x * q.y - q.x * p.y ; }

lf orient(pt a, pt b, pt c) { return cross( b - a, c - a ); };

lf angle(pt a, pt b){ return atan2(cross(a, b), dot(a, b)); }
// rad => * 180.0 / M_PI
lf angle2(pt a, pt b){ return acos(dot(a, b) / abs(a) / abs(b)); }

lf abs(pt a) { return sqrt(a.x * a.x + a.y * a.y); }

lf proj(pt a, pt b) { return dot(a, b) / abs(b) }

bool in_angle( pt a, pt b, pt c, pt p ) {
  //assert( fabsl( orient( a, b, c ) ) > E0 );
  if( orient( a, b, c ) < -E0 )
    return orient( a, b, p ) >= -E0 || orient( a, c, p ) <= E0;
  return orient( a, b, p ) >= -E0 && orient( a, c, p ) <= E0;
}

struct line {
  pt nv;
  lf c;

  line( pt _nv, lf _c ) : nv( _nv ), c( _c ) {}

  line( lf _a, lf _b, lf _c ) : nv( {_b, -_a} ), c( _c ) {}

  line ( pt p, pt q ) {
    nv = { p.y - q.y, q.x - p.x };
    c = -dot( p, nv );
  }

  lf eval( pt p ) { return dot( nv, p ) + c; }

  lf distance2( pt p ) {
    return eval( p ) / norm2( nv ) * eval( p );
  }

  lf distance( pt p ) {
    return fabsl( eval( p ) ) / norm( nv );
  }

  pt projection( pt p ) {
    return p - nv * ( eval( p ) / norm2( nv ) );
  }

  bool contains(const pt& r) {
    return fabs(cross(nv, r) - c) < EPS;
  }
};

pt lines_intersection( line a, line b ) {
  lf d = cross( a.nv, b.nv );
  //assert( fabsl( d ) > E0 );
  lf dx = a.nv.y * b.c - a.c * b.nv.y;
  lf dy = a.c * b.nv.x - a.nv.x * b.c;
  return { dx / d, dy / d };
}

line bisector( pt a, pt b ) {
  pt nv = ( b - a ), p = ( a + b ) * 0.5L;
  lf c = -dot( nv, p );
  return line( nv, c );
}

struct Circle {
  pt center;
  lf r;

  Circle( pt p, lf rad ) : center( p ), r( rad ) {};

  Circle( pt p, pt q ) {
    center = ( p + q ) * 0.5L;
    r = dis( p, q ) * 0.5L;
  }

  Circle( pt a, pt b, pt c ) {
    line lb = bisector( a, b ), lc = bisector( a, c );
    center = lines_intersection( lb, lc );
    r = dis( a, center );
  }

  int contains( pt &p ) {
    lf det = r * r - dis2( center, p );
    if( fabsl( det ) <= E0 ) return ON;
    return ( det > E0 ? IN : OUT );
  }
};

lf part(pt a, pt b, lf r) {
  lf l = abs(a-b);
  pt p = (b-a)/l;
  lf c = dot(a, p), d = 4.0 * (c*c - dot(a, a) + r*r);
  if (d < EPS) return angle(a, b) * r * r * 0.5;
  d = sqrt(d) * 0.5;
  lf s = -c - d, t = -c + d;
  if (s < 0.0) s = 0.0; else if (s > l) s = l;
  if (t < 0.0) t = 0.0; else if (t > l) t = l;
  pt u = a + p*s, v = a + p*t;
  return (cross(u, v) + (angle(a, u) + angle(v, b)) * r * r) * 0.5;
}

lf circle_poly_intersection( Circle c, vector<pt> p){
  lf ans = 0;
  int n = p.size();
  for (int i = 0; i < n; i++) {
    ans += part(p[i]-c.center, p[(i+1)%n]-c.center, c.r);
  }
  return abs(ans);
}

vector< pt > circle_line_intersection( Circle c, line l ) {
  lf h2 = c.r * c.r - l.distance2( c.center );
  if( fabsl( h2 ) < EPS ) return { l.projection( c.center ) };
  if( h2 < 0.0L ) return {};

  pt dir = rot90( l.nv );
  pt p = l.projection( c.center );
  lf t = sqrtl( h2 / norm2( dir ) );

  return { p + dir * t, p - dir * t };
}

vector< pt > circle_circle_intersection( Circle c1, Circle c2 ) {
  pt dir = c2.center - c1.center;
  lf d2 = dis2( c1.center, c2.center );

  if( d2 <= E0 ) {
    //assert( fabsl( c1.r - c2.r ) > E0 );
    return {};
  }

  lf td = 0.5L * ( d2 + c1.r * c1.r - c2.r * c2.r );
  lf h2 = c1.r * c1.r - td / d2 * td;

  pt p = c1.center + dir * ( td / d2 );
  if( fabsl( h2 ) < EPS ) return { p };
  if( h2 < 0.0L ) return {};

  pt dir_h = rot90(dir) * sqrtl( h2 / d2 );
  return { p + dir_h, p - dir_h };
}

vector< pt > convex_hull( vector< pt > v ) {
  sort( v.begin(), v.end() );//remove repeated points if needed
  const int n = v.size();
  if( n < 3 ) return v;
  vector< pt > ch( 2 * n );

  int k = 0;
  for( int i = 0; i < n; ++ i ) {
    while( k > 1 && orient( ch[k-2], ch[k-1], v[i] ) <= E0 )
```

```
      --k;
    ch[k++] = v[i];
  }

  const int t = k;
  for( int i = n - 2; i >= 0; -- i ) {
    while( k > t && orient( ch[k-2], ch[k-1], v[i] ) <= E0 )
      --k;
    ch[k++] = v[i];
  }
  ch.resize( k - 1 );
  return ch;
}

vector<pt> minkowski( vector<pt> P, vector<pt> Q ) {
  rotate( P.begin(), min_element( P.begin(), P.end() ), P.end() );
  rotate( Q.begin(), min_element( Q.begin(), Q.end() ), Q.end() );

  P.push_back(P[0]), P.push_back(P[1]);
  Q.push_back(Q[0]), Q.push_back(Q[1]);

  vector<pt> ans;
  size_t i = 0, j = 0;
  while(i < P.size() - 2 || j < Q.size() - 2) {
      ans.push_back(P[i] + Q[j]);
      lf dt = cross( P[i + 1] - P[i], Q[j + 1] - Q[j]);
      if(dt >= E0 && i < P.size() - 2) ++i;
      if(dt <= E0 && j < Q.size() - 2) ++j;
  }
  return ans;
}

vector< pt > cut( const vector< pt > &pol, line l ) {
  vector< pt > ans;
  for( int i = 0, n = pol.size(); i < n; ++ i ) {
    lf s1 = l.eval( pol[i] ), s2 = l.eval( pol[(i+1)%n] );
    if( s1 >= -EPS ) ans.push_back( pol[i] );
    if( ( s1 < -EPS && s2 > EPS ) || ( s1 > EPS && s2 < -EPS ) ) {
      line li = line( pol[i], pol[(i+1)%n] );
      ans.push_back( lines_intersection( l, li ) );
    }
  }
  return ans;
}

int point_in_polygon( const vector< pt > &pol, const pt &p ) {
  int wn = 0;
  for( int i = 0, n = pol.size(); i < n; ++ i ) {
    lf c = orient( p, pol[i], pol[(i+1)%n] );
    if( fabsl( c ) <= E0 && dot( pol[i] - p, pol[(i+1)%n] - p ) <= E0 ) return ON;
    if( c > 0 && pol[i].y <= p.y + E0 && pol[(i+1)%n].y - p.y > E0 ) ++wn;
    if( c < 0 && pol[(i+1)%n].y <= p.y + E0 && pol[i].y - p.y > E0 ) --wn;
  }
  return wn ? IN : OUT;
}

int point_in_convex_polygon( const vector < pt > &pol, const pt &p ) {
  int low = 1, high = pol.size() - 1;
  while( high - low > 1 ) {
    int mid = ( low + high ) / 2;
    if( orient( pol[0], pol[mid], p ) >= -E0 ) low = mid;
    else high = mid;
  }
  if( orient( pol[0], pol[low], p ) < -E0 ) return OUT;
  if( orient( pol[low], pol[high], p ) < -E0 ) return OUT;
  if( orient( pol[high], pol[0], p ) < -E0 ) return OUT;

  if( low == 1 && orient( pol[0], pol[low], p ) <= E0 ) return ON;
  if( orient( pol[low], pol[high], p ) <= E0 ) return ON;
  if( high == (int) pol.size() -1 && orient( pol[high], pol[0], p ) <= E0 ) return ON;
  return IN;
}
```

5.2 3D Library

```
typedef double T;
struct p3 {
  T x, y, z;
  // Basic vector operations
  p3 operator + (p3 p) { return {x+p.x, y+p.y, z+p.z }; }
  p3 operator - (p3 p) { return {x - p.x, y - p.y, z - p.z}; }
  p3 operator * (T d) { return {x*d, y*d, z*d}; }
  p3 operator / (T d) { return {x / d, y / d, z / d}; } // only for floating point
  // Some comparators
  bool operator == (p3 p) { return tie(x, y, z) == tie(p.x, p.y, p.z); }
  bool operator != (p3 p) { return !operator == (p); }
};
p3 zero {0, 0, 0 };
T operator | (p3 v, p3 w) { /// dot
  return v.x*w.x + v.y*w.y + v.z*w.z;
}
p3 operator * (p3 v, p3 w) { /// cross
  return { v.y*w.z - v.z*w.y, v.z*w.x - v.x*w.z, v.x*w.y - v.y*w.x };
}
T sq(p3 v) { return v | v; }
double abs(p3 v) { return sqrt(sq(v)); }
p3 unit(p3 v) { return v / abs(v); }
double angle(p3 v, p3 w) {
  double cos_theta = (v | w) / abs(v) / abs(w);
  return acos(max(-1.0, min(1.0, cos_theta)));
}
T orient(p3 p, p3 q, p3 r, p3 s) { /// orient s, pqr form a triangle
  return (q - p) * (r - p) | (s - p);
}
T orient_by_normal(p3 p, p3 q, p3 r, p3 n) { /// same as 2D but in n-normal direction
  return (q - p) * (r - p) | n;
}
struct plane {
  p3 n; T d;
  /// From normal n and offset d
  plane(p3 n, T d): n(n), d(d) {}
  /// From normal n and point P
  plane(p3 n, p3 p): n(n), d(n | p) {}
  /// From three non-collinear points P,Q,R
  plane(p3 p, p3 q, p3 r): plane((q - p) * (r - p), p) {}
  /// - these work with T = int
  T side(p3 p) { return (n | p) - d; }
  double dist(p3 p) { return abs(side(p)) / abs(n); }
  plane translate(p3 t) {return {n, d + (n | t)}; }
  /// - these require T = double
  plane shift_up(double dist) { return {n, d + dist * abs(n)}; }
  p3 proj(p3 p) { return p - n * side(p) / sq(n); }
  p3 refl(p3 p) { return p - n * 2 * side(p) / sq(n); }
};

struct line3d {
  p3 d, o;
  /// From two points P, Q
  line3d(p3 p, p3 q): d(q - p), o(p) {}
  /// From two planes p1, p2 (requires T = double)
  line3d(plane p1, plane p2) {
    d = p1.n * p2.n;
    o = (p2.n * p1.d - p1.n * p2.d) * d / sq(d);
  }
  /// - these work with T = int
  double sq_dist(p3 p) { return sq(d * (p - o)) / sq(d); }
  double dist(p3 p) { return sqrt(sq_dist(p)); }
  bool cmp_proj(p3 p, p3 q) { return (d | p) < (d | q); }
  /// - these require T = double
  p3 proj(p3 p) { return o + d * (d | (p - o)) / sq(d); }
  p3 refl(p3 p) { return proj(p) * 2 - p; }
  p3 inter(plane p) { return o - d * p.side(o) / (p.n | d); }
};

double dist(line3d l1, line3d l2) {
  p3 n = l1.d * l2.d;
  if(n == zero) // parallel
    return l1.dist(l2.o);
  return abs((l2.o - l1.o) | n) / abs(n);
}
p3 closest_on_line1(line3d l1, line3d l2) { /// closest point on l1 to l2
  p3 n2 = l2.d * (l1.d * l2.d);
  return l1.o + l1.d * ((l2.o - l1.o) | n2) / (l1.d | n2);
}
double small_angle(p3 v, p3 w) { return acos(min(abs(v | w) / abs(v) / abs(w), 1.0)); }
double angle(plane p1, plane p2) { return small_angle(p1.n, p2.n); }
bool is_parallel(plane p1, plane p2) { return p1.n * p2.n == zero; }
bool is_perpendicular(plane p1, plane p2) { return (p1.n | p2.n) == 0; }
double angle(line3d l1, line3d l2) { return small_angle(l1.d, l2.d); }
bool is_parallel(line3d l1, line3d l2) { return l1.d * l2.d == zero; }
bool is_perpendicular(line3d l1, line3d l2) { return (l1.d | l2.d) == 0; }
double angle(plane p, line3d l) { return _PI / 2 - small_angle(p.n, l.d); }
bool is_parallel(plane p, line3d l) { return (p.n | l.d) == 0; }
bool is_perpendicular(plane p, line3d l) { return p.n * l.d == zero; }
line3d perp_through(plane p, p3 o) { return line(o, o + p.n); }
plane perp_through(line3d l, p3 o) { return plane(l.d, o); }
```

5.3 Closest points

```
long long dist2(pair<int, int> a, pair<int, int> b) {
  return 1LL * (a.F - b.F) * (a.F - b.F) + 1LL * (a.S - b.S) * (a.S - b.S);
}

pair<int, int> closest_pair(vector<pair<int, int>> a) {
```

```cpp
  int n = a.size();
  assert(n >= 2);
  vector<pair<pair<int, int>, int>> p(n);
  for (int i = 0; i < n; i++) p[i] = {a[i], i};
  sort(p.begin(), p.end());
  int l = 0, r = 2;
  long long ans = dist2(p[0].F, p[1].F);
  pair<int, int> ret = {p[0].S, p[1].S};
  while (r < n) {
    while (l < r && 1LL * (p[r].F.F - p[l].F.F) * (p[r].F.F - p[l].F.F) >= ans) l++;
    for (int i = l; i < r; i++) {
      long long nw = dist2(p[i].F, p[r].F);
      if (nw < ans) {
        ans = nw;
        ret = {p[i].S, p[r].S};
      }
    }
    r++;
  }
  return ret;
}
```

5.4 Convex Hull

```cpp
int orientation(pt a, pt b, pt c) {
  lf v = a.x * (b.y - c.y) + b.x * (c.y - a.y) + c.x * (a.y - b.y);
  if (v < 0) return -1; // clockwise
  if (v > 0) return 1; // counter-clockwise
  return 0;
}

bool cw(pt a, pt b, pt c, bool include_collinear) {
  int o = orientation(a, b, c);
  return o < 0 || (include_collinear && o == 0);
}

bool collinear(pt a, pt b, pt c) { return orientation(a, b, c) == 0; }

void convex_hull(vector<pt>& a, bool include_collinear) {
  pt p0 = *min_element(all(a), [](pt a, pt b) {
    return make_pair(a.y, a.x) < make_pair(b.y, b.x);
  });
  sort(all(a), [&p0](const pt& a, const pt& b) {
    int o = orientation(p0, a, b);
    if (o == 0)
      return (p0.x - a.x) * (p0.x - a.x) + (p0.y - a.y) * (p0.y - a.y)
            < (p0.x - b.x) * (p0.x - b.x) + (p0.y - b.y) * (p0.y - b.y);

    return o < 0;
  });

  if (include_collinear) {
    int i = sz(a) - 1;
    while (i >= 0 && collinear(p0, a[i], a.back())) i--;
    reverse(a.begin() + i + 1, a.end());
  }

  vector<pt> st;
  for (int i = 0; i < sz(a); i++) {
    while (sz(st) > 1 && !cw(st[sz(st) - 2], st.back(), a[i], include_collinear))
      st.pop_back();

    st.push_back(a[i]);
  }

  a = st;
}

lf area(const vector<pt>& fig) {
  lf res = 0;
  for (unsigned i = 0; i < fig.size(); i++) {
    pt p = i ? fig[i - 1] : fig.back();
    pt q = fig[i];
    res += (p.x - q.x) * (p.y + q.y);
  }

  return fabs(res) / 2;
}

lf areaPolygon(const vector<pt>& fig) {
  lf area = 0;
  int n = fig.size();
  for (int i = 0; i < n; i++) {
    int j = (i + 1) % n;
    area += fig[i].x * fig[i].y;
    area -= fig[j].x * fig[j].y;
  }
  return fabs(area) / 2;
}
```

5.5 Point in convex polygon

```cpp
struct pt {
    long long x, y;
    pt() {}
    pt(long long _x, long long _y) : x(_x), y(_y) {}
    pt operator+(const pt &p) const { return pt(x + p.x, y + p.y); }
    pt operator-(const pt &p) const { return pt(x - p.x, y - p.y); }
    long long cross(const pt &p) const { return x * p.y - y * p.x; }
    long long dot(const pt &p) const { return x * p.x + y * p.y; }
    long long cross(const pt &a, const pt &b) const { return (a - *this).cross(b - *this); }
    long long dot(const pt &a, const pt &b) const { return (a - *this).dot(b - *this); }
    long long sqrLen() const { return this->dot(*this); }
};

bool lexComp(const pt &l, const pt &r) {
    return l.x < r.x || (l.x == r.x && l.y < r.y);
}

int sgn(long long val) { return val > 0 ? 1 : (val == 0 ? 0 : -1); }

vector<pt> seq;
pt translation;
int n;

bool pointInTriangle(pt a, pt b, pt c, pt point) {
    long long s1 = abs(a.cross(b, c));
    long long s2 = abs(point.cross(a, b)) + abs(point.cross(b, c)) + abs(point.cross(c, a));
    return s1 == s2;
}

void prepare(vector<pt> &points) {
    n = points.size();
    int pos = 0;
    for (int i = 1; i < n; i++) {
        if (lexComp(points[i], points[pos]))
            pos = i;
    }
    rotate(points.begin(), points.begin() + pos, points.end());

    n--;
    seq.resize(n);
    for (int i = 0; i < n; i++)
        seq[i] = points[i + 1] - points[0];
    translation = points[0];
}

bool pointInConvexPolygon(pt point) {
    point = point - translation;
    if (seq[0].cross(point) != 0 &&
            sgn(seq[0].cross(point)) != sgn(seq[0].cross(seq[n - 1])))
        return false;
    if (seq[n - 1].cross(point) != 0 &&
            sgn(seq[n - 1].cross(point)) != sgn(seq[n - 1].cross(seq[0])))
        return false;

    if (seq[0].cross(point) == 0)
        return seq[0].sqrLen() >= point.sqrLen();

    int l = 0, r = n - 1;
    while (r - l > 1) {
        int mid = (l + r) / 2;
        int pos = mid;
        if (seq[pos].cross(point) >= 0)
            l = mid;
        else
            r = mid;
    }
    int pos = l;
    return pointInTriangle(seq[pos], seq[pos + 1], pt(0, 0), point);
}

bool isIn(const vector<pt>& v, pt p) {
  int n = sz(v);
  if (n < 3) return false;

  lf angleSum = 0;
  for (int i = 0; i < n; i++) {
    pt a = v[i];
    pt b = v[(i + 1) % n];
    double angle = atan2(b.y - p.y, b.x - p.x) - atan2(a.y - p.y, a.x - p.x);
    if (angle >= M_PI) angle -= 2 * M_PI;
    if (angle <= -M_PI) angle += 2 * M_PI;
    angleSum += angle;
```

```
    }

    return fabs(fabs(angleSum) - 2 * M_PI) < 1e-9;
}
```

6 Math

6.1 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 is 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;
}
```

6.2 Advanced

```
/* Line integral = integral(sqrt(1 + (dy/dx)^2)) dx */

/* Multiplicative Inverse over MOD for all 1..N - 1 < MOD in O(N)
 Only works for prime MOD. If all 1..MOD - 1 needed, use N = MOD */
ll inv[N];
inv[1] = 1;
for(int i = 2; i < N; ++i)
        inv[i] = MOD - (MOD / i) * inv[MOD % i] % MOD;

/* Catalan
 f(n) = sum(f(i) * f(n - i - 1)), i in [0, n - 1] = (2n)! / ((n+1)! * n!) = ...
 If you have any function f(n) (there are many) that follows this sequence (0-indexed):
 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440
 than it's the Catalan function */
ll cat[N];
cat[0] = 1;
for(int i = 1; i + 1 < N; i++) // needs inv[i + 1] till inv[N - 1]
        cat[i] = 2ll * (2ll * i - 1) * inv[i + 1] % MOD * cat[i - 1] % MOD;

/* Floor(n / i), i = [1, n], has <= 2 * sqrt(n) diff values.
 Proof: i = [1, sqrt(n)] has sqrt(n) diff values.
 For i = [sqrt(n), n] we have that 1 <= n / i <= sqrt(n)
 and thus has <= sqrt(n) diff values.
*/
/* l = first number that has floor(N / l) = x
 r = last number that has floor(N / r) = x
 N / r >= floor(N / l)
 r <= N / floor(N / l)*/
for(int l = 1, r; l <= n; l = r + 1){
        r = n / (n / l);
        // floor(n / i) has the same value for l <= i <= r
}

/* Recurrence using matriz
 h[i + 2] = a1 * h[i + 1] + a0 * h[i]
 [h[i] h[i-1]] = [h[1] h[0]] * [a1 1] ^ (i - 1)
                              [a0 0]

/* Fibonacci in O(log(N)) with memoization
 f(0) = f(1) = 1
 f(2*k) = f(k)^2 + f(k - 1)^2
 f(2*k + 1) = f(k)*[f(k) + 2*f(k - 1)] */

/* Wilson's Theorem Extension
 B = b1 * b2 * ... * bm (mod n) = +-1, all bi <= n such that gcd(bi, n) = 1
 if(n <= 4 or n = (odd prime)^k or n = 2 * (odd prime)^k) B = -1; for any k
 else B = 1; */

/* Stirling numbers of the second kind
 S(n, k) = Number of ways to split n numbers into k non-empty sets
 S(n, 1) = S(n, n) = 1
 S(n, k) = k * S(n - 1, k) + S(n - 1, k - 1)
 Sr(n, k) = S(n, k) with at least r numbers in each set
 Sr(n, k) = k * Sr(n - 1, k) + (n - 1) * Sr(n - r, k - 1)
                                        (r - 1)
 S(n - d + 1, k - d + 1) = S(n, k) where if indexes i, j belong to the same set, then |i - j| >= d */

/* Burnside's Lemma
 |Classes| = 1 / |G| * sum(K ^ C(g)) for each g in G
 G = Different permutations possible
 C(g) = Number of cycles on the permutation g
 K = Number of states for each element

 Different ways to paint a necklace with N beads and K colors:
 G = {(1, 2, ... N), (2, 3, ... N, 1), ... (N, 1, ... N - 1)}
 gi = (i, i + 1, ... i + N), (taking mod N to get it right) i = 1 ... N
 i -> 2i -> 3i ..., Cycles in gi all have size n / gcd(i, n), so C(gi) = gcd(i, n)
 Ans = 1 / N * sum(K ^ gcd(i, n)), i = 1 ... N
 (For the brave, you can get to Ans = 1 / N * sum(euler_phi(N / d) * K ^ d), d | N) */

/* Mobius Inversion
 Sum of gcd(i, j), 1 <= i, j <= N?
 sum(k->N) k * sum(i->N) sum(j->N) [gcd(i, j) == k], i = a * k, j = b * k
 = sum(k->N) k * sum(a->N/k) sum(b->N/k) [gcd(a, b) == 1]
 = sum(k->N) k * sum(a->N/k) sum(b->N/k) sum(d->N/k) [d | a] * [d | b] * mi(d)
 = sum(k->N) k * sum(d->N/k) mi(d) * floor(N / kd)^2, l = kd, l <= N, k | l, d = l / k
 = sum(l->N) floor(N / l)^2 * sum(k|l) k * mi(l / k)
 If f(n) = sum(x|n)(g(x) * h(x)) with g(x) and h(x) multiplicative, than f(n) is multiplicative
 Hence, g(l) = sum(k|l) k * mi(l / k) is multiplicative
 = sum(l->N) floor(N / l)^2 * g(l) */

/* Frobenius / Chicken McNugget
n, m given, gcd(n, m) = 1, we want to know if it's possible to create N = a * n + b * m
N, a, b >= 0
The greatest number NOT possible is n * m - n - m
We can NOT create (n - 1) * (m - 1) / 2 numbers */
```

6.3 Discrete log

```
// O(sqrt(m))
// Solve c * a^x = b mod(m) for integer x >= 0.
// Return the smallest x possible, or -1 if there is no solution
// If all solutions needed, solve c * a^x = b mod(m) and (a*b) * a^y = b mod(m)
// x + k * (y + 1) for k >= 0 are all solutions
// Works for any integer values of c, a, b and positive m

// Corner Cases:
// 0^x = 1 mod(m) returns x = 0, so you may want to change it to -1
// You also may want to change for 0^x = 0 mod(1) to return x = 1 instead
// We leave it like it is because you might be actually checking for m^x = 0^x mod(m)
// which would have x = 0 as the actual solution.
ll discrete_log(ll c, ll a, ll b, ll m){
        c = ((c % m) + m) % m, a = ((a % m) + m) % m, b = ((b % m) + m) % m;
        if(c == b)
                return 0;

        ll g = __gcd(a, m);
        if(b % g) return -1;

        if(g > 1){
                ll r = discrete_log(c * a / g, a, b / g, m / g);
                return r + (r >= 0);
        }

        unordered_map<ll, ll> babystep;
        ll n = 1, an = a % m;
                   a0
        // set n to the ceil of sqrt(m):
        while(n * n < m) n++, an = (an * a) % m;

        // babysteps:
        ll bstep = b;
        for(ll i = 0; i <= n; i++){
                babystep[bstep] = i;
```

```
		bstep = (bstep * a) % m;
	}

	// giantsteps:
	ll gstep = c * an % m;
	for(ll i = 1; i <= n; i++){
		if(babystep.find(gstep) != babystep.end())
			return n * i - babystep[gstep];
		gstep = (gstep * an) % m;
	}
	return -1;
}
```

6.4 Euler Phi

```
// Euler phi (totient)
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;

// IME2014
int phi[N];
void totient() {
	for (int i = 1; i < N; ++i)  phi[i]=i;
	for (int i = 2; i < N; i+=2) phi[i]>>=1;
	for (int j = 3; j < N; j+=2) if (phi[j]==j) {
		phi[j]--;
		for (int i = 2*j; i < N; i+=j) phi[i]=phi[i]/j*(j-1);
	}
}
```

6.5 Extended euclid

```
// Extended Euclid:
void euclid(ll a, ll b, ll &x, ll &y) {
	if (b) euclid(b, a%b, y, x), y -= x*(a/b);
	else x = 1, y = 0;
}

// find (x, y) such that a*x + b*y = c or return false if it's not possible
// [x + k*b/gcd(a, b), y - k*a/gcd(a, b)] are also solutions
bool diof(ll a, ll b, ll c, ll &x, ll &y){
	euclid(abs(a), abs(b), x, y);
	ll g = abs(__gcd(a, b));
	if(c % g) return false;
	x *= c / g;
	y *= c / g;
	if(a < 0) x = -x;
	if(b < 0) y = -y;
	return true;
}

// auxiliar to find_all_solutions
void shift_solution (ll &x, ll &y, ll a, ll b, ll cnt) {
	x += cnt * b;
	y -= cnt * a;
}

// Find the amount of solutions of
// ax + by = c
// in given intervals for x and y
ll find_all_solutions (ll a, ll b, ll c, ll minx, ll maxx, ll miny, ll maxy) {
	ll x, y, g = __gcd(a, b);
	if(!diof(a, b, c, x, y)) return 0;
	a /= g; b /= g;

	int sign_a = a>0 ? +1 : -1;
	int sign_b = b>0 ? +1 : -1;

	shift_solution (x, y, a, b, (minx - x) / b);
	if (x < minx)
		shift_solution (x, y, a, b, sign_b);
	if (x > maxx)
		return 0;
	int lx1 = x;

	shift_solution (x, y, a, b, (maxx - x) / b);
	if (x > maxx)
		shift_solution (x, y, a, b, -sign_b);
	int rx1 = x;

	shift_solution (x, y, a, b, - (miny - y) / a);
	if (y < miny)
		shift_solution (x, y, a, b, -sign_a);
	if (y > maxy)
		return 0;
	int lx2 = x;

	shift_solution (x, y, a, b, - (maxy - y) / a);
	if (y > maxy)
		shift_solution (x, y, a, b, sign_a);
	int rx2 = x;

	if (lx2 > rx2)
		swap (lx2, rx2);
	int lx = max (lx1, lx2);
	int rx = min (rx1, rx2);

	if (lx > rx) return 0;
	return (rx - lx) / abs(b) + 1;
}

bool crt_auxiliar(ll a, ll b, ll m1, ll m2, ll &ans){
	ll x, y;
	if(!diof(m1, m2, b - a, x, y)) return false;
	ll lcm = m1 / __gcd(m1, m2) * m2;
	ans = ((a + x % (lcm / m1) * m1) % lcm + lcm) % lcm;
	return true;
}

// find ans such that ans = a[i] mod b[i] for all 0 <= i < n or return false if not possible
// ans + k * lcm(b[i]) are also solutions
bool crt(int n, ll a[], ll b[], ll &ans){
	if(!b[0]) return false;
	ans = a[0] % b[0];
	ll l = b[0];
	for(int i = 1; i < n; i++){
	 if(!b[i]) return false;
		if(!crt_auxiliar(ans, a[i] % b[i], l, b[i], ans)) return false;
		l *= (b[i] / __gcd(b[i], l));
	}
	return true;
}
```

6.6 FFT

```
// Fast Fourier Transform - O(nlogn)

/*
// Use struct instead. Performance will be way better!
typedef complex<ld> T;
T a[N], b[N];
*/

struct T {
	ld x, y;
	T() : x(0), y(0) {}
	T(ld a, ld b=0) : x(a), y(b) {}

	T operator/=(ld k) { x/=k; y/=k; return (*this); }
	T operator*(T a) const { return T(x*a.x - y*a.y, x*a.y + y*a.x); }
	T operator+(T a) const { return T(x+a.x, y+a.y); }
	T operator-(T a) const { return T(x-a.x, y-a.y); }
} a[N], b[N];

// a: vector containing polynomial
// n: power of two greater or equal product size
/*
// Use iterative version!
void fft_recursive(T* a, int n, int s) {
	if (n == 1) return;
	T tmp[n];
	for (int i = 0; i < n/2; ++i)
		tmp[i] = a[2*i], tmp[i+n/2] = a[2*i+1];

	fft_recursive(&tmp[0], n/2, s);
	fft_recursive(&tmp[n/2], n/2, s);

	T wn = T(cos(s*2*PI/n), sin(s*2*PI/n)), w(1,0);
	for (int i = 0; i < n/2; i++, w=w*wn)
		a[i] = tmp[i] + w*tmp[i+n/2],
		a[i+n/2] = tmp[i] - w*tmp[i+n/2];
}
*/

void fft(T* a, int n, int s) {
	for (int i=0, j=0; i<n; i++) {
```

```
			if (i>j) swap(a[i], a[j]);
			for (int l=n/2; (j^=l) < l; l>>=1);
		}

		for(int i = 1; (1<<i) <= n; i++){
			int M = 1 << i;
			int K = M >> 1;
			T wn = T(cos(s*2*PI/M), sin(s*2*PI/M));
			for(int j = 0; j < n; j += M) {
				T w = T(1, 0);
				for(int l = j; l < K + j; ++l){
					T t = w*a[l + K];
					a[l + K] = a[l]-t;
					a[l] = a[l] + t;
					w = wn*w;
				}
			}
		}
	}

	// assert n is a power of two greater of equal product size
	// n = na + nb; while (n&(n-1)) n++;
	void multiply(T* a, T* b, int n) {
		fft(a,n,1);
		fft(b,n,1);
		for (int i = 0; i < n; i++) a[i] = a[i]*b[i];
		fft(a,n,-1);
		for (int i = 0; i < n; i++) a[i] /= n;
	}

	// Convert to integers after multiplying:
	// (int)(a[i].x + 0.5);
```

6.7 FFT Tourist

```
	//
	//  FFT made by tourist. It if faster and more supportive, although it requires more lines of code.
	//  Also, it allows operations with MOD, which is usually an issue in FFT problems.
	//
	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);
	}
}
```

6.8 FWHT

```
// Fast Walsh-Hadamard Transform - O(nlogn)
//
// Multiply two polynomials, but instead of x^a * x^b = x^(a+b)
// we have x^a * x^b = x^(a XOR b).
//
// WARNING: assert n is a power of two!
void fwht(ll* a, int n, bool inv) {
	for(int l=1; 2*l <= n; l<<=1) {
		for(int i=0; i < n; i+=2*l) {
			for(int j=0; j<l; j++) {
				ll u = a[i+j], v = a[i+l+j];

				a[i+j] = (u+v) % MOD;
				a[i+l+j] = (u-v+MOD) % MOD;
				// % is kinda slow, you can use add() macro instead
				// #define add(x,y) (x+y >= MOD ? x+y-MOD : x+y)
			}
		}
	}

	if(inv) {
		for(int i=0; i<n; i++) {
			a[i] = a[i] / n;
		}
	}
}


/* FWHT AND
	Matrix : Inverse
	0 1      -1 1
	1 1       1 0
*/
void fwht_and(vi &a, bool inv) {
	vi ret = a;
	ll u, v;
	int tam = a.size() / 2;
	for(int len = 1; 2 * len <= tam; len <<= 1) {
		for(int i = 0; i < tam; i += 2 * len) {
			for(int j = 0; j < len; j++) {
				u = ret[i + j];
				v = ret[i + len + j];
				if(!inv) {
					ret[i + j] = v;
					ret[i + len + j] = u + v;
				}
				else {
					ret[i + j] = -u + v;
					ret[i + len + j] = u;
				}
			}
		}
	}
	a = ret;
}


/* FWHT OR
	Matrix : Inverse
	1 1      0  1
	1 0      1 -1
*/
void fft_or(vi &a, bool inv) {
	vi ret = a;
	ll u, v;
	int tam = a.size() / 2;
	for(int len = 1; 2 * len <= tam; len <<= 1) {
		for(int i = 0; i < tam; i += 2 * len) {
			for(int j = 0; j < len; j++) {
				u = ret[i + j];
				v = ret[i + len + j];
				if(!inv) {
					ret[i + j] = u + v;
					ret[i + len + j] = u;
				}
				else {
					ret[i + j] = v;
					ret[i + len + j] = u - v;
				}
			}
		}
	}
	a = ret;
}
```

6.9 Gauss elim

```
//Gaussian Elimination
//double A[N][M+1], X[M]

// if n < m, there's no solution
// column m holds the right side of the equation
// X holds the solutions

for(int j=0; j<m; j++) { //collumn to eliminate
	int l = j;
	for(int i=j+1; i<n; i++) //find largest pivot
		if(abs(A[i][j])>abs(A[l][j]))
			l=i;
	if(abs(A[i][j]) < EPS) continue;
	for(int k = 0; k < m+1; k++) { //Swap lines
		swap(A[l][k],A[j][k]);
	}
	for(int i = j+1; i < n; i++) { //eliminate column
		double t=A[i][j]/A[j][j];
		for(int k = j; k < m+1; k++)
			A[i][k]-=t*A[j][k];
	}
}

for(int i = m-1; i >= 0; i--) { //solve triangular system
	for(int j = m-1; j > i; j--)
		A[i][m] -= A[i][j]*X[j];
	X[i]=A[i][m]/A[i][i];
}
```

6.10 Gauss elim ext

```
// Gauss-Jordan Elimination with Scaled Partial Pivoting
// Extended to Calculate Inverses - O(n^3)
// To get more precision choose m[j][i] as pivot the element such that m[j][i] / mx[j] is maximized.
// mx[j] is the element with biggest absolute value of row j.

ld C[N][M]; // N = 1000, M = 2*N+1;
int row, col;

bool elim() {
	for(int i=0; i<row; ++i) {
		int p = i; // Choose the biggest pivot
		for(int j=i; j<row; ++j) if (abs(C[j][i]) > abs(C[p][i])) p = j;
		for(int j=i; j<col; ++j) swap(C[i][j], C[p][j]);

		if (!C[i][i]) return 0;

		ld c = 1/C[i][i]; // Normalize pivot line
		for(int j=0; j<col; ++j) C[i][j] *= c;

		for(int k=i+1; k<col; ++k) {
			ld c = -C[k][i]; // Remove pivot variable from other lines
			for(int j=0; j<col; ++j) C[k][j] += c*C[i][j];
		}
	}

	// Make triangular system a diagonal one
	for(int i=row-1; i>=0; --i) for(int j=i-1; j>=0; --j) {
```

```
			ld c = -C[j][i];
			for(int k=i; k<col; ++k) C[j][k] += c*C[i][k];
		}

		return 1;
}

// Finds inv, the inverse of matrix m of size n x n.
// Returns true if procedure was successful.
bool inverse(int n, ld m[N][N], ld inv[N][N]) {
	for(int i=0; i<n; ++i) for(int j=0; j<n; ++j)
		C[i][j] = m[i][j], C[i][j+n] = (i == j);

	row = n, col = 2*n;
	bool ok = elim();

	for(int i=0; i<n; ++i) for(int j=0; j<n; ++j) inv[i][j] = C[i][j+n];
	return ok;
}

// Solves linear system m*x = y, of size n x n
bool linear_system(int n, ld m[N][N], ld *x, ld *y) {
	for(int i = 0; i < n; ++i) for(int j = 0; j < n; ++j) C[i][j] = m[i][j];
	for(int j = 0; j < n; ++j) C[j][n] = x[j];

	row = n, col = n+1;
	bool ok = elim();

	for(int j=0; j<n; ++j) y[j] = C[j][n];
	return ok;
}
```

6.11 Gauss elim prime

```
//ll A[N][M+1], X[M]

for(int j=0; j<m; j++) { //collumn to eliminate
	int l = j;
	for(int i=j+1; i<n; i++) //find nonzero pivot
		if(A[i][j]%p)
			l=i;
	for(int k = 0; k < m+1; k++) { //Swap lines
		swap(A[l][k],A[j][k]);
	}
	for(int i = j+1; i < n; i++) { //eliminate column
		ll t=mulmod(A[i][j],inv(A[j][j],p),p);
		for(int k = j; k < m+1; k++)
			A[i][k]=(A[i][k]-mulmod(t,A[j][k],p)+p)%p;
	}
}

for(int i = m-1; i >= 0; i--) { //solve triangular system
	for(int j = m-1; j > i; j--)
		A[i][m] = (A[i][m] - mulmod(A[i][j],X[j],p)+p)%p;
		X[i] = mulmod(A[i][m],inv(A[i][i],p),p);
}
```

6.12 Gauss elim xor

```
// Gauss Elimination for xor boolean operations
// Return false if not possible to solve
// Use boolean matrixes 0-indexed
// n equations, m variables, O(n * m * m)
// eq[i][j] = coefficient of j-th element in i-th equation
// r[i] = result of i-th equation
// Return ans[j] = xj that gives the lexicographically greatest solution (if possible)
// (Can be changed to lexicographically least, follow the comments in the code)
// WARNING!! The arrays get changed during de algorithm

bool eq[N][M], r[N], ans[M];

bool gauss_xor(int n, int m){
	for(int i = 0; i < m; i++)
		ans[i] = true;
	int lid[N] = {0}; // id + 1 of last element present in i-th line of final matrix
	int l = 0;
	for(int i = m - 1; i >= 0; i--){
		for(int j = l; j < n; j++)
			if(eq[j][i]){ // pivot
				swap(eq[l], eq[j]);
				swap(r[l], r[j]);
			}
		if(l == n || !eq[l][i])
			continue;
		lid[l] = i + 1;
		for(int j = l + 1; j < n; j++){ // eliminate column
			if(!eq[j][i])
				continue;
			for(int k = 0; k <= i; k++)
				eq[j][k] ^= eq[l][k];
			r[j] ^= r[l];
		}
		l++;
	}
	for(int i = n - 1; i >= 0; i--){ // solve triangular matrix
		for(int j = 0; j < lid[i + 1]; j++)
			r[i] ^= (eq[i][j] && ans[j]);
		// for lexicographically least just delete the for bellow
		for(int j = lid[i + 1]; j + 1 < lid[i]; j++){
			ans[j] = true;
			r[i] ^= eq[i][j];
		}
		if(lid[i])
			ans[lid[i] - 1] = r[i];
		else if(r[i])
			return false;
	}
	return true;
}
```

6.13 GSS

```
double gss(double l, double r) {
	double m1 = r-(r-l)/gr, m2 = l+(r-l)/gr;
	double f1 = f(m1), f2 = f(m2);
	while(fabs(l-r)>EPS) {
		if(f1>f2) l=m1, f1=f2, m1=m2, m2=l+(r-l)/gr, f2=f(m2);
		else r=m2, f2=f1, m2=m1, m1=r-(r-l)/gr, f1=f(m1);
	}
	return l;
}
```

6.14 Josephus

```
// UFMG
/* Josephus Problem - It returns the position to be, in order to not die. O(n)*/
/* With k=2, for instance, the game begins with 2 being killed and then n+2, n+4, ... */
ll josephus(ll n, ll k) {
	if(n==1) return 1;
	else return (josephus(n-1, k)+k-1)%n+1;
}

/* Another Way to compute the last position to be killed - O(d * log n) */
ll josephus(ll n, ll d) {
	ll K = 1;
	while (K <= (d - 1)*n) K = (d * K + d - 2) / (d - 1);
	return d * n + 1 - K;
}
```

6.15 Matrix

```
/*
	This code assumes you are multiplying two matrices that can be multiplied: (A nxp * B pxm)
	Matrix fexp assumes square matrices
*/

const int MOD = 1e9 + 7;
typedef long long ll;
typedef long long type;

struct matrix{
	//matrix n x m
	vector<vector<type>> a;
	int n, m;
	matrix() = default;

	matrix(int _n, int _m) : n(_n), m(_m){
		a.resize(n, vector<type>(m));
	}

	matrix operator *(matrix other){
		matrix result(this->n, other.m);
```

```
			for(int i = 0; i < result.n; i++){
				for(int j = 0; j < result.m; j++){
					for(int k = 0; k < this->m; k++){
						result.a[i][j] = (result.a[i][j] + a[i][k] * other.a[k][j]);
						//result.a[i][j] = (result.a[i][j] + (a[i][k] * other.a[k][j])
							% MOD) % MOD;
					}
				}
			}
			return result;
		}
};

matrix identity(int n){
	matrix id(n, n);
	for(int i = 0; i < n; i++) id.a[i][i] = 1;
	return id;
}

matrix fexp(matrix b, ll e){
	matrix ans = identity(b.n);
	while(e){
		if(e & 1) ans = (ans * b);
		b = b * b;
		e >>= 1;
	}
	return ans;
}
```

6.16 Mobius

```
// 1 if n == 1
// 0 if exists x | n%(x^2) == 0
// else (-1)^k, k = #(p) | p is prime and n%p == 0

//Calculate Mobius for all integers using sieve
//O(n*log(log(n)))
void mobius() {
	for(int i = 1; i < N; i++) mob[i] = 1;

	for(ll i = 2; i < N; i++) if(!sieve[i]){
		for(ll j = i; j < N; j += i) sieve[j] = i, 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 /= i;
			p++;
			if(n%i == 0) return 0;
		}
	if(n > 1) p++;
	return p&1 ? -1 : 1;
}
*/
```

6.17 Mobius inversion

```
// multiplicative function calculator
// euler_phi and mobius are multiplicative
// if another f[N] needed just remove comments
// O(N)

bool p[N];
vector<ll> primes;
ll g[N];
// ll f[N];

void mfc(){
	// if g(1) != 1 than it's not multiplicative
	g[1] = 1;
	// f[1] = 1;
	primes.clear();
	primes.reserve(N / 10);
	for(ll i = 2; i < N; i++){
		if(!p[i]){
			primes.push_back(i);
			for(ll j = i; j < N; j *= i){
				g[j] = // g(p^k) you found
				// f[j] = f(p^k) you found
				p[j] = (j != i);
			}
		}
		for(ll j : primes){
			if(i * j >= N || i % j == 0)
				break;
			for(ll k = j; i * k < N; k *= j){
				g[i * k] = g[i] * g[k];
				// f[i * k] = f[i] * f[k];
				p[i * k] = true;
			}
		}
	}
}
```

6.18 NTT

```
// Number Theoretic Transform - O(nlogn)

// if long long is not necessary, use int instead to improve performance
const int mod = 20*(1<<23)+1;
const int root = 3;

ll w[N];

// a: vector containing polynomial
// n: power of two greater or equal product size
void ntt(ll* a, int n, bool inv) {
	for (int i=0, j=0; i<n; i++) {
		if (i>j) swap(a[i], a[j]);
		for (int l=n/2; (j^=l) < l; l>>=1);
	}

	// TODO: Rewrite this loop using FFT version
	ll k, t, nrev;
	w[0] = 1;
	k = exp(root, (mod-1) / n, mod);
	for (int i=1;i<=n;i++) w[i] = w[i-1] * k % mod;
	for(int i=2; i<=n; i<<=1) for(int j=0; j<n; j+=i) for(int l=0; l<(i/2); l++) {
		int x = j+l, y = j+l+(i/2), z = (n/i)*l;
		t = a[y] * w[inv ? (n-z) : z] % mod;
		a[y] = (a[x] - t + mod) % mod;
		a[x] = (a[j+l] + t) % mod;
	}

	nrev = exp(n, mod-2, mod);
	if (inv) for(int i=0; i<n; ++i) a[i] = a[i] * nrev % mod;
}

// assert n is a power of two greater of equal product size
// n = na + nb; while (n&(n-1)) n++;
void multiply(ll* a, ll* b, int n) {
	ntt(a, n, 0);
	ntt(b, n, 0);
	for (int i = 0; i < n; i++) a[i] = a[i]*b[i] % mod;
	ntt(a, n, 1);
}
```

6.19 Pollard rho

```
// factor(N, v) to get N factorized in vector v
// O(N ^ (1 / 4)) on average
// Miller-Rabin - Primarily Test O(|base|*(logn)^2)
ll addmod(ll a, ll b, ll m){
	if(a >= m - b) return a + b - m;
	return a + b;
}

ll mulmod(ll a, ll b, ll m){
	ll ans = 0;
	while(b){
		if(b & 1) ans = addmod(ans, a, m);
		a = addmod(a, a, m);
		b >>= 1;
	}
	return ans;
}

ll fexp(ll a, ll b, ll n){
```

```
	ll r = 1;
	while(b){
		if(b & 1) r = mulmod(r, a, n);
		a = mulmod(a, a, n);
		b >>= 1;
	}
	return r;
}

bool miller(ll a, ll n){
	if (a >= n) return true;
	ll s = 0, d = n - 1;
	while(d % 2 == 0) d >>= 1, s++;
	ll x = fexp(a, d, n);
	if (x == 1 || x == n - 1) return true;
	for (int r = 0; r < s; r++, x = mulmod(x,x,n)){
		if (x == 1) return false;
		if (x == n - 1) return true;
	}
	return false;
}

bool isprime(ll n){
	if(n == 1) return false;
	int base[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
	for (int i = 0; i < 12; ++i) if (!miller(base[i], n)) return false;
	return true;
}

ll pollard(ll n){
	ll x, y, d, c = 1;
	if (n % 2 == 0) return 2;
	while(true){
		y = x = 2;
		while(true){
			x = addmod(mulmod(x,x,n), c, n);
			y = addmod(mulmod(y,y,n), c, n);
			y = addmod(mulmod(y,y,n), c, n);
			if (x == y) break;
			d = __gcd(abs(x-y), n);
			if (d > 1) return d;
		}
		c++;
	}
}

vector<ll> factor(ll n){
	if (n == 1 || isprime(n)) return {n};
	ll f = pollard(n);
	vector<ll> l = factor(f), r = factor(n / f);
	l.insert(l.end(), r.begin(), r.end());
	sort(l.begin(), l.end());
	return l;
}

//n < 2,047 base = {2};
//n < 9,080,191 base = {31, 73};
//n < 2,152,302,898,747 base = {2, 3, 5, 7, 11};
//n < 318,665,857,834,031,151,167,461 base = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
//n < 3,317,044,064,679,887,385,961,981 base = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41};
```

6.20 Pollard rho optimization

```
// We recomend you to use pollard-rho.cpp! I've never needed this code, but here it is.
// This uses Brent's algorithm for cycle detection
//
std::mt19937 rng((int) std::chrono::steady_clock::now().time_since_epoch().count());

ull func(ull x, ull n, ull c) { return (mulmod(x, x, n) + c) % n; // f(x) = (x^2 + c) % n; }

ull pollard(ull n) {
	// Finds a positive divisor of n
	ull x, y, d, c;
	ull pot, lam;
	if(n % 2 == 0) return 2;
	if(isprime(n)) return n;

	while(1) {
		y = x = 2; d = 1;
		pot = lam = 1;
		while(1) {
			c = rng() % n;
			if(c != 0 and (c+2)%n != 0) break;
		}
		while(1) {
			if(pot == lam) {
				x = y;
				pot <<= 1;
				lam = 0;
			}
			y = func(y, n, c);
			lam++;
			d = gcd(x >= y ? x-y : y-x, n);
			if (d > 1) {
				if(d == n) break;
				else return d;
			}
		}
	}
}

void fator(ull n, vector<ull> &v) {
	// prime factorization of n, put into a vector v.
	//
	// for each prime factor of n, it is repeated the amount of times
	// that it divides n
	//
	// ex : n == 120, v = {2, 2, 2, 3, 5};
	//
	//
	if(isprime(n)) { v.pb(n); return; }
	vector<ull> w, t; w.pb(n); t.pb(1);

	while(!w.empty()) {
		ull bck = w.back();
		ull div = pollard(bck);

		if(div == w.back()) {
			int amt = 0;
			for(int i=0; i < (int) w.size(); i++) {
				int cur = 0;
				while(w[i] % div == 0) {
					w[i] /= div;
					cur++;
				}
				amt += cur * t[i];
				if(w[i] == 1) {
					swap(w[i], w.back());
					swap(t[i], t.back());
					w.pop_back();
					t.pop_back();
				}
			}
			while(amt--) v.pb(div);
		}
		else {
			int amt = 0;
			while(w.back() % div == 0) {
				w.back() /= div;
				amt++;
			}
			amt *= t.back();
			if(w.back() == 1) {
				w.pop_back();
				t.pop_back();
			}

			w.pb(div);
			t.pb(amt);
		}
	}

	// the divisors will not be sorted, so you need to sort it afterwards
	sort(v.begin(), v.end());
}
```

6.21 Prime factors

```
// Prime factors (up to 9*10^13. For greater see Pollard Rho)
vi factors;
int ind=0, pf = primes[0];
while (pf*pf <= n) {
	while (n%pf == 0) n /= pf, factors.pb(pf);
	pf = primes[++ind];
}
if (n != 1) factors.pb(n);
```

6.22 Primitive root

```
// Finds a primitive root modulo p
```

```
// To make it works for any value of p, we must add calculation of phi(p)
// n is 1, 2, 4 or p^k or 2*p^k (p odd in both cases)
ll root(ll p) {
        ll n = p-1;
        vector<ll> fact;

        for (int i=2; i*i<=n; ++i) if (n % i == 0) {
                fact.push_back (i);
                while (n % i == 0) n /= i;
        }

        if (n > 1) fact.push_back (n);

        for (int res=2; res<=p; ++res) {
                bool ok = true;
                for (size_t i=0; i<fact.size() && ok; ++i)
                        ok &= exp(res, (p-1) / fact[i], p) != 1;
                if (ok)  return res;
        }

        return -1;
}
```

6.23 Sieve

```
// Sieve of Erasthotenes
int p[N]; vi primes;

for (ll i = 2; i < N; ++i) if (!p[i]) {
        for (ll j = i*i; j < N; j+=i) p[j]=1;
        primes.pb(i);
}
```

6.24 Simpson rule

```
// Simpson Integration Rule
// define the function f
double f(double x) {
        // ...
}

double simpson(double a, double b, int n = 1e6) {
                double h = (b - a) / n;
                double s = f(a) + f(b);
                for (int i = 1; i < n; i += 2) s += 4 * f(a + h*i);
                for (int i = 2; i < n; i += 2) s += 2 * f(a + h*i);
                return s*h/3;
}
```

6.25 Stanford simplex

```
// Two-phase simplex algorithm for solving linear programs of the form
//
//     maximize     c^T x
//     subject to   Ax <= b
//                  x >= 0
//
// INPUT: A -- an m x n matrix
//        b -- an m-dimensional vector
//        c -- an n-dimensional vector
//        x -- a vector where the optimal solution will be stored
//
// OUTPUT: value of the optimal solution (infinity if unbounded
//         above, nan if infeasible)
//
// To use this code, create an LPSolver object with A, b, and c as
// arguments.  Then, call Solve(x).

#include <iostream>
#include <iomanip>
#include <vector>
#include <cmath>
#include <limits>

using namespace std;
typedef long double DOUBLE;
typedef vector<DOUBLE> VD;
typedef vector<VD> VVD;
typedef vector<int> VI;
const DOUBLE EPS = 1e-9;

struct LPSolver {
        int m, n;
        VI B, N;
        VVD D;

        LPSolver(const VVD &A, const VD &b, const VD &c) :
                m(b.size()), n(c.size()), N(n + 1), B(m), D(m + 2, VD(n + 2)) {
                for (int i = 0; i < m; i++) for (int j = 0; j < n; j++) D[i][j] = A[i][j];
                for (int i = 0; i < m; i++) { B[i] = n + i; D[i][n] = -1; D[i][n + 1] = b[i]; }
                for (int j = 0; j < n; j++) { N[j] = j; D[m][j] = -c[j]; }
                N[n] = -1; D[m + 1][n] = 1;
        }

        void Pivot(int r, int s) {
                for (int i = 0; i < m + 2; i++) if (i != r)
                        for (int j = 0; j < n + 2; j++) if (j != s)
                                D[i][j] -= D[r][j] * D[i][s] / D[r][s];
                for (int j = 0; j < n + 2; j++) if (j != s) D[r][j] /= D[r][s];
                for (int i = 0; i < m + 2; i++) if (i != r) D[i][s] /= -D[r][s];
                D[r][s] = 1.0 / D[r][s];
                swap(B[r], N[s]);
        }

        bool Simplex(int phase) {
                int x = phase == 1 ? m + 1 : m;
                while (true) {
                        int s = -1;
                        for (int j = 0; j <= n; j++) {
                                if (phase == 2 && N[j] == -1) continue;
                                if (s == -1 || D[x][j] < D[x][s] || D[x][j] == D[x][s] && N[j] < N[s])
                                        s = j;
                        }
                        if (D[x][s] > -EPS) return true;
                        int r = -1;
                        for (int i = 0; i < m; i++) {
                                if (D[i][s] < EPS) continue;
                                if (r == -1 || D[i][n + 1] / D[i][s] < D[r][n + 1] / D[r][s] ||
                                        (D[i][n + 1] / D[i][s]) == (D[r][n + 1] / D[r][s]) && B[i] < B
                                                [r]) r = i;
                        }
                        if (r == -1) return false;
                        Pivot(r, s);
                }
        }

        DOUBLE Solve(VD &x) {
                int r = 0;
                for (int i = 1; i < m; i++) if (D[i][n + 1] < D[r][n + 1]) r = i;
                if (D[r][n + 1] < -EPS) {
                        Pivot(r, n);
                        if (!Simplex(1) || D[m + 1][n + 1] < -EPS) return -numeric_limits<DOUBLE>::
                                infinity();
                        for (int i = 0; i < m; i++) if (B[i] == -1) {
                                int s = -1;
                                for (int j = 0; j <= n; j++)
                                        if (s == -1 || D[i][j] < D[i][s] || D[i][j] == D[i][s] && N[j]
                                                < N[s]) s = j;
                                Pivot(i, s);
                        }
                }
                if (!Simplex(2)) return numeric_limits<DOUBLE>::infinity();
                x = VD(n);
                for (int i = 0; i < m; i++) if (B[i] < n) x[B[i]] = D[i][n + 1];
                return D[m][n + 1];
        }
};

int main() {

        const int m = 4;
        const int n = 3;
        DOUBLE _A[m][n] = {
                { 6, -1, 0 },
                { -1, -5, 0 },
                { 1, 5, 1 },
                { -1, -5, -1 }
        };
        DOUBLE _b[m] = { 10, -4, 5, -5 };
        DOUBLE _c[n] = { 1, -1, 0 };

        VVD A(m);
        VD b(_b, _b + m);
        VD c(_c, _c + n);
        for (int i = 0; i < m; i++) A[i] = VD(_A[i], _A[i] + n);

        LPSolver solver(A, b, c);
        VD x;
        DOUBLE value = solver.Solve(x);
```

```
        cerr << "VALUE: " << value << endl; // VALUE: 1.29032
        cerr << "SOLUTION:"; // SOLUTION: 1.74194 0.451613 1
        for (size_t i = 0; i < x.size(); i++) cerr << " " << x[i];
        cerr << endl;
        return 0;
}
```

7 Strings

7.1 KMP

```
vi kmp_builder(string &s, int n) {
  vi dp(n, 0);
  int j = 0;
  forx(i, 1, n) {
    while (j && s[i] != s[j]) j = dp[j - 1];

    if (s[i] == s[j]) dp[i] = ++j;
    else dp[i] = 0;
  }

  return dp;
}

// Return all occurrences of the pattern in the text
vi kmp(string &t, string &p) {
  string q = p + "#" + t;
  vi v = kmp_builder(q, sz(q));
  vi res;
  forn(i, sz(q)) if (v[i] == sz(p)) res.pb(i - 2 * sz(p) + 1);

  return res;
}
```

7.2 Algorithm Z

```
// Example answer aabb#aaxnaabba -> 01000210041001
vi alz(const string &s) // pattern#where_to_look
{
        int n = s.size();
        vi z(n, 0);
        for(int i = 1, l = 0, r = 0; i < n; i++)
        {
                if(i <= r)
                        z[i] = min(z[i - l], r - i + 1);
                while(i + z[i] < n && s[z[i]] == s[i + z[i]])
                        z[i]++;
                if(r < i + z[i] - 1)
                        l = i, r = i + z[i] - 1;
        }
        return z;
}
```

7.3 Rabin Karp

```
const ll mod[2] = {1000000007, 998244353};
const ll px[2] = {29, 31};

vl rabin_karp(string &s, string &p) {
  vl ss[2], pp[2], ppx[2];
  for (ll i = 0; i < 2; i++)
    ss[i] = rolling_hash(s, px[i], mod[i]),
    pp[i] = rolling_hash(p, px[i], mod[i]);

  vi res;
  for (int i = 0; i + sz(p) - 1 < sz(s); i++) {
    ll ok = 1;
    for (ll j = 0; j < 2; j++) {
      int fh = fast_hash(ss[j], px[j], mod[j], i, i + sz(p) - 1) % mod[j];
      ok &= (fh == pp[j].back());
    }
    if (ok) res.pb(i + 1);
  }

  return res;
}
```

7.4 Aho-Corasick

```
const int K = 26;

struct Vertex {
    int next[K];
    bool output = false;
    int p = -1;
    char pch;
    int link = -1;
    int go[K];

    Vertex(int p=-1, char ch='$') : p(p), pch(ch) {
        fill(begin(next), end(next), -1);
        fill(begin(go), end(go), -1);
    }
};

vector<Vertex> t(1);

void aho_init() {
  t.clear();
  t.pb(Vertex());
}

void add_string(string const& s) {
    int v = 0;
    for (char ch : s) {
        int c = ch - 'a';
        if (t[v].next[c] == -1) {
            t[v].next[c] = t.size();
            t.emplace_back(v, ch);
        }
        v = t[v].next[c];
    }
    t[v].output = true;
}

int go(int v, char ch);

int get_link(int v) {
    if (t[v].link == -1) {
        if (v == 0 || t[v].p == 0)
            t[v].link = 0;
        else
            t[v].link = go(get_link(t[v].p), t[v].pch);
    }
    return t[v].link;
}

int go(int v, char ch) {
    int c = ch - 'a';
    if (t[v].go[c] == -1) {
        if (t[v].next[c] != -1)
            t[v].go[c] = t[v].next[c];
        else
            t[v].go[c] = v == 0 ? 0 : go(get_link(v), ch);
    }
    return t[v].go[c];
}

vector<int> search_in_text(const string& text) {
  vector<int> occurrences;
  int v = 0;
  for (int i = 0; i < text.size(); i++) {
    char ch = text[i];
    v = go(v, ch);

    for (int u = v; u != 0; u = get_link(u)) {
      if (t[u].output) {
        occurrences.push_back(i);
      }
    }
  }

  return occurrences;
}
```

7.5 Hashing

```
const int K = 2;
struct Hash {
  const ll MOD[K] = {999727999, 1070777777};
  const ll P = 1777771;
```

```
  vector<ll> h[K], p[K];
  Hash(string &s) {
    int n = s.size();
    for(int k = 0; k < K; k++) {
      h[k].resize(n + 1, 0);
      p[k].resize(n + 1, 1);
      for(int i = 1; i <= n; i++) {
        h[k][i] = (h[k][i - 1] * P + s[i - 1]) % MOD[k];
        p[k][i] = (p[k][i - 1] * P) % MOD[k];
      }
    }
  }
  vector<ll> get(int i, int j) { // hash [i, j]
    j++;
    vector<ll> r(K);
    for(int k = 0; k < K; k++) {
      r[k] = (h[k][j] - h[k][i] * p[k][j - i]) % MOD[k];
      r[k] = (r[k] + MOD[k]) % MOD[k];
    }
    return r;
  }
};

// Other
ll pow(ll b, ll e, ll m) {
  ll res = 1;
  for (; e; e >>= 1, b = (b * b) % m)
  if (e & 1) res = (res * b) % m;
  return res;
}

ll inv(ll b, ll e, ll m) {
  return pow(pow(b, e, m), m - 2, m);
}

vl rolling_hash(string &s, ll p, ll m) {
  ll n = sz(s);
  vl v(n, 0);
  v[0] = (s[0]) % m;
  for (ll i = 1; i < n; i++)
    v[i] = (v[i - 1] + (s[i] * pow(p, i, m)) % m) % m;

  return v;
}

ll fast_hash(vl &v, ll p, ll m, ll i, ll j) {
  return (((v[j] - (i ? v[i - 1] : 0) + m) % m) * inv(p, i, m)) % m;
}

// Hash 128
#define bint __int128
struct Hash {
  bint MOD=212345678987654321LL,P=1777771,PI=106955741089659571LL;
  vector<bint> h,pi;
  Hash(string& s){
    assert((P*PI)%MOD==1);
    h.resize(s.size()+1);pi.resize(s.size()+1);
    h[0]=0;pi[0]=1;
    bint p=1;
    forx(i,1,s.size()+1){
      h[i]=(h[i-1]+p*s[i-1])%MOD;
      pi[i]=(pi[i-1]*PI)%MOD;
      p=(p*P)%MOD;
    }
  }
  ll get(int s, int e){
    return (((h[e]-h[s]+MOD)%MOD)*pi[s])%MOD;
  }
};
```

7.6 Manacher

```
/* Find palindromes in a string
f = 1 para pares, 0 impar
a a a a a a
1 2 3 3 2 1   f = 0 impar
0 1 2 3 2 1   f = 1 par centrado entre [i-1,i]
Time: O(n)
*/
void manacher(string &s, int f, vi &d) {
  int l = 0, r = -1, n = s.size();
  d.assign(n, 0);
  for (int i = 0; i < n; i++) {
    int k = (i > r ? (1 - f) : min(d[l + r - i + f], r - i + f)) + f;
    while (i + k - f < n && i - k >= 0 && s[i + k - f] == s[i - k]) ++k;
    d[i] = k - f; --k;
    if (i + k - f > r) l = i - k, r = i + k - f;
  }
}
```

7.7 Suffix Array

```
struct suffix {
        int index;
        int rank[2];
};

int cmp(struct suffix a, struct suffix b) {
        return (a.rank[0] == b.rank[0])? (a.rank[1] < b.rank[1] ?1: 0):
                    (a.rank[0] < b.rank[0] ?1: 0);
}

int *buildSuffixArray(char *txt, int n) {
        struct suffix suffixes[n];

        for (int i = 0; i < n; i++) {
                suffixes[i].index = i;
                suffixes[i].rank[0] = txt[i] - 'a';
                suffixes[i].rank[1] = ((i+1) < n)? (txt[i + 1] - 'a'): -1;
        }

        sort(suffixes, suffixes+n, cmp);

        int ind[n];
        for (int k = 4; k < 2*n; k = k*2)       {
                int rank = 0;
                int prev_rank = suffixes[0].rank[0];
                suffixes[0].rank[0] = rank;
                ind[suffixes[0].index] = 0;

                for (int i = 1; i < n; i++) {
                        if (suffixes[i].rank[0] == prev_rank &&
                                        suffixes[i].rank[1] == suffixes[i-1].rank[1]) {
                                prev_rank = suffixes[i].rank[0];
                                suffixes[i].rank[0] = rank;
                        } else {
                                prev_rank = suffixes[i].rank[0];
                                suffixes[i].rank[0] = ++rank;
                        }
                        ind[suffixes[i].index] = i;
                }

                for (int i = 0; i < n; i++) {
                        int nextindex = suffixes[i].index + k/2;
                        suffixes[i].rank[1] = (nextindex < n)?
                                                        suffixes[ind[nextindex]].rank[0]: -1;
                }

                sort(suffixes, suffixes+n, cmp);
        }

        int *suffixArr = new int[n];
        for (int i = 0; i < n; i++)
                suffixArr[i] = suffixes[i].index;

        return suffixArr;
}

void printArr(int arr[], int n)
{
        for (int i = 0; i < n; i++)
                cout << arr[i] << " ";
        cout << endl;
}

void solve() {
        char txt[] = "banana";
        int n = strlen(txt);
        int *suffixArr = buildSuffixArray(txt, n);
        cout << "Following is suffix array for " << txt << endl;
        printArr(suffixArr, n);
}
```

8 Others

8.1 Grundy (Nim Game)

```
#define PLAYER1 1
```

```
#define PLAYER2 2

int calculate_mex(unordered_set<int> my_set) {
        int mex = 0;
        while (my_set.find(mex) != my_set.end()) mex++;
        return mex;
}

int calculate_grundy(int n, int grundy[]) {
        grundy[0] = 0;
        if (grundy[n] != -1) return (grundy[n]);

        unordered_set<int> my_set
        for (int i = 3; i <= 5; i++) // Range of numbers of items we can take
    my_set.insert(calculate_grundy(n - i, grundy));

        grundy[n] = calculate_mex(my_set);
        return grundy[n];
}

void declare_winner(int whoseTurn, int piles[],
                                        int grundy[], int n) {
        int xorValue = grundy[piles[0]];
        for (int i = 1; i <= n - 1; i++)
                xorValue = xorValue ^ grundy[piles[i]];

        if (xorValue != 0) {
                if (whoseTurn == PLAYER1)
                        printf("Player 1 will win\n");
                else
                        printf("Player 2 will win\n");
        } else {
                if (whoseTurn == PLAYER1)
                        printf("Player 2 will win\n");
                else
                        printf("Player 1 will win\n");
        }
}

void solve() {
        // Each of the piles is a sub game
        int piles[] = {12 + 34 + 11 + 1 + 23};
        int n = sizeof(piles) / sizeof(piles[0]);

        int maximum = *max_element(piles, piles + n);
        int grundy[maximum + 1];
        memset(grundy, -1, sizeof(grundy));

        for (int i = 0; i <= n - 1; i++)
                calculate_grundy(piles[i], grundy);

        declareWinner(PLAYER1, piles, Grundy, n);
}
```

$f(n) = O(g(n))$	iff $\exists$ positive c, n_0 such that $0 \le f(n) \le cg(n)$ $\forall n \ge n_0$.
$f(n) = \Omega(g(n))$	iff $\exists$ positive c, n_0 such that $f(n) \ge cg(n) \ge 0$ $\forall n \ge n_0$.
$f(n) = \Theta(g(n))$	iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.
$f(n) = o(g(n))$	iff $\lim_{n\to\infty} f(n)/g(n) = 0$.
$\lim\limits_{n\to\infty} a_n = a$	iff $\forall \epsilon > 0$, $\exists n_0$ such that $\|a_n - a\| < \epsilon$, $\forall n \ge n_0$.
$\sup S$	least $b \in \mathbb{R}$ such that $b \ge s$, $\forall s \in S$.
$\inf S$	greatest $b \in \mathbb{R}$ such that $b \le s$, $\forall s \in S$.
$\liminf\limits_{n\to\infty} a_n$	$\lim\limits_{n\to\infty} \inf\{a_i \mid i \ge n, i \in \mathbb{N}\}$.
$\limsup\limits_{n\to\infty} a_n$	$\lim\limits_{n\to\infty} \sup\{a_i \mid i \ge n, i \in \mathbb{N}\}$.
$\binom{n}{k}$	Combinations: Size k subsets of a size n set.
$\left[n \atop k \right]$	Stirling numbers (1st kind): Arrangements of an n element set into k cycles.
$\left\{ n \atop k \right\}$	Stirling numbers (2nd kind): Partitions of an n element set into k non-empty sets.
$\left\langle n \atop k \right\rangle$	1st order Eulerian numbers: Permutations $\pi_1\pi_2\ldots\pi_n$ on $\{1, 2, \ldots, n\}$ with k ascents.
$\left\langle\!\left\langle n \atop k \right\rangle\!\right\rangle$	2nd order Eulerian numbers.
C_n	Catalan Numbers: Binary trees with $n+1$ vertices.

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}, \qquad \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}, \qquad \sum_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4}.$$

In general:

$$\sum_{i=1}^{n} i^m = \frac{1}{m+1}\left[(n+1)^{m+1} - 1 - \sum_{i=1}^{n}\left((i+1)^{m+1} - i^{m+1} - (m+1)i^m\right)\right]$$

$$\sum_{i=1}^{n-1} i^m = \frac{1}{m+1}\sum_{k=0}^{m}\binom{m+1}{k} B_k n^{m+1-k}.$$

Geometric series:

$$\sum_{i=0}^{n} c^i = \frac{c^{n+1}-1}{c-1}, \quad c \ne 1, \qquad \sum_{i=0}^{\infty} c^i = \frac{1}{1-c}, \qquad \sum_{i=1}^{\infty} c^i = \frac{c}{1-c}, \quad |c| < 1,$$

$$\sum_{i=0}^{n} ic^i = \frac{nc^{n+2} - (n+1)c^{n+1} + c}{(c-1)^2}, \quad c \ne 1, \qquad \sum_{i=0}^{\infty} ic^i = \frac{c}{(1-c)^2}, \quad |c| < 1.$$

Harmonic series:

$$H_n = \sum_{i=1}^{n}\frac{1}{i}, \qquad \sum_{i=1}^{n} iH_i = \frac{n(n+1)}{2}H_n - \frac{n(n-1)}{4}.$$

$$\sum_{i=1}^{n} H_i = (n+1)H_n - n, \qquad \sum_{i=1}^{n}\binom{i}{m}H_i = \binom{n+1}{m+1}\left(H_{n+1} - \frac{1}{m+1}\right).$$

1. $\displaystyle\binom{n}{k} = \frac{n!}{(n-k)!k!}$, **2.** $\displaystyle\sum_{k=0}^{n}\binom{n}{k} = 2^n$, **3.** $\displaystyle\binom{n}{k} = \binom{n}{n-k}$,

4. $\displaystyle\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$, **5.** $\displaystyle\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$,

6. $\displaystyle\binom{n}{m}\binom{m}{k} = \binom{n}{k}\binom{n-k}{m-k}$, **7.** $\displaystyle\sum_{k=0}^{n}\binom{r+k}{k} = \binom{r+n+1}{n}$,

8. $\displaystyle\sum_{k=0}^{n}\binom{k}{m} = \binom{n+1}{m+1}$, **9.** $\displaystyle\sum_{k=0}^{n}\binom{r}{k}\binom{s}{n-k} = \binom{r+s}{n}$,

10. $\displaystyle\binom{n}{k} = (-1)^k\binom{k-n-1}{k}$, **11.** $\displaystyle\left\{ n \atop 1 \right\} = \left\{ n \atop n \right\} = 1$,

12. $\displaystyle\left\{ n \atop 2 \right\} = 2^{n-1} - 1$, **13.** $\displaystyle\left\{ n \atop k \right\} = k\left\{ n-1 \atop k \right\} + \left\{ n-1 \atop k-1 \right\}$,

14. $\displaystyle\left[n \atop 1 \right] = (n-1)!$, **15.** $\displaystyle\left[n \atop 2 \right] = (n-1)!H_{n-1}$, **16.** $\displaystyle\left[n \atop n \right] = 1$, **17.** $\displaystyle\left[n \atop k \right] \ge \left\{ n \atop k \right\}$,

18. $\displaystyle\left[n \atop k \right] = (n-1)\left[n-1 \atop k \right] + \left[n-1 \atop k-1 \right]$, **19.** $\displaystyle\left\{ n \atop n-1 \right\} = \left[n \atop n-1 \right] = \binom{n}{2}$, **20.** $\displaystyle\sum_{k=0}^{n}\left[n \atop k \right] = n!$, **21.** $\displaystyle C_n = \frac{1}{n+1}\binom{2n}{n}$,

22. $\displaystyle\left\langle n \atop 0 \right\rangle = \left\langle n \atop n-1 \right\rangle = 1$, **23.** $\displaystyle\left\langle n \atop k \right\rangle = \left\langle n \atop n-1-k \right\rangle$, **24.** $\displaystyle\left\langle n \atop k \right\rangle = (k+1)\left\langle n-1 \atop k \right\rangle + (n-k)\left\langle n-1 \atop k-1 \right\rangle$,

25. $\displaystyle\left\langle 0 \atop k \right\rangle = \begin{cases} 1 & \text{if } k = 0, \\ 0 & \text{otherwise} \end{cases}$ **26.** $\displaystyle\left\langle n \atop 1 \right\rangle = 2^n - n - 1$, **27.** $\displaystyle\left\langle n \atop 2 \right\rangle = 3^n - (n+1)2^n + \binom{n+1}{2}$,

28. $\displaystyle x^n = \sum_{k=0}^{n}\left\langle n \atop k \right\rangle\binom{x+k}{n}$, **29.** $\displaystyle\left\langle n \atop m \right\rangle = \sum_{k=0}^{m}\binom{n+1}{k}(m+1-k)^n(-1)^k$, **30.** $\displaystyle m!\left\{ n \atop m \right\} = \sum_{k=0}^{n}\left\langle n \atop k \right\rangle\binom{k}{n-m}$,

31. $\displaystyle\left\langle n \atop m \right\rangle = \sum_{k=0}^{n}\left\{ n \atop k \right\}\binom{n-k}{m}(-1)^{n-k-m}k!$, **32.** $\displaystyle\left\langle\!\left\langle n \atop 0 \right\rangle\!\right\rangle = 1$, **33.** $\displaystyle\left\langle\!\left\langle n \atop n \right\rangle\!\right\rangle = 0$ for $n \ne 0$,

34. $\displaystyle\left\langle\!\left\langle n \atop k \right\rangle\!\right\rangle = (k+1)\left\langle\!\left\langle n-1 \atop k \right\rangle\!\right\rangle + (2n-1-k)\left\langle\!\left\langle n-1 \atop k-1 \right\rangle\!\right\rangle$, **35.** $\displaystyle\sum_{k=0}^{n}\left\langle\!\left\langle n \atop k \right\rangle\!\right\rangle = \frac{(2n)^{\underline{n}}}{2^n}$,

36. $\displaystyle\left\{ x \atop x-n \right\} = \sum_{k=0}^{n}\left\langle\!\left\langle n \atop k \right\rangle\!\right\rangle\binom{x+n-1-k}{2n}$, **37.** $\displaystyle\left\{ n+1 \atop m+1 \right\} = \sum_{k}\binom{n}{k}\left\{ k \atop m \right\} = \sum_{k=0}^{n}\left\{ k \atop m \right\}(m+1)^{n-k}$,

38. $\left[{n+1 \atop m+1}\right] = \sum_k \left[{n \atop k}\right]\binom{k}{m} = \sum_{k=0}^{n} \left[{k \atop m}\right] n^{\underline{n-k}} = n! \sum_{k=0}^{n} \frac{1}{k!}\left[{k \atop m}\right],$

39. $\left[{x \atop x-n}\right] = \sum_{k=0}^{n} \left\langle\!\!\left\langle {n \atop k} \right\rangle\!\!\right\rangle \binom{x+k}{2n},$

40. $\left\{{n \atop m}\right\} = \sum_k \binom{n}{k}\left\{{k+1 \atop m+1}\right\}(-1)^{n-k},$

41. $\left[{n \atop m}\right] = \sum_k \left[{n+1 \atop k+1}\right]\binom{k}{m}(-1)^{m-k},$

42. $\left\{{m+n+1 \atop m}\right\} = \sum_{k=0}^{m} k\left\{{n+k \atop k}\right\},$

43. $\left[{m+n+1 \atop m}\right] = \sum_{k=0}^{m} k(n+k)\left[{n+k \atop k}\right],$

44. $\binom{n}{m} = \sum_k \left\{{n+1 \atop k+1}\right\}\left[{k \atop m}\right](-1)^{m-k},$

45. $(n-m)!\binom{n}{m} = \sum_k \left[{n+1 \atop k+1}\right]\left\{{k \atop m}\right\}(-1)^{m-k}, \quad \text{for } n \ge m,$

46. $\left\{{n \atop n-m}\right\} = \sum_k \binom{m-n}{m+k}\binom{m+n}{n+k}\left[{m+k \atop k}\right],$

47. $\left[{n \atop n-m}\right] = \sum_k \binom{m-n}{m+k}\binom{m+n}{n+k}\left\{{m+k \atop k}\right\},$

48. $\left\{{n \atop \ell+m}\right\}\binom{\ell+m}{\ell} = \sum_k \left\{{k \atop \ell}\right\}\left\{{n-k \atop m}\right\}\binom{n}{k},$

49. $\left[{n \atop \ell+m}\right]\binom{\ell+m}{\ell} = \sum_k \left[{k \atop \ell}\right]\left[{n-k \atop m}\right]\binom{n}{k}.$

Every tree with n vertices has $n-1$ edges.

Kraft inequality: If the depths of the leaves of a binary tree are $d_1, \ldots, d_n$:
$$\sum_{i=1}^{n} 2^{-d_i} \le 1,$$
and equality holds only if every internal node has 2 sons.

Recurrences

Master method:
$$T(n) = aT(n/b) + f(n), \quad a \ge 1, b > 1$$

If $\exists \epsilon > 0$ such that $f(n) = O(n^{\log_b a - \epsilon})$ then
$$T(n) = \Theta(n^{\log_b a}).$$

If $f(n) = \Theta(n^{\log_b a})$ then
$$T(n) = \Theta(n^{\log_b a} \log_2 n).$$

If $\exists \epsilon > 0$ such that $f(n) = \Omega(n^{\log_b a + \epsilon})$, and $\exists c < 1$ such that $af(n/b) \le cf(n)$ for large n, then
$$T(n) = \Theta(f(n)).$$

Substitution (example): Consider the following recurrence
$$T_{i+1} = 2^{2^i} \cdot T_i^2, \quad T_1 = 2.$$

Note that T_i is always a power of two. Let $t_i = \log_2 T_i$. Then we have
$$t_{i+1} = 2^i + 2t_i, \quad t_1 = 1.$$

Let $u_i = t_i/2^i$. Dividing both sides of the previous equation by 2^{i+1} we get
$$\frac{t_{i+1}}{2^{i+1}} = \frac{2^i}{2^{i+1}} + \frac{t_i}{2^i}.$$

Substituting we find
$$u_{i+1} = \tfrac{1}{2} + u_i, \qquad u_1 = \tfrac{1}{2},$$

which is simply $u_i = i/2$. So we find that T_i has the closed form $T_i = 2^{i2^{i-1}}$.

Summing factors (example): Consider the following recurrence
$$T(n) = 3T(n/2) + n, \quad T(1) = 1.$$

Rewrite so that all terms involving T are on the left side
$$T(n) - 3T(n/2) = n.$$

Now expand the recurrence, and choose a factor which makes the left side "telescope"

$$1\big(T(n) - 3T(n/2) = n\big)$$
$$3\big(T(n/2) - 3T(n/4) = n/2\big)$$
$$\vdots \quad \vdots \quad \vdots$$
$$3^{\log_2 n - 1}\big(T(2) - 3T(1) = 2\big)$$

Let $m = \log_2 n$. Summing the left side we get $T(n) - 3^m T(1) = T(n) - 3^m = T(n) - n^k$ where $k = \log_2 3 \approx 1.58496$. Summing the right side we get
$$\sum_{i=0}^{m-1} \frac{n}{2^i} 3^i = n \sum_{i=0}^{m-1} \left(\tfrac{3}{2}\right)^i.$$

Let $c = \frac{3}{2}$. Then we have
$$\begin{aligned} n \sum_{i=0}^{m-1} c^i &= n\left(\frac{c^m - 1}{c - 1}\right) \\ &= 2n(c^{\log_2 n} - 1) \\ &= 2n(c^{(k-1)\log_c n} - 1) \\ &= 2n^k - 2n, \end{aligned}$$

and so $T(n) = 3n^k - 2n$. Full history recurrences can often be changed to limited history ones (example): Consider
$$T_i = 1 + \sum_{j=0}^{i-1} T_j, \quad T_0 = 1.$$

Note that
$$T_{i+1} = 1 + \sum_{j=0}^{i} T_j.$$

Subtracting we find
$$\begin{aligned} T_{i+1} - T_i &= 1 + \sum_{j=0}^{i} T_j - 1 - \sum_{j=0}^{i-1} T_j \\ &= T_i. \end{aligned}$$

And so $T_{i+1} = 2T_i = 2^{i+1}$.

Generating functions:

1. Multiply both sides of the equation by x^i.
2. Sum both sides over all i for which the equation is valid.
3. Choose a generating function $G(x)$. Usually $G(x) = \sum_{i=0}^{\infty} x^i g_i$.
3. Rewrite the equation in terms of the generating function $G(x)$.
4. Solve for $G(x)$.
5. The coefficient of x^i in $G(x)$ is g_i.

Example:
$$g_{i+1} = 2g_i + 1, \quad g_0 = 0.$$

Multiply and sum:
$$\sum_{i \ge 0} g_{i+1} x^i = \sum_{i \ge 0} 2g_i x^i + \sum_{i \ge 0} x^i.$$

We choose $G(x) = \sum_{i \ge 0} x^i g_i$. Rewrite in terms of $G(x)$:
$$\frac{G(x) - g_0}{x} = 2G(x) + \sum_{i \ge 0} x^i.$$

Simplify:
$$\frac{G(x)}{x} = 2G(x) + \frac{1}{1-x}.$$

Solve for $G(x)$:
$$G(x) = \frac{x}{(1-x)(1-2x)}.$$

Expand this using partial fractions:
$$\begin{aligned} G(x) &= x\left(\frac{2}{1-2x} - \frac{1}{1-x}\right) \\ &= x\left(2\sum_{i \ge 0} 2^i x^i - \sum_{i \ge 0} x^i\right) \\ &= \sum_{i \ge 0} (2^{i+1} - 1) x^{i+1}. \end{aligned}$$

So $g_i = 2^i - 1$.

$\pi \approx 3.14159$, $e \approx 2.71828$, $\gamma \approx 0.57721$, $\phi = \frac{1+\sqrt{5}}{2} \approx 1.61803$, $\hat{\phi} = \frac{1-\sqrt{5}}{2} \approx -.61803$

i	2^i	p_i
1	2	2
2	4	3
3	8	5
4	16	7
5	32	11
6	64	13
7	128	17
8	256	19
9	512	23
10	1,024	29
11	2,048	31
12	4,096	37
13	8,192	41
14	16,384	43
15	32,768	47
16	65,536	53
17	131,072	59
18	262,144	61
19	524,288	67
20	1,048,576	71
21	2,097,152	73
22	4,194,304	79
23	8,388,608	83
24	16,777,216	89
25	33,554,432	97
26	67,108,864	101
27	134,217,728	103
28	268,435,456	107
29	536,870,912	109
30	1,073,741,824	113
31	2,147,483,648	127
32	4,294,967,296	131

Pascal's Triangle

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
1 10 45 120 210 252 210 120 45 10 1

General

Bernoulli Numbers ($B_i = 0$, odd $i \neq 1$):

$$B_0 = 1,\ B_1 = -\tfrac{1}{2},\ B_2 = \tfrac{1}{6},\ B_4 = -\tfrac{1}{30},\ B_6 = \tfrac{1}{42},\ B_8 = -\tfrac{1}{30},\ B_{10} = \tfrac{5}{66}.$$

Change of base, quadratic formula:

$$\log_b x = \frac{\log_a x}{\log_a b}, \qquad \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Euler's number e:

$$e = 1 + \tfrac{1}{2} + \tfrac{1}{6} + \tfrac{1}{24} + \tfrac{1}{120} + \cdots$$

$$\lim_{n\to\infty} \left(1 + \frac{x}{n}\right)^n = e^x.$$

$$\left(1 + \tfrac{1}{n}\right)^n < e < \left(1 + \tfrac{1}{n}\right)^{n+1}.$$

$$\left(1 + \tfrac{1}{n}\right)^n = e - \frac{e}{2n} + \frac{11e}{24n^2} - O\left(\frac{1}{n^3}\right).$$

Harmonic numbers:

$$1,\ \tfrac{3}{2},\ \tfrac{11}{6},\ \tfrac{25}{12},\ \tfrac{137}{60},\ \tfrac{49}{20},\ \tfrac{363}{140},\ \tfrac{761}{280},\ \tfrac{7129}{2520},\ \ldots$$

$$\ln n < H_n < \ln n + 1,$$

$$H_n = \ln n + \gamma + O\left(\frac{1}{n}\right).$$

Factorial, Stirling's approximation:

$$1,\ 2,\ 6,\ 24,\ 120,\ 720,\ 5040,\ 40320,\ 362880,\ \ldots$$

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right).$$

Ackermann's function and inverse:

$$a(i,j) = \begin{cases} 2^j & i = 1 \\ a(i-1, 2) & j = 1 \\ a(i-1, a(i, j-1)) & i, j \geq 2 \end{cases}$$

$$\alpha(i) = \min\{j \mid a(j,j) \geq i\}.$$

Binomial distribution:

$$\Pr[X = k] = \binom{n}{k} p^k q^{n-k}, \qquad q = 1 - p,$$

$$\mathrm{E}[X] = \sum_{k=1}^{n} k \binom{n}{k} p^k q^{n-k} = np.$$

Poisson distribution:

$$\Pr[X = k] = \frac{e^{-\lambda}\lambda^k}{k!}, \quad \mathrm{E}[X] = \lambda.$$

Normal (Gaussian) distribution:

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2}, \quad \mathrm{E}[X] = \mu.$$

The "coupon collector": We are given a random coupon each day, and there are n different types of coupons. The distribution of coupons is uniform. The expected number of days to pass before we to collect all n types is

$$nH_n.$$

Probability

Continuous distributions: If

$$\Pr[a < X < b] = \int_a^b p(x)\, dx,$$

then p is the probability density function of X. If

$$\Pr[X < a] = P(a),$$

then P is the distribution function of X. If P and p both exist then

$$P(a) = \int_{-\infty}^{a} p(x)\, dx.$$

Expectation: If X is discrete

$$\mathrm{E}[g(X)] = \sum_x g(x) \Pr[X = x].$$

If X continuous then

$$\mathrm{E}[g(X)] = \int_{-\infty}^{\infty} g(x)p(x)\, dx = \int_{-\infty}^{\infty} g(x)\, dP(x).$$

Variance, standard deviation:

$$\mathrm{VAR}[X] = \mathrm{E}[X^2] - \mathrm{E}[X]^2,$$

$$\sigma = \sqrt{\mathrm{VAR}[X]}.$$

For events A and B:

$$\Pr[A \vee B] = \Pr[A] + \Pr[B] - \Pr[A \wedge B]$$

$$\Pr[A \wedge B] = \Pr[A] \cdot \Pr[B],$$

iff A and B are independent.

$$\Pr[A|B] = \frac{\Pr[A \wedge B]}{\Pr[B]}$$

For random variables X and Y:

$$\mathrm{E}[X \cdot Y] = \mathrm{E}[X] \cdot \mathrm{E}[Y],$$

if X and Y are independent.

$$\mathrm{E}[X + Y] = \mathrm{E}[X] + \mathrm{E}[Y],$$

$$\mathrm{E}[cX] = c\,\mathrm{E}[X].$$

Bayes' theorem:

$$\Pr[A_i|B] = \frac{\Pr[B|A_i]\Pr[A_i]}{\sum_{j=1}^{n} \Pr[A_j]\Pr[B|A_j]}.$$

Inclusion-exclusion:

$$\Pr\left[\bigvee_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \Pr[X_i] + \sum_{k=2}^{n} (-1)^{k+1} \sum_{i_i < \cdots < i_k} \Pr\left[\bigwedge_{j=1}^{k} X_{i_j}\right].$$

Moment inequalities:

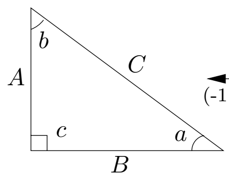
$$\Pr\left[|X| \geq \lambda\,\mathrm{E}[X]\right] \leq \frac{1}{\lambda},$$

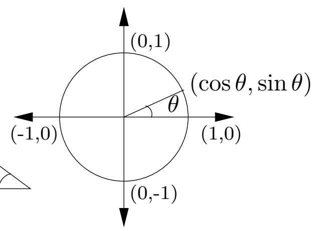
$$\Pr\left[\left|X - \mathrm{E}[X]\right| \geq \lambda \cdot \sigma\right] \leq \frac{1}{\lambda^2}.$$

Geometric distribution:

$$\Pr[X = k] = pq^{k-1}, \qquad q = 1 - p,$$

$$\mathrm{E}[X] = \sum_{k=1}^{\infty} kpq^{k-1} = \frac{1}{p}.$$





Pythagorean theorem:

$$C^2 = A^2 + B^2.$$

Definitions:

$$\sin a = A/C, \quad \cos a = B/C,$$
$$\csc a = C/A, \quad \sec a = C/B,$$
$$\tan a = \frac{\sin a}{\cos a} = \frac{A}{B}, \quad \cot a = \frac{\cos a}{\sin a} = \frac{B}{A}.$$

Area, radius of inscribed circle:

$$\tfrac{1}{2}AB, \quad \frac{AB}{A+B+C}.$$

Identities:

$$\sin x = \frac{1}{\csc x}, \qquad \cos x = \frac{1}{\sec x},$$
$$\tan x = \frac{1}{\cot x}, \qquad \sin^2 x + \cos^2 x = 1,$$
$$1 + \tan^2 x = \sec^2 x, \qquad 1 + \cot^2 x = \csc^2 x,$$
$$\sin x = \cos\left(\tfrac{\pi}{2} - x\right), \qquad \sin x = \sin(\pi - x),$$
$$\cos x = -\cos(\pi - x), \qquad \tan x = \cot\left(\tfrac{\pi}{2} - x\right),$$
$$\cot x = -\cot(\pi - x), \qquad \csc x = \cot \tfrac{x}{2} - \cot x,$$
$$\sin(x \pm y) = \sin x \cos y \pm \cos x \sin y,$$
$$\cos(x \pm y) = \cos x \cos y \mp \sin x \sin y,$$
$$\tan(x \pm y) = \frac{\tan x \pm \tan y}{1 \mp \tan x \tan y},$$
$$\cot(x \pm y) = \frac{\cot x \cot y \mp 1}{\cot x \pm \cot y},$$
$$\sin 2x = 2 \sin x \cos x, \qquad \sin 2x = \frac{2\tan x}{1 + \tan^2 x},$$
$$\cos 2x = \cos^2 x - \sin^2 x, \qquad \cos 2x = 2\cos^2 x - 1,$$
$$\cos 2x = 1 - 2\sin^2 x, \qquad \cos 2x = \frac{1 - \tan^2 x}{1 + \tan^2 x},$$
$$\tan 2x = \frac{2\tan x}{1 - \tan^2 x}, \qquad \cot 2x = \frac{\cot^2 x - 1}{2\cot x},$$
$$\sin(x+y)\sin(x-y) = \sin^2 x - \sin^2 y,$$
$$\cos(x+y)\cos(x-y) = \cos^2 x - \sin^2 y.$$

Euler's equation:

$$e^{ix} = \cos x + i \sin x, \qquad e^{i\pi} = -1.$$

Multiplication:

$$C = A \cdot B, \quad c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}.$$

Determinants: $\det A \neq 0$ iff A is non-singular.

$$\det A \cdot B = \det A \cdot \det B,$$
$$\det A = \sum_{\pi} \prod_{i=1}^{n} \operatorname{sign}(\pi) a_{i,\pi(i)}.$$

2×2 and 3×3 determinant:

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc,$$
$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = g \begin{vmatrix} b & c \\ e & f \end{vmatrix} - h \begin{vmatrix} a & c \\ d & f \end{vmatrix} + i \begin{vmatrix} a & b \\ d & e \end{vmatrix}$$
$$= \begin{matrix} aei + bfg + cdh \\ -\, ceg - fha - ibd. \end{matrix}$$

Permanents:

$$\operatorname{perm} A = \sum_{\pi} \prod_{i=1}^{n} a_{i,\pi(i)}.$$

Hyperbolic Functions

Definitions:

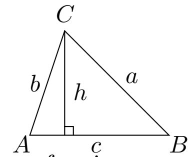
$$\sinh x = \frac{e^x - e^{-x}}{2}, \qquad \cosh x = \frac{e^x + e^{-x}}{2},$$
$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \qquad \operatorname{csch} x = \frac{1}{\sinh x},$$
$$\operatorname{sech} x = \frac{1}{\cosh x}, \qquad \coth x = \frac{1}{\tanh x}.$$

Identities:

$$\cosh^2 x - \sinh^2 x = 1, \qquad \tanh^2 x + \operatorname{sech}^2 x = 1,$$
$$\coth^2 x - \operatorname{csch}^2 x = 1, \qquad \sinh(-x) = -\sinh x,$$
$$\cosh(-x) = \cosh x, \qquad \tanh(-x) = -\tanh x,$$
$$\sinh(x+y) = \sinh x \cosh y + \cosh x \sinh y,$$
$$\cosh(x+y) = \cosh x \cosh y + \sinh x \sinh y,$$
$$\sinh 2x = 2\sinh x \cosh x,$$
$$\cosh 2x = \cosh^2 x + \sinh^2 x,$$
$$\cosh x + \sinh x = e^x, \qquad \cosh x - \sinh x = e^{-x},$$
$$(\cosh x + \sinh x)^n = \cosh nx + \sinh nx, \quad n \in \mathbb{Z},$$
$$2\sinh^2 \tfrac{x}{2} = \cosh x - 1, \qquad 2\cosh^2 \tfrac{x}{2} = \cosh x + 1.$$

θ	$\sin\theta$	$\cos\theta$	$\tan\theta$
0	0	1	0
$\frac{\pi}{6}$	$\frac{1}{2}$	$\frac{\sqrt{3}}{2}$	$\frac{\sqrt{3}}{3}$
$\frac{\pi}{4}$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{2}}{2}$	1
$\frac{\pi}{3}$	$\frac{\sqrt{3}}{2}$	$\frac{1}{2}$	$\sqrt{3}$
$\frac{\pi}{2}$	1	0	∞

. . . in mathematics you don't understand things, you just get used to them.
– J. von Neumann



Law of cosines:

$$c^2 = a^2 + b^2 - 2ab\cos C.$$

Area:

$$A = \tfrac{1}{2}hc,$$
$$= \tfrac{1}{2}ab\sin C,$$
$$= \frac{c^2 \sin A \sin B}{2\sin C}.$$

Heron's formula:

$$A = \sqrt{s \cdot s_a \cdot s_b \cdot s_c},$$
$$s = \tfrac{1}{2}(a+b+c),$$
$$s_a = s - a,$$
$$s_b = s - b,$$
$$s_c = s - c.$$

More identities:

$$\sin \tfrac{x}{2} = \sqrt{\frac{1 - \cos x}{2}},$$
$$\cos \tfrac{x}{2} = \sqrt{\frac{1 + \cos x}{2}},$$
$$\tan \tfrac{x}{2} = \sqrt{\frac{1 - \cos x}{1 + \cos x}},$$
$$= \frac{1 - \cos x}{\sin x},$$
$$= \frac{\sin x}{1 + \cos x},$$
$$\cot \tfrac{x}{2} = \sqrt{\frac{1 + \cos x}{1 - \cos x}},$$
$$= \frac{1 + \cos x}{\sin x},$$
$$= \frac{\sin x}{1 - \cos x},$$
$$\sin x = \frac{e^{ix} - e^{-ix}}{2i},$$
$$\cos x = \frac{e^{ix} + e^{-ix}}{2},$$
$$\tan x = -i\frac{e^{ix} - e^{-ix}}{e^{ix} + e^{-ix}},$$
$$= -i\frac{e^{2ix} - 1}{e^{2ix} + 1},$$
$$\sin x = \frac{\sinh ix}{i},$$
$$\cos x = \cosh ix,$$
$$\tan x = \frac{\tanh ix}{i}.$$

The Chinese remainder theorem: There exists a number C such that:

$$C \equiv r_1 \bmod m_1$$
$$\vdots \quad \vdots \quad \vdots$$
$$C \equiv r_n \bmod m_n$$

if m_i and m_j are relatively prime for $i \neq j$.

Euler's function: $\phi(x)$ is the number of positive integers less than x relatively prime to x. If $\prod_{i=1}^n p_i^{e_i}$ is the prime factorization of x then

$$\phi(x) = \prod_{i=1}^n p_i^{e_i - 1}(p_i - 1).$$

Euler's theorem: If a and b are relatively prime then

$$1 \equiv a^{\phi(b)} \bmod b.$$

Fermat's theorem:

$$1 \equiv a^{p-1} \bmod p.$$

The Euclidean algorithm: if $a > b$ are integers then

$$\gcd(a, b) = \gcd(a \bmod b, b).$$

If $\prod_{i=1}^n p_i^{e_i}$ is the prime factorization of x then

$$S(x) = \sum_{d|x} d = \prod_{i=1}^n \frac{p_i^{e_i+1} - 1}{p_i - 1}.$$

Perfect Numbers: x is an even perfect number iff $x = 2^{n-1}(2^n - 1)$ and $2^n - 1$ is prime.

Wilson's theorem: n is a prime iff

$$(n-1)! \equiv -1 \bmod n.$$

Möbius inversion:

$$\mu(i) = \begin{cases} 1 & \text{if } i = 1. \\ 0 & \text{if } i \text{ is not square-free.} \\ (-1)^r & \text{if } i \text{ is the product of} \\ & r \text{ distinct primes.} \end{cases}$$

If

$$G(a) = \sum_{d|a} F(d),$$

then

$$F(a) = \sum_{d|a} \mu(d) G\left(\frac{a}{d}\right).$$

Prime numbers:

$$p_n = n \ln n + n \ln\ln n - n + n \frac{\ln\ln n}{\ln n} + O\left(\frac{n}{\ln n}\right),$$

$$\pi(n) = \frac{n}{\ln n} + \frac{n}{(\ln n)^2} + \frac{2!n}{(\ln n)^3} + O\left(\frac{n}{(\ln n)^4}\right).$$

Definitions:

Loop	An edge connecting a vertex to itself.
Directed	Each edge has a direction.
Simple	Graph with no loops or multi-edges.
Walk	A sequence $v_0 e_1 v_1 \ldots e_\ell v_\ell$.
Trail	A walk with distinct edges.
Path	A trail with distinct vertices.
Connected	A graph where there exists a path between any two vertices.
Component	A maximal connected subgraph.
Tree	A connected acyclic graph.
Free tree	A tree with no root.
DAG	Directed acyclic graph.
Eulerian	Graph with a trail visiting each edge exactly once.
Hamiltonian	Graph with a cycle visiting each vertex exactly once.
Cut	A set of edges whose removal increases the number of components.
Cut-set	A minimal cut.
Cut edge	A size 1 cut.
k-Connected	A graph connected with the removal of any $k-1$ vertices.
k-Tough	$\forall S \subseteq V, S \neq \emptyset$ we have $k \cdot c(G - S) \leq \lvert S \rvert$.
k-Regular	A graph where all vertices have degree k.
k-Factor	A k-regular spanning subgraph.
Matching	A set of edges, no two of which are adjacent.
Clique	A set of vertices, all of which are adjacent.
Ind. set	A set of vertices, none of which are adjacent.
Vertex cover	A set of vertices which cover all edges.
Planar graph	A graph which can be embeded in the plane.
Plane graph	An embedding of a planar graph.

$$\sum_{v \in V} \deg(v) = 2m.$$

If G is planar then $n - m + f = 2$, so

$$f \leq 2n - 4, \quad m \leq 3n - 6.$$

Any planar graph has a vertex with degree ≤ 5.

Notation:

$E(G)$	Edge set
$V(G)$	Vertex set
$c(G)$	Number of components
$G[S]$	Induced subgraph
$\deg(v)$	Degree of v
$\Delta(G)$	Maximum degree
$\delta(G)$	Minimum degree
$\chi(G)$	Chromatic number
$\chi_E(G)$	Edge chromatic number
G^c	Complement graph
K_n	Complete graph
K_{n_1,n_2}	Complete bipartite graph
$\mathrm{r}(k, \ell)$	Ramsey number

Geometry

Projective coordinates: triples (x, y, z), not all x, y and z zero.

$$(x, y, z) = (cx, cy, cz) \quad \forall c \neq 0.$$

Cartesian	Projective
(x, y)	$(x, y, 1)$
$y = mx + b$	$(m, -1, b)$
$x = c$	$(1, 0, -c)$

Distance formula, L_p and L_∞ metric:

$$\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2},$$

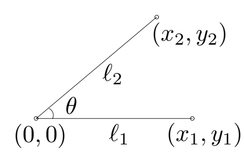
$$\left[|x_1 - x_0|^p + |y_1 - y_0|^p\right]^{1/p},$$

$$\lim_{p \to \infty} \left[|x_1 - x_0|^p + |y_1 - y_0|^p\right]^{1/p}.$$

Area of triangle (x_0, y_0), (x_1, y_1) and (x_2, y_2):

$$\tfrac{1}{2} \operatorname{abs} \begin{vmatrix} x_1 - x_0 & y_1 - y_0 \\ x_2 - x_0 & y_2 - y_0 \end{vmatrix}.$$

Angle formed by three points:



$$\cos\theta = \frac{(x_1, y_1) \cdot (x_2, y_2)}{\ell_1 \ell_2}.$$

Line through two points (x_0, y_0) and (x_1, y_1):

$$\begin{vmatrix} x & y & 1 \\ x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \end{vmatrix} = 0.$$

Area of circle, volume of sphere:

$$A = \pi r^2, \qquad V = \tfrac{4}{3}\pi r^3.$$

If I have seen farther than others, it is because I have stood on the shoulders of giants.
– Issac Newton

Wallis' identity:

$$\pi = 2 \cdot \frac{2 \cdot 2 \cdot 4 \cdot 4 \cdot 6 \cdot 6 \cdots}{1 \cdot 3 \cdot 3 \cdot 5 \cdot 5 \cdot 7 \cdots}$$

Brouncker's continued fraction expansion:

$$\tfrac{\pi}{4} = 1 + \cfrac{1^2}{2 + \cfrac{3^2}{2 + \cfrac{5^2}{2 + \cfrac{7^2}{2 + \cdots}}}}$$

Gregrory's series:

$$\tfrac{\pi}{4} = 1 - \tfrac{1}{3} + \tfrac{1}{5} - \tfrac{1}{7} + \tfrac{1}{9} - \cdots$$

Newton's series:

$$\tfrac{\pi}{6} = \frac{1}{2} + \frac{1}{2 \cdot 3 \cdot 2^3} + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5 \cdot 2^5} + \cdots$$

Sharp's series:

$$\tfrac{\pi}{6} = \frac{1}{\sqrt{3}}\left(1 - \frac{1}{3^1 \cdot 3} + \frac{1}{3^2 \cdot 5} - \frac{1}{3^3 \cdot 7} + \cdots\right)$$

Euler's series:

$$\tfrac{\pi^2}{6} = \tfrac{1}{1^2} + \tfrac{1}{2^2} + \tfrac{1}{3^2} + \tfrac{1}{4^2} + \tfrac{1}{5^2} + \cdots$$

$$\tfrac{\pi^2}{8} = \tfrac{1}{1^2} + \tfrac{1}{3^2} + \tfrac{1}{5^2} + \tfrac{1}{7^2} + \tfrac{1}{9^2} + \cdots$$

$$\tfrac{\pi^2}{12} = \tfrac{1}{1^2} - \tfrac{1}{2^2} + \tfrac{1}{3^2} - \tfrac{1}{4^2} + \tfrac{1}{5^2} - \cdots$$

Partial Fractions

Let $N(x)$ and $D(x)$ be polynomial functions of x. We can break down $N(x)/D(x)$ using partial fraction expansion. First, if the degree of N is greater than or equal to the degree of D, divide N by D, obtaining

$$\frac{N(x)}{D(x)} = Q(x) + \frac{N'(x)}{D(x)},$$

where the degree of N' is less than that of D. Second, factor $D(x)$. Use the following rules: For a non-repeated factor:

$$\frac{N(x)}{(x-a)D(x)} = \frac{A}{x-a} + \frac{N'(x)}{D(x)},$$

where

$$A = \left[\frac{N(x)}{D(x)}\right]_{x=a}.$$

For a repeated factor:

$$\frac{N(x)}{(x-a)^m D(x)} = \sum_{k=0}^{m-1} \frac{A_k}{(x-a)^{m-k}} + \frac{N'(x)}{D(x)},$$

where

$$A_k = \frac{1}{k!}\left[\frac{d^k}{dx^k}\left(\frac{N(x)}{D(x)}\right)\right]_{x=a}.$$

The reasonable man adapts himself to the world; the unreasonable persists in trying to adapt the world to himself. Therefore all progress depends on the unreasonable.
– George Bernard Shaw

Derivatives:

1. $\dfrac{d(cu)}{dx} = c\dfrac{du}{dx}$, **2.** $\dfrac{d(u+v)}{dx} = \dfrac{du}{dx} + \dfrac{dv}{dx}$, **3.** $\dfrac{d(uv)}{dx} = u\dfrac{dv}{dx} + v\dfrac{du}{dx}$,

4. $\dfrac{d(u^n)}{dx} = nu^{n-1}\dfrac{du}{dx}$, **5.** $\dfrac{d(u/v)}{dx} = \dfrac{v\left(\frac{du}{dx}\right) - u\left(\frac{dv}{dx}\right)}{v^2}$, **6.** $\dfrac{d(e^{cu})}{dx} = ce^{cu}\dfrac{du}{dx}$,

7. $\dfrac{d(c^u)}{dx} = (\ln c)c^u\dfrac{du}{dx}$, **8.** $\dfrac{d(\ln u)}{dx} = \dfrac{1}{u}\dfrac{du}{dx}$,

9. $\dfrac{d(\sin u)}{dx} = \cos u\dfrac{du}{dx}$, **10.** $\dfrac{d(\cos u)}{dx} = -\sin u\dfrac{du}{dx}$,

11. $\dfrac{d(\tan u)}{dx} = \sec^2 u\dfrac{du}{dx}$, **12.** $\dfrac{d(\cot u)}{dx} = \csc^2 u\dfrac{du}{dx}$,

13. $\dfrac{d(\sec u)}{dx} = \tan u \sec u\dfrac{du}{dx}$, **14.** $\dfrac{d(\csc u)}{dx} = -\cot u \csc u\dfrac{du}{dx}$,

15. $\dfrac{d(\arcsin u)}{dx} = \dfrac{1}{\sqrt{1-u^2}}\dfrac{du}{dx}$, **16.** $\dfrac{d(\arccos u)}{dx} = \dfrac{-1}{\sqrt{1-u^2}}\dfrac{du}{dx}$,

17. $\dfrac{d(\arctan u)}{dx} = \dfrac{1}{1+u^2}\dfrac{du}{dx}$, **18.** $\dfrac{d(\operatorname{arccot} u)}{dx} = \dfrac{-1}{1+u^2}\dfrac{du}{dx}$,

19. $\dfrac{d(\operatorname{arcsec} u)}{dx} = \dfrac{1}{u\sqrt{1-u^2}}\dfrac{du}{dx}$, **20.** $\dfrac{d(\operatorname{arccsc} u)}{dx} = \dfrac{-1}{u\sqrt{1-u^2}}\dfrac{du}{dx}$,

21. $\dfrac{d(\sinh u)}{dx} = \cosh u\dfrac{du}{dx}$, **22.** $\dfrac{d(\cosh u)}{dx} = \sinh u\dfrac{du}{dx}$,

23. $\dfrac{d(\tanh u)}{dx} = \operatorname{sech}^2 u\dfrac{du}{dx}$, **24.** $\dfrac{d(\coth u)}{dx} = -\operatorname{csch}^2 u\dfrac{du}{dx}$,

25. $\dfrac{d(\operatorname{sech} u)}{dx} = -\operatorname{sech} u \tanh u\dfrac{du}{dx}$, **26.** $\dfrac{d(\operatorname{csch} u)}{dx} = -\operatorname{csch} u \coth u\dfrac{du}{dx}$,

27. $\dfrac{d(\operatorname{arcsinh} u)}{dx} = \dfrac{1}{\sqrt{1+u^2}}\dfrac{du}{dx}$, **28.** $\dfrac{d(\operatorname{arccosh} u)}{dx} = \dfrac{1}{\sqrt{u^2-1}}\dfrac{du}{dx}$,

29. $\dfrac{d(\operatorname{arctanh} u)}{dx} = \dfrac{1}{1-u^2}\dfrac{du}{dx}$, **30.** $\dfrac{d(\operatorname{arccoth} u)}{dx} = \dfrac{1}{u^2-1}\dfrac{du}{dx}$,

31. $\dfrac{d(\operatorname{arcsech} u)}{dx} = \dfrac{-1}{u\sqrt{1-u^2}}\dfrac{du}{dx}$, **32.** $\dfrac{d(\operatorname{arccsch} u)}{dx} = \dfrac{-1}{|u|\sqrt{1+u^2}}\dfrac{du}{dx}$.

Integrals:

1. $\displaystyle\int cu\,dx = c\int u\,dx$, **2.** $\displaystyle\int (u+v)\,dx = \int u\,dx + \int v\,dx$,

3. $\displaystyle\int x^n\,dx = \frac{1}{n+1}x^{n+1}, \quad n \neq -1$, **4.** $\displaystyle\int \frac{1}{x}dx = \ln x$, **5.** $\displaystyle\int e^x\,dx = e^x$,

6. $\displaystyle\int \frac{dx}{1+x^2} = \arctan x$, **7.** $\displaystyle\int u\frac{dv}{dx}dx = uv - \int v\frac{du}{dx}dx$,

8. $\displaystyle\int \sin x\,dx = -\cos x$, **9.** $\displaystyle\int \cos x\,dx = \sin x$,

10. $\displaystyle\int \tan x\,dx = -\ln|\cos x|$, **11.** $\displaystyle\int \cot x\,dx = \ln|\cos x|$,

12. $\displaystyle\int \sec x\,dx = \ln|\sec x + \tan x|$, **13.** $\displaystyle\int \csc x\,dx = \ln|\csc x + \cot x|$,

14. $\displaystyle\int \arcsin\tfrac{x}{a}dx = \arcsin\tfrac{x}{a} + \sqrt{a^2-x^2}, \quad a > 0$,

15. $\int \arccos \frac{x}{a} dx = \arccos \frac{x}{a} - \sqrt{a^2 - x^2}, \quad a > 0,$

16. $\int \arctan \frac{x}{a} dx = x \arctan \frac{x}{a} - \frac{a}{2} \ln(a^2 + x^2), \quad a > 0,$

17. $\int \sin^2(ax) dx = \frac{1}{2a}\big(ax - \sin(ax)\cos(ax)\big),$

18. $\int \cos^2(ax) dx = \frac{1}{2a}\big(ax + \sin(ax)\cos(ax)\big),$

19. $\int \sec^2 x\, dx = \tan x,$

20. $\int \csc^2 x\, dx = -\cot x,$

21. $\int \sin^n x\, dx = -\frac{\sin^{n-1} x \cos x}{n} + \frac{n-1}{n} \int \sin^{n-2} x\, dx,$

22. $\int \cos^n x\, dx = \frac{\cos^{n-1} x \sin x}{n} + \frac{n-1}{n} \int \cos^{n-2} x\, dx,$

23. $\int \tan^n x\, dx = \frac{\tan^{n-1} x}{n-1} - \int \tan^{n-2} x\, dx, \quad n \neq 1,$

24. $\int \cot^n x\, dx = -\frac{\cot^{n-1} x}{n-1} - \int \cot^{n-2} x\, dx, \quad n \neq 1,$

25. $\int \sec^n x\, dx = \frac{\tan x \sec^{n-1} x}{n-1} + \frac{n-2}{n-1} \int \sec^{n-2} x\, dx, \quad n \neq 1,$

26. $\int \csc^n x\, dx = -\frac{\cot x \csc^{n-1} x}{n-1} + \frac{n-2}{n-1} \int \csc^{n-2} x\, dx, \quad n \neq 1,$

27. $\int \sinh x\, dx = \cosh x,$

28. $\int \cosh x\, dx = \sinh x,$

29. $\int \tanh x\, dx = \ln|\cosh x|,$

30. $\int \coth x\, dx = \ln|\sinh x|,$

31. $\int \operatorname{sech} x\, dx = \arctan \sinh x,$

32. $\int \operatorname{csch} x\, dx = \ln\left|\tanh \frac{x}{2}\right|,$

33. $\int \sinh^2 x\, dx = \frac{1}{4}\sinh(2x) - \frac{1}{2}x,$

34. $\int \cosh^2 x\, dx = \frac{1}{4}\sinh(2x) + \frac{1}{2}x,$

35. $\int \operatorname{sech}^2 x\, dx = \tanh x,$

36. $\int \operatorname{arcsinh} \frac{x}{a} dx = x \operatorname{arcsinh} \frac{x}{a} - \sqrt{x^2 + a^2}, \quad a > 0,$

37. $\int \operatorname{arctanh} \frac{x}{a} dx = x \operatorname{arctanh} \frac{x}{a} + \frac{a}{2} \ln|a^2 - x^2|,$

38. $\int \operatorname{arccosh} \frac{x}{a} dx = \begin{cases} x \operatorname{arccosh} \dfrac{x}{a} - \sqrt{x^2 + a^2}, & \text{if } \operatorname{arccosh} \frac{x}{a} > 0 \text{ and } a > 0, \\ x \operatorname{arccosh} \dfrac{x}{a} + \sqrt{x^2 + a^2}, & \text{if } \operatorname{arccosh} \frac{x}{a} < 0 \text{ and } a > 0, \end{cases}$

39. $\int \frac{dx}{\sqrt{a^2 + x^2}} = \ln\left(x + \sqrt{a^2 + x^2}\right), \quad a > 0,$

40. $\int \frac{dx}{a^2 + x^2} = \frac{1}{a} \arctan \frac{x}{a}, \quad a > 0,$

41. $\int \sqrt{a^2 - x^2}\, dx = \frac{x}{2}\sqrt{a^2 - x^2} + \frac{a^2}{2} \arcsin \frac{x}{a}, \quad a > 0,$

42. $\int (a^2 - x^2)^{3/2} dx = \frac{x}{8}(5a^2 - 2x^2)\sqrt{a^2 - x^2} + \frac{3a^4}{8} \arcsin \frac{x}{a}, \quad a > 0,$

43. $\int \frac{dx}{\sqrt{a^2 - x^2}} = \arcsin \frac{x}{a}, \quad a > 0,$

44. $\int \frac{dx}{a^2 - x^2} = \frac{1}{2a} \ln\left|\frac{a + x}{a - x}\right|,$

45. $\int \frac{dx}{(a^2 - x^2)^{3/2}} = \frac{x}{a^2\sqrt{a^2 - x^2}},$

46. $\int \sqrt{a^2 \pm x^2}\, dx = \frac{x}{2}\sqrt{a^2 \pm x^2} \pm \frac{a^2}{2} \ln\left|x + \sqrt{a^2 \pm x^2}\right|,$

47. $\int \frac{dx}{\sqrt{x^2 - a^2}} = \ln\left|x + \sqrt{x^2 - a^2}\right|, \quad a > 0,$

48. $\int \frac{dx}{ax^2 + bx} = \frac{1}{a} \ln\left|\frac{x}{a + bx}\right|,$

49. $\int x\sqrt{a + bx}\, dx = \frac{2(3bx - 2a)(a + bx)^{3/2}}{15b^2},$

50. $\int \frac{\sqrt{a + bx}}{x}\, dx = 2\sqrt{a + bx} + a \int \frac{1}{x\sqrt{a + bx}} dx,$

51. $\int \frac{x}{\sqrt{a + bx}}\, dx = \frac{1}{\sqrt{2}} \ln\left|\frac{\sqrt{a + bx} - \sqrt{a}}{\sqrt{a + bx} + \sqrt{a}}\right|, \quad a > 0,$

52. $\int \frac{\sqrt{a^2 - x^2}}{x}\, dx = \sqrt{a^2 - x^2} - a \ln\left|\frac{a + \sqrt{a^2 - x^2}}{x}\right|,$

53. $\int x\sqrt{a^2 - x^2}\, dx = -\frac{1}{3}(a^2 - x^2)^{3/2},$

54. $\int x^2\sqrt{a^2 - x^2}\, dx = \frac{x}{8}(2x^2 - a^2)\sqrt{a^2 - x^2} + \frac{a^4}{8} \arcsin \frac{x}{a}, \quad a > 0,$

55. $\int \frac{dx}{\sqrt{a^2 - x^2}} = -\frac{1}{a} \ln\left|\frac{a + \sqrt{a^2 - x^2}}{x}\right|,$

56. $\int \frac{x\, dx}{\sqrt{a^2 - x^2}} = -\sqrt{a^2 - x^2},$

57. $\int \frac{x^2\, dx}{\sqrt{a^2 - x^2}} = -\frac{x}{2}\sqrt{a^2 - x^2} + \frac{a^2}{2} \arcsin \frac{x}{a,} \quad a > 0,$

58. $\int \frac{\sqrt{a^2 + x^2}}{x}\, dx = \sqrt{a^2 + x^2} - a \ln\left|\frac{a + \sqrt{a^2 + x^2}}{x}\right|,$

59. $\int \frac{\sqrt{x^2 - a^2}}{x}\, dx = \sqrt{x^2 - a^2} - a \arccos \frac{a}{|x|}, \quad a > 0,$

60. $\int x\sqrt{x^2 \pm a^2}\, dx = \frac{1}{3}(x^2 \pm a^2)^{3/2},$

61. $\int \frac{dx}{x\sqrt{x^2 + a^2}} = \frac{1}{a} \ln\left|\frac{x}{a + \sqrt{a^2 + x^2}}\right|,$

62. $\displaystyle\int \frac{dx}{x\sqrt{x^2-a^2}} = \tfrac{1}{a}\arccos\tfrac{a}{|x|}, \quad a>0,$

63. $\displaystyle\int \frac{dx}{x^2\sqrt{x^2\pm a^2}} = \mp\frac{\sqrt{x^2\pm a^2}}{a^2x},$

64. $\displaystyle\int \frac{x\,dx}{\sqrt{x^2\pm a^2}} = \sqrt{x^2\pm a^2},$

65. $\displaystyle\int \frac{\sqrt{x^2\pm a^2}}{x^4}\,dx = \mp\frac{(x^2+a^2)^{3/2}}{3a^2x^3},$

66. $$\int \frac{dx}{ax^2+bx+c} = \begin{cases} \dfrac{1}{\sqrt{b^2-4ac}}\ln\left|\dfrac{2ax+b-\sqrt{b^2-4ac}}{2ax+b+\sqrt{b^2-4ac}}\right|, & \text{if } b^2>4ac, \\[2ex] \dfrac{2}{\sqrt{4ac-b^2}}\arctan\dfrac{2ax+b}{\sqrt{4ac-b^2}}, & \text{if } b^2<4ac, \end{cases}$$

67. $$\int \frac{dx}{\sqrt{ax^2+bx+c}} = \begin{cases} \dfrac{1}{\sqrt{a}}\ln\left|2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}\right|, & \text{if } a>0, \\[2ex] \dfrac{1}{\sqrt{-a}}\arcsin\dfrac{-2ax-b}{\sqrt{b^2-4ac}}, & \text{if } a<0, \end{cases}$$

68. $$\int \sqrt{ax^2+bx+c}\,dx = \frac{2ax+b}{4a}\sqrt{ax^2+bx+c} + \frac{4ax-b^2}{8a}\int\frac{dx}{\sqrt{ax^2+bx+c}},$$

69. $$\int \frac{x\,dx}{\sqrt{ax^2+bx+c}} = \frac{\sqrt{ax^2+bx+c}}{a} - \frac{b}{2a}\int\frac{dx}{\sqrt{ax^2+bx+c}},$$

70. $$\int \frac{dx}{x\sqrt{ax^2+bx+c}} = \begin{cases} \dfrac{-1}{\sqrt{c}}\ln\left|\dfrac{2\sqrt{c}\sqrt{ax^2+bx+c}+bx+2c}{x}\right|, & \text{if } c>0, \\[2ex] \dfrac{1}{\sqrt{-c}}\arcsin\dfrac{bx+2c}{|x|\sqrt{b^2-4ac}}, & \text{if } c<0, \end{cases}$$

71. $\displaystyle\int x^3\sqrt{x^2+a^2}\,dx = (\tfrac{1}{3}x^2-\tfrac{2}{15}a^2)(x^2+a^2)^{3/2},$

72. $\displaystyle\int x^n\sin(ax)\,dx = -\tfrac{1}{a}x^n\cos(ax)+\tfrac{n}{a}\int x^{n-1}\cos(ax)\,dx,$

73. $\displaystyle\int x^n\cos(ax)\,dx = \tfrac{1}{a}x^n\sin(ax)-\tfrac{n}{a}\int x^{n-1}\sin(ax)\,dx,$

74. $\displaystyle\int x^ne^{ax}\,dx = \frac{x^ne^{ax}}{a}-\tfrac{n}{a}\int x^{n-1}e^{ax}\,dx,$

75. $\displaystyle\int x^n\ln(ax)\,dx = x^{n+1}\left(\frac{\ln(ax)}{n+1}-\frac{1}{(n+1)^2}\right),$

76. $\displaystyle\int x^n(\ln ax)^m\,dx = \frac{x^{n+1}}{n+1}(\ln ax)^m-\frac{m}{n+1}\int x^n(\ln ax)^{m-1}\,dx.$

$$\begin{array}{rclcl}
x^1 = & x^{\underline{1}} & = & x^{\overline{1}} \\
x^2 = & x^{\underline{2}}+x^{\underline{1}} & = & x^{\overline{2}}-x^{\overline{1}} \\
x^3 = & x^{\underline{3}}+3x^{\underline{2}}+x^{\underline{1}} & = & x^{\overline{3}}-3x^{\overline{2}}+x^{\overline{1}} \\
x^4 = & x^{\underline{4}}+6x^{\underline{3}}+7x^{\underline{2}}+x^{\underline{1}} & = & x^{\overline{4}}-6x^{\overline{3}}+7x^{\overline{2}}-x^{\overline{1}} \\
x^5 = & x^{\underline{5}}+15x^{\underline{4}}+25x^{\underline{3}}+10x^{\underline{2}}+x^{\underline{1}} & = & x^{\overline{5}}-15x^{\overline{4}}+25x^{\overline{3}}-10x^{\overline{2}}+x^{\overline{1}}
\end{array}$$

$$\begin{array}{rlrl}
x^{\overline{1}} = & x^1 & x^{\underline{1}} = & x^1 \\
x^{\overline{2}} = & x^2+x^1 & x^{\underline{2}} = & x^2-x^1 \\
x^{\overline{3}} = & x^3+3x^2+2x^1 & x^{\underline{3}} = & x^3-3x^2+2x^1 \\
x^{\overline{4}} = & x^4+6x^3+11x^2+6x^1 & x^{\underline{4}} = & x^4-6x^3+11x^2-6x^1 \\
x^{\overline{5}} = & x^5+10x^4+35x^3+50x^2+24x^1 & x^{\underline{5}} = & x^5-10x^4+35x^3-50x^2+24x^1
\end{array}$$

Difference, shift operators:

$$\Delta f(x) = f(x+1)-f(x),$$
$$\mathrm{E}\,f(x) = f(x+1).$$

Fundamental Theorem:

$$f(x) = \Delta F(x) \Leftrightarrow \sum f(x)\delta x = F(x)+C.$$
$$\sum_a^b f(x)\delta x = \sum_{i=a}^{b-1} f(i).$$

Differences:

$\Delta(cu) = c\Delta u, \qquad \Delta(u+v) = \Delta u+\Delta v,$

$\Delta(uv) = u\Delta v+\mathrm{E}\,v\Delta u,$

$\Delta(x^{\underline{n}}) = nx^{\underline{n-1}},$

$\Delta(H_x) = x^{\underline{-1}}, \qquad \Delta(2^x) = 2^x,$

$\Delta(c^x) = (c-1)c^x, \qquad \Delta\binom{x}{m} = \binom{x}{m-1}.$

Sums:

$\sum cu\,\delta x = c\sum u\,\delta x,$

$\sum(u+v)\,\delta x = \sum u\,\delta x+\sum v\,\delta x,$

$\sum u\Delta v\,\delta x = uv-\sum \mathrm{E}\,v\Delta u\,\delta x,$

$\sum x^{\underline{n}}\,\delta x = \frac{x^{\underline{n+1}}}{m+1}, \qquad \sum x^{\underline{-1}}\,\delta x = H_x,$

$\sum c^x\,\delta x = \frac{c^x}{c-1}, \qquad \sum\binom{x}{m}\,\delta x = \binom{x}{m+1}.$

Falling Factorial Powers:

$x^{\underline{n}} = x(x-1)\cdots(x-n+1), \quad n>0,$

$x^{\underline{0}} = 1,$

$x^{\underline{n}} = \dfrac{1}{(x+1)\cdots(x+|n|)}, \quad n<0,$

$x^{\underline{n+m}} = x^{\underline{m}}(x-m)^{\underline{n}}.$

Rising Factorial Powers:

$x^{\overline{n}} = x(x+1)\cdots(x+n-1), \quad n>0,$

$x^{\overline{0}} = 1,$

$x^{\overline{n}} = \dfrac{1}{(x-1)\cdots(x-|n|)}, \quad n<0,$

$x^{\overline{n+m}} = x^{\overline{m}}(x+m)^{\overline{n}}.$

Conversion:

$x^{\underline{n}} = (-1)^n(-x)^{\overline{n}} = (x-n+1)^{\overline{n}}$
$\quad = 1/(x+1)^{\overline{-n}},$

$x^{\overline{n}} = (-1)^n(-x)^{\underline{n}} = (x+n-1)^{\underline{n}}$
$\quad = 1/(x-1)^{\underline{-n}},$

$$x^n = \sum_{k=1}^n \left\{ n \atop k \right\} x^{\underline{k}} = \sum_{k=1}^n \left\{ n \atop k \right\}(-1)^{n-k}x^{\overline{k}},$$

$$x^{\underline{n}} = \sum_{k=1}^n \left[n \atop k \right](-1)^{n-k}x^k,$$

$$x^{\overline{n}} = \sum_{k=1}^n \left[n \atop k \right] x^k.$$

Taylor's series:

$$f(x) = f(a) + (x-a)f'(a) + \frac{(x-a)^2}{2}f''(a) + \cdots = \sum_{i=0}^{\infty} \frac{(x-a)^i}{i!} f^{(i)}(a).$$

Expansions:

$$\frac{1}{1-x} = 1 + x + x^2 + x^3 + x^4 + \cdots = \sum_{i=0}^{\infty} x^i,$$

$$\frac{1}{1-cx} = 1 + cx + c^2x^2 + c^3x^3 + \cdots = \sum_{i=0}^{\infty} c^i x^i,$$

$$\frac{1}{1-x^n} = 1 + x^n + x^{2n} + x^{3n} + \cdots = \sum_{i=0}^{\infty} x^{ni},$$

$$\frac{x}{(1-x)^2} = x + 2x^2 + 3x^3 + 4x^4 + \cdots = \sum_{i=0}^{\infty} ix^i,$$

$$x^k \frac{d^n}{dx^n}\left(\frac{1}{1-x}\right) = x + 2^n x^2 + 3^n x^3 + 4^n x^4 + \cdots = \sum_{i=0}^{\infty} i^n x^i,$$

$$e^x = 1 + x + \tfrac{1}{2}x^2 + \tfrac{1}{6}x^3 + \cdots = \sum_{i=0}^{\infty} \frac{x^i}{i!},$$

$$\ln(1+x) = x - \tfrac{1}{2}x^2 + \tfrac{1}{3}x^3 - \tfrac{1}{4}x^4 - \cdots = \sum_{i=1}^{\infty} (-1)^{i+1}\frac{x^i}{i},$$

$$\ln\frac{1}{1-x} = x + \tfrac{1}{2}x^2 + \tfrac{1}{3}x^3 + \tfrac{1}{4}x^4 + \cdots = \sum_{i=1}^{\infty} \frac{x^i}{i},$$

$$\sin x = x - \tfrac{1}{3!}x^3 + \tfrac{1}{5!}x^5 - \tfrac{1}{7!}x^7 + \cdots = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)!},$$

$$\cos x = 1 - \tfrac{1}{2!}x^2 + \tfrac{1}{4!}x^4 - \tfrac{1}{6!}x^6 + \cdots = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i}}{(2i)!},$$

$$\tan^{-1} x = x - \tfrac{1}{3}x^3 + \tfrac{1}{5}x^5 - \tfrac{1}{7}x^7 + \cdots = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)},$$

$$(1+x)^n = 1 + nx + \tfrac{n(n-1)}{2}x^2 + \cdots = \sum_{i=0}^{\infty} \binom{n}{i} x^i,$$

$$\frac{1}{(1-x)^{n+1}} = 1 + (n+1)x + \tbinom{n+2}{2}x^2 + \cdots = \sum_{i=0}^{\infty} \binom{i+n}{i} x^i,$$

$$\frac{x}{e^x - 1} = 1 - \tfrac{1}{2}x + \tfrac{1}{12}x^2 - \tfrac{1}{720}x^4 + \cdots = \sum_{i=0}^{\infty} \frac{B_i x^i}{i!},$$

$$\frac{1}{2x}(1 - \sqrt{1-4x}) = 1 + x + 2x^2 + 5x^3 + \cdots = \sum_{i=0}^{\infty} \frac{1}{i+1}\binom{2i}{i} x^i,$$

$$\frac{1}{\sqrt{1-4x}} = 1 + x + 2x^2 + 6x^3 + \cdots = \sum_{i=0}^{\infty} \binom{2i}{i} x^i,$$

$$\frac{1}{\sqrt{1-4x}}\left(\frac{1-\sqrt{1-4x}}{2x}\right)^n = 1 + (2+n)x + \tbinom{4+n}{2}x^2 + \cdots = \sum_{i=0}^{\infty} \binom{2i+n}{i} x^i,$$

$$\frac{1}{1-x}\ln\frac{1}{1-x} = x + \tfrac{3}{2}x^2 + \tfrac{11}{6}x^3 + \tfrac{25}{12}x^4 + \cdots = \sum_{i=1}^{\infty} H_i x^i,$$

$$\frac{1}{2}\left(\ln\frac{1}{1-x}\right)^2 = \tfrac{1}{2}x^2 + \tfrac{3}{4}x^3 + \tfrac{11}{24}x^4 + \cdots = \sum_{i=2}^{\infty} \frac{H_{i-1}x^i}{i},$$

$$\frac{x}{1-x-x^2} = x + x^2 + 2x^3 + 3x^4 + \cdots = \sum_{i=0}^{\infty} F_i x^i,$$

$$\frac{F_n x}{1 - (F_{n-1} + F_{n+1})x - (-1)^n x^2} = F_n x + F_{2n}x^2 + F_{3n}x^3 + \cdots = \sum_{i=0}^{\infty} F_{ni} x^i.$$

Ordinary power series:

$$A(x) = \sum_{i=0}^{\infty} a_i x^i.$$

Exponential power series:

$$A(x) = \sum_{i=0}^{\infty} a_i \frac{x^i}{i!}.$$

Dirichlet power series:

$$A(x) = \sum_{i=1}^{\infty} \frac{a_i}{i^x}.$$

Binomial theorem:

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k.$$

Difference of like powers:

$$x^n - y^n = (x-y)\sum_{k=0}^{n-1} x^{n-1-k} y^k.$$

For ordinary power series:

$$\alpha A(x) + \beta B(x) = \sum_{i=0}^{\infty} (\alpha a_i + \beta b_i) x^i,$$

$$x^k A(x) = \sum_{i=k}^{\infty} a_{i-k} x^i,$$

$$\frac{A(x) - \sum_{i=0}^{k-1} a_i x^i}{x^k} = \sum_{i=0}^{\infty} a_{i+k} x^i,$$

$$A(cx) = \sum_{i=0}^{\infty} c^i a_i x^i,$$

$$A'(x) = \sum_{i=0}^{\infty} (i+1) a_{i+1} x^i,$$

$$xA'(x) = \sum_{i=1}^{\infty} i a_i x^i,$$

$$\int A(x)\,dx = \sum_{i=1}^{\infty} \frac{a_{i-1}}{i} x^i,$$

$$\frac{A(x) + A(-x)}{2} = \sum_{i=0}^{\infty} a_{2i} x^{2i},$$

$$\frac{A(x) - A(-x)}{2} = \sum_{i=0}^{\infty} a_{2i+1} x^{2i+1}.$$

Summation: If $b_i = \sum_{j=0}^{i} a_i$ then

$$B(x) = \frac{1}{1-x} A(x).$$

Convolution:

$$A(x)B(x) = \sum_{i=0}^{\infty} \left(\sum_{j=0}^{i} a_j b_{i-j} \right) x^i.$$

God made the natural numbers;
all the rest is the work of man.
– Leopold Kronecker

Expansions:

$$\frac{1}{(1-x)^{n+1}}\ln\frac{1}{1-x} = \sum_{i=0}^{\infty}(H_{n+i}-H_n)\binom{n+i}{i}x^i,$$

$$\left(\frac{1}{x}\right)^{\overline{-n}} = \sum_{i=0}^{\infty}\left\{ {i \atop n} \right\}x^i,$$

$$x^{\overline{n}} = \sum_{i=0}^{\infty}\left[{n \atop i} \right]x^i,$$

$$(e^x-1)^n = \sum_{i=0}^{\infty}\left\{ {i \atop n} \right\}\frac{n!x^i}{i!},$$

$$\left(\ln\frac{1}{1-x}\right)^n = \sum_{i=0}^{\infty}\left[{i \atop n} \right]\frac{n!x^i}{i!},$$

$$x\cot x = \sum_{i=0}^{\infty}\frac{(-4)^i B_{2i}x^{2i}}{(2i)!},$$

$$\tan x = \sum_{i=1}^{\infty}(-1)^{i-1}\frac{2^{2i}(2^{2i}-1)B_{2i}x^{2i-1}}{(2i)!},$$

$$\zeta(x) = \sum_{i=1}^{\infty}\frac{1}{i^x},$$

$$\frac{1}{\zeta(x)} = \sum_{i=1}^{\infty}\frac{\mu(i)}{i^x},$$

$$\frac{\zeta(x-1)}{\zeta(x)} = \sum_{i=1}^{\infty}\frac{\phi(i)}{i^x},$$

$$\zeta(x) = \prod_p \frac{1}{1-p^{-x}},$$

$$\zeta^2(x) = \sum_{i=1}^{\infty}\frac{d(i)}{x^i} \quad \text{where } d(n)=\textstyle\sum_{d|n}1,$$

$$\zeta(x)\zeta(x-1) = \sum_{i=1}^{\infty}\frac{S(i)}{x^i} \quad \text{where } S(n)=\textstyle\sum_{d|n}d,$$

$$\zeta(2n) = \frac{2^{2n-1}|B_{2n}|}{(2n)!}\pi^{2n}, \quad n\in\mathbb{N},$$

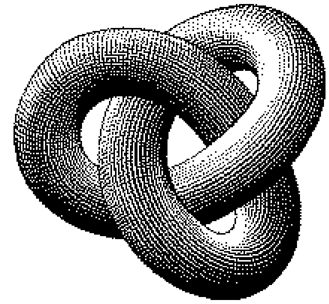
$$\frac{x}{\sin x} = \sum_{i=0}^{\infty}(-1)^{i-1}\frac{(4^i-2)B_{2i}x^{2i}}{(2i)!},$$

$$\left(\frac{1-\sqrt{1-4x}}{2x}\right)^n = \sum_{i=0}^{\infty}\frac{n(2i+n-1)!}{i!(n+i)!}x^i,$$

$$e^x\sin x = \sum_{i=1}^{\infty}\frac{2^{i/2}\sin\frac{i\pi}{4}}{i!}x^i,$$

$$\sqrt{\frac{1-\sqrt{1-x}}{x}} = \sum_{i=0}^{\infty}\frac{(4i)!}{16^i\sqrt{2}(2i)!(2i+1)!}x^i,$$

$$\left(\frac{\arcsin x}{x}\right)^2 = \sum_{i=0}^{\infty}\frac{4^i i!^2}{(i+1)(2i+1)!}x^{2i}.$$



Stieltjes Integration

If G is continuous in the interval $[a,b]$ and F is nondecreasing then

$$\int_a^b G(x)\,dF(x)$$

exists. If $a\le b\le c$ then

$$\int_a^c G(x)\,dF(x) = \int_a^b G(x)\,dF(x) + \int_b^c G(x)\,dF(x).$$

If the integrals involved exist

$$\int_a^b \big(G(x)+H(x)\big)\,dF(x) = \int_a^b G(x)\,dF(x) + \int_a^b H(x)\,dF(x),$$

$$\int_a^b G(x)\,d\big(F(x)+H(x)\big) = \int_a^b G(x)\,dF(x) + \int_a^b G(x)\,dH(x),$$

$$\int_a^b c\cdot G(x)\,dF(x) = \int_a^b G(x)\,d\big(c\cdot F(x)\big) = c\int_a^b G(x)\,dF(x),$$

$$\int_a^b G(x)\,dF(x) = G(b)F(b) - G(a)F(a) - \int_a^b F(x)\,dG(x).$$

If the integrals involved exist, and F possesses a derivative F' at every point in $[a,b]$ then

$$\int_a^b G(x)\,dF(x) = \int_a^b G(x)F'(x)\,dx.$$

Cramer's Rule

If we have equations:

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n &= b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n &= b_2 \\ \vdots \qquad\quad \vdots \qquad\qquad\quad &\vdots \\ a_{n,1}x_1 + a_{n,2}x_2 + \cdots + a_{n,n}x_n &= b_n \end{aligned}$$

Let $A=(a_{i,j})$ and B be the column matrix (b_i). Then there is a unique solution iff $\det A \ne 0$. Let A_i be A with column i replaced by B. Then

$$x_i = \frac{\det A_i}{\det A}.$$

00	47	18	76	29	93	85	34	61	52
86	11	57	28	70	39	94	45	02	63
95	80	22	67	38	71	49	56	13	04
59	96	81	33	07	48	72	60	24	15
73	69	90	82	44	17	58	01	35	26
68	74	09	91	83	55	27	12	46	30
37	08	75	19	92	84	66	23	50	41
14	25	36	40	51	62	03	77	88	99
21	32	43	54	65	06	10	89	97	78
42	53	64	05	16	20	31	98	79	87

The Fibonacci number system: Every integer n has a unique representation

$$n = F_{k_1} + F_{k_2} + \cdots + F_{k_m},$$

where $k_i \ge k_{i+1}+2$ for all i, $1\le i<m$ and $k_m\ge 2$.

Fibonacci Numbers

$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \ldots$

Definitions:

$$F_i = F_{i-1}+F_{i-2}, \quad F_0=F_1=1,$$

$$F_{-i} = (-1)^{i-1}F_i,$$

$$F_i = \frac{1}{\sqrt{5}}\left(\phi^i - \hat{\phi}^i\right),$$

Cassini's identity: for $i>0$:

$$F_{i+1}F_{i-1} - F_i^2 = (-1)^i.$$

Additive rule:

$$F_{n+k} = F_kF_{n+1} + F_{k-1}F_n,$$

$$F_{2n} = F_nF_{n+1} + F_{n-1}F_n.$$

Calculation by matrices:

$$\begin{pmatrix} F_{n-2} & F_{n-1} \\ F_{n-1} & F_n \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n.$$

Improvement makes strait roads, but the crooked roads without Improvement, are roads of Genius.
– William Blake (The Marriage of Heaven and Hell)