# GRAPH

**PERMUTATION**

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

int main () {
  int perm [5], n = 5;

  for (int i = 0; i < n; i++)
    perm [i] = i;

  for (int i = 0; i < (1 << n); i++) {
    //printf ("i: %d\n", i);
    for (int j = 0; j < n; j++) {
      //printf ("i: %d, (1 << %d): %d -- ", i, j, (1 << j));
      if (i & (1 << j))
        printf ("%d", perm [j]);

      //puts ("");
    }

    puts ("");
  }
  return 0;
}
```

**QUEEN**

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

const int MAXN = 8; // board size

// X == column | Y == row
// row [X] stores which row the queen is placed in the column X
int row [MAXN], lc, x, y;

bool check (int r, int c) {
  // for until column == check placed queens
  for (int i = 0; i < c; i++) {
    // if no two queens are in the same row
    // && no two queens share the same diagonal
    if (row [i] == r || (abs (c - i)) == (abs (r - row [i])))
      return false;
  }
  return true;
}

void backtrack (int column = 0) {
  if (column == MAXN && row [x] == y) {
    // all the queens are place == possible solution
    printf ("%2d      ", lc++);
    for (int i = 0; i < MAXN; i++) {
      if (i + 1 < MAXN)
        printf ("%d ", row [i] + 1);
      else
        printf ("%d\n", row [i] + 1);
    }
  }
  // which row from the column X the queen will be place
  for (int i = 0; i < MAXN; i++) {
    if (check (i, column)) {
      row [column] = i;
      // advance a column and calls the recursion
      backtrack (column + 1);
    }
  }
```

```
}

int main () {

  int n; scanf ("%d", &n);

  while (n--) {
    scanf ("%d%d", &x, &y);
    std::swap (x, y);
    x--; y--;

    memset (row, 0, sizeof (row));
    puts ("SOLN       COLUMN");
    puts (" #      1 2 3 4 5 6 7 8\n");

    lc = 1;
    backtrack ();

    if (n)
      puts ("");
  }


  return 0;
}
```

**BELLMAN-FORD**

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

void input ();
void solve ();

typedef std::pair <int, int> pi;
typedef std::vector <pi> vpi;
typedef std::vector <vpi> vvpi;

const int MAXN = 1e2 + 1;

int n, m;
vvpi g;
int dst [MAXN], INF = 0x3f3f3f3f;

void bl (int s) {
      dst [s] = 0;

      for (int i = 0; i < n - 1; i++)
            for (int j = 0; j < n; j++)
                  for (pi v : g [j])
                        dst [v.first] = std::min (dst [v.first], dst [j] +
v.second);

}

void input () {
      scanf ("%d%d", &n, &m);
      g.resize (n);
      memset (dst, 0x3f, sizeof dst);
      for (int i = 0; i < m; i++) {
            int a, b, c; scanf ("%d%d%d", &a, &b, &c);
            g [a].push_back ({b, c});
      }
      solve ();
}

void solve () {
```

```
	bl (0);
	for (int i = 0; i < n; i++)
		printf ("%d ", dst [i]);
	puts ("");
}

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

**BFS**

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

void input ();
void solve ();

typedef std::vector <int> vi;
typedef std::vector <vi> vvi;
typedef std::queue <int> qi;

const int MAXN = 1e3;

int n;
vvi g;
int visited [MAXN];

void bfs (int x) {
	printf ("x: %d", x);
	qi q;
	q.push (x);
	//memset (visited, -1, sizeof visited);
	visited [x] = 1;
	while (!q.empty ()) {
		int nd = q.front (); q.pop ();
		for (int v : g [nd]) {
			if (!visited [v]) {
				visited [v] = visited [nd] + 1;
				q.push (v);
			}
		}
	}
}

void input () {
	scanf ("%d", &n);
	g.resize (n + 1);
	for (int i = 0; i <= n; i++) {
		int a, b; scanf ("%d", &a);
		for (int j = 0; j < a; j++) {
			scanf ("%d", &b);
			g [i].push_back (b);
		}
	}
}

void solve () {
	for (int i = 0; i < g.size (); i++) {
		for (int j = 0; j < g [i].size (); j++)
			printf ("%d ", g [i][j]);
		puts ("");
	}
	bfs (5);
	for (int i = 0; i < g.size (); i++) {
		printf ("i: %d => ", i);
```

```
            for (int j = 0; j < g.size (); j++)
                  if (visited [j] == i)
                        printf ("%d ", j);
            puts ("");
      }
}

int main () {
      input ();
      solve ();
      return 0;
}
```

**BIPARTIDE CHECK**

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

void input ();
void solve ();

typedef std::vector <int> vi;
typedef std::vector <vi> vvi;

const int MAXN = 2e2 + 1;

int n, m;
vvi g;
int visited [MAXN];

bool bfs (int s) {
      int ans = 1;
      std::queue <int> q; q.push (s);
      visited [s] = 0;
      while (!q.empty () && ans) {
            int x = q.front (); q.pop ();
            for (int v : g [x]) {
                  if (visited [v] == -1) {
                        visited [v] = 1 - visited [x];
                        q.push (v);
                  }
                  else if (visited [v] == visited [x])
                        ans = 0;
            }
      }
      return ans;
}

bool dfs (int x, int f) {
      visited [x] = 1 - visited [f];
      int ans = 1;
      for (int v : g [x]) {
            if (visited [v] == -1)
                  dfs (v, x);
            else if (visited [v] == visited [x])
                  return false;
      }
      return true;
}

void input () {
      while (scanf ("%d", &n) && n != 0) {
            scanf ("%d", &m);
            g.clear ();
            g.resize (n);
            memset (visited, -1, sizeof visited);
            for (int i = 0; i < m; i++) {
                  int a, b; scanf ("%d%d", &a, &b);
```

```
                  g [a].push_back (b);
                  g [b].push_back (a);
            }
            solve ();
      }
}

void solve () {
      if (!dfs (0, 0))
            puts ("NOT BICOLORABLE.");
      else
            puts ("BICOLORABLE.");
}

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

**BIPARTIDE KUHN**

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

void input ();
void solve ();

typedef std::pair <int, int> pi;
typedef std::vector <pi> vpi;
typedef std::vector <int> vi;
typedef std::vector <vi> vvi;

int n, m;
int match [251], vis [251];
vvi g;
vpi ans;

bool dfs (int x) {
      if (vis [x])
            return 0;
      vis [x] = 1;
      for (auto v : g [x]) {
            if (match [v] == -1 || dfs (match [v])) {
                  match [v] = x;
                  return 1;
            }
      }

      return 0;
}

void input () {
      scanf ("%d%d", &n, &m);
      g.resize (n);
      memset (match, -1, sizeof match);
      for (int i = 0; i < n; i++) {
            while (1) {
                  int a; scanf ("%d", &a);
                  if (a)
                        g [i].push_back (a - 1);
                  else
                        break;
            }
      }
      solve ();
}
```

```
void solve () {
	for (int i = 0; i < n; i++) {
		memset (vis, 0, sizeof vis);
		dfs (i);
	}

	for (int i = 0; i < m; i++)
		if (match [i] != -1)
			ans.push_back ({match [i], i});

	printf ("%d\n", (int) ans.size ());
	for (int i = 0; i < ans.size (); i++)
		printf ("%d %d\n", ans [i].first + 1, ans [i].second + 1);

}

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

**BRIDGE ARTICULATION CHECK**

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

void input ();
void solve ();

typedef std::vector <int> vi;
typedef std::vector <vi> vvi;

const int MAXN = 1e2 + 1;

int n, m, rch, root;
vvi g;
vi stk;
int num [MAXN], low [MAXN], visited [MAXN];


// if num [x] <= low [v] it means that V cannot reach a vertex
// with num [w] <= num [x] so by removing vertex V the graph become
// disconect because V cannot reach X, so X is an articulation point.
//
// if num [x] < low [v] it means that edge {X, V} is a bridge becouse
// when that edge {X, V} is removed an ancestor os X is still reachable
// by V.
int dfs (int x, int f) {
	num [x] = ++rch;
	low [x] = num [x];
	for (int v : g [x]) {
		if (!num [v])
			low [x] = std::min (dfs (v, x), low [x]);
		if (v != f) {
			low [x] = std::min (low [v], low [x]);
			low [f] = std::min (low [x], low [f]);
		}
		// if can be chanced so that never prints more than one time
		// the same articulation vertex
		if (num [x] <= low [v]) {
			if (x != root)
				printf ("%d is an articulation vertex.\n", x);
			if (num [x] != low [v])
				printf ("{%d, %d} is a bridge.\n", x, v);
		}
	}
	return num [x];
}
```

```
int dfsSCC (int x, int f) {
	num [x] = ++rch;
	low [x] = num [x];
	visited [x] = 1;
	stk.push_back (x);

	for (int v : g [x]) {
		if (!visited [v])
			dfs (v, x);
		if (visited [v])
			low [x] = std::min (low [v], low [x]);
	}

	if (low [x] == num [x]) {
		nmb++;
		while (true) {
			int t = stk.back ();
			stk.pop_back ();
			visited [t] = 0;
			if (x == t)
				break;
		}
	}

	return num [x];
}

void cp3_dfs (int u, int f) {
	low [u] = num [u] = ++rch;

	for (int v : g [u]) {
		if (!num [v]) {
			dfs (v, u);
			if (num [u] <= low [v]) {
				if (u != root)
					printf ("%d is an articulation vertex.\n", u);
				if (num [u] != low [v])
					printf ("{%d, %d} is a bridge.\n", u, v);
			}
			low [u] = std::min (low [u], low [v]);
		}
		else if (v != f)
			low [u] = std::min (low [u], num [v]);
	}
}

void input () {
	scanf ("%d%d", &n, &m);
	g.clear ();
	g.resize (n);
	for (int i = 0; i < m; i++) {
		int a, b; scanf ("%d%d", &a, &b);
		g [a].push_back (b);
		g [b].push_back (a);
	}
	solve ();
}

void solve () {
	cp3_dfs (root, root);

	for (int i = 0; i < n; i++)
		printf ("%d = {%d, %d}\n", i, num [i] - 1, low [i] - 1);
}

int main () {
```

```
	input ();
	return 0;
}
```

**EDMOND KARP**

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

void input ();
void solve ();

typedef std::pair <int, int> pi;
typedef std::vector <pi> vpi;
typedef std::vector <vpi> vvpi;
typedef std::vector <int> vi;

const int MAXN = 1e3, INF = 0x3f3f3f3f;

int n, m, f, s, t;
vvpi g;
int res [MAXN][MAXN];
vi bst;

void augmented_path (int v, int min) {
	if (v == s) f = min;
	else if (bst [v] != -1) {
		// augmented_path (father of V, (min_edge (maximum flow) between the
current MIN and the edge {bst [v] -> v}));
		augmented_path (bst [v], std::min (min, res [bst [v]][v]));
		// F is a global variable which stores the min_edge (maximum_flow) on
the
		// bsf spanning tree
		res [bst [v]][v] -= f;
		res [v][bst [v]] += f;
	}
}

void ek () {
	int ans = 0;

	while (1) {
		f = 0;
		vi dst (n, INF); dst [s] = 0;
		std::queue <int> q;
		q.push (s);
		bst.assign (n, -1);
		// run bfs
		// bst [X] stores the father of X
		// on the bfs spanning tree
		while (!q.empty ()) {
			int u = q.front (); q.pop ();
			if (u == t) break;

			for (int v = 0; v < n; v++) {
				// if there is unused capacity and
				// v was not visited yet
				if (res [u][v] > 0 && dst [v] == INF) {
					dst [v] = dst [u] + 1;
					q.push (v);
					bst [v] = u;
				}
			}
		}
		// find a augmented path if exists
		augmented_path (t, INF);
		if (f == 0) break;
		ans += f;
```

```
	}
	printf ("ans: %d\n", ans);
}

void input () {
	scanf ("%d%d", &n, &m);
	scanf ("%d%d", &s, &t);
	g.clear ();
	g.resize (n);
	for (int i = 0; i < m; i++) {
		int a, b, c; scanf ("%d%d%d", &a, &b, &c);
		g [a].push_back ({b, c});
		res [a][b] = c;
	}
	solve ();
}

void solve () {
	ek ();
}

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

**DIJKSTRA**

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

void input ();
void solve ();

typedef std::pair <int, int> pi;
typedef std::vector <int> vi;
typedef std::vector <pi> vpi;
typedef std::vector <vpi> vvpi;

const int MAXN = 1e3, INF = 0x3f3f3f3f;

int n, m;
vi dst;
vvpi g;
std::priority_queue <pi> pq;

void bfs (int x) {
	pq.push ({0, x});
	dst [x] = 0;
	while (!pq.empty ()) {
		pi top = pq.top (); pq.pop ();
    printf ("{%d, %d}\n", top.first, top.second);
		if (top.first > dst [top.second]) continue;

		for (pi i : g [top.second]) {
			if (dst [top.second] + i.first < dst [i.second]) {
				dst [i.second] = dst [top.second] + i.first;
				pq.push ({dst [i.second], i.second});
			}
		}
	}
}

void input () {
	scanf ("%d%d", &n, &m);
	g.resize (n);
	dst.assign (n, INF);
```

```
	for (int i = 0; i < m; i++) {
		int a, b, c; scanf ("%d%d%d", &a, &b, &c);
		g[a].push_back ({c, b});
		g[b].push_back ({c, a});
	}

	solve ();
}

void solve () {
  puts ("calling dijkstra");
	bfs (0);

	for (int i = 0; i < dst.size (); i++)
		printf ("0 -> %d = %d\n", i, dst [i]);
}

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

FLOYD WARSHALL

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

void input ();
void solve ();

const int MAXN = 1e3, INF = 0x3f3f3f3f;

typedef std::pair <int, int> pi;
typedef std::vector <pi> vpi;
typedef std::vector <vpi> vvpi;

int n, m;
vvpi g;
int arr [MAXN][MAXN], p [MAXN][MAXN];

void fw () {
	for (int k = 0; k < n; k++)
		for (int i = 0; i < n; i++)
			for (int j = 0; j < n; j++) {
				printf ("{%d, %d} => {%d, %d} {%d, %d}\n", i, j, i, k, k,
j);
				if (arr [i][j] > arr [i][k] + arr [k][j] ) {
					printf ("{%d, %d} => {%d, %d} = %d\n", i, j, k, j, p
[k][j]);
					arr [i][j] = std::min (arr [i][j], arr [i][k] + arr
[k][j]);
					p [i][j] = p [k][j];
				}
			}
}

void path (int i, int j) {
	if (i != j)
		path (i, p [i][j]);
	printf ("%d ", j);
}

void input () {
	scanf ("%d%d", &n, &m);
	g.resize (n);
	memset (arr, 0x3f, sizeof arr);
	for (int i = 0; i < n; i++) {
		for (int j = 0; j < n; j++)
```

```
                  p [i][j] = i;
            arr [i][i] = 0;
      }
      for (int i = 0; i < m; i++) {
            int a, b, c; scanf ("%d%d%d", &a, &b, &c);
            g [a].push_back ({b, c});
            arr [a][b] = c;
      }
      solve ();
}

void solve () {
      fw ();
      for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++)
                  printf ("%d ", arr [i][j]);
            puts ("");
      }
      puts ("shortest path");
      path (3, 4);
      puts ("");
}

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

**EDGE PROPERTIE CHECK**

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

void input ();
void solve ();

typedef std::vector <int> vi;
typedef std::vector <vi> vvi;

const int MAXN = 2e2 + 1, explored = 2;

int n, m;
vvi g;
int visited [MAXN];

void dfs (int x, int f) {
      visited [x] = explored;
      for (int v : g [x]) {
            if (!visited [v]) {
                  printf ("Tree-edge: {%d, %d}\n", x, v);
                  dfs (v, x);
            }
            else if (visited [v] == explored && v != x)
                  printf ("Back-edge: {%d, %d}\n", x, v);
            else
                  printf ("Forward/cross: {%d, %d}\n", x, v);
      }
      visited [x] = 1;
}

void input () {
      scanf ("%d%d", &n, &m);
      g.clear ();
      g.resize (n);
      memset (visited, 0, sizeof visited);
      for (int i = 0; i < m; i++) {
            int a, b; scanf ("%d%d", &a, &b);
            g [a].push_back (b);
```

```
            g [b].push_back (a);
      }
      solve ();
}

void solve () {
      dfs (0, 0);
}

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

**MAXFLOW DINIC**

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

struct flow_edge {
      int v, u; // flow from 'v' to 'u'
      long long cap, flow = 0; // cap is the max flow in this edge
      flow_edge (int v, int u, long long cap) : v (v), u (u), cap (cap) {}
};

void input ();
void solve ();

typedef long long ll;
typedef std::pair <int, int> pi;
typedef std::vector <int> vi;
typedef std::vector <vi> vvi;
typedef std::vector <pi> vpi;
typedef std::vector <vpi> vvpi;
typedef std::vector <flow_edge> vf;
typedef std::queue <int> qi;

const long long flow_inf = 0x3f3f3f3f3f3f3f3f;
int n, pos;
int s, t;

vf edges;
vvi adj;
vi level, ptr;
qi q;

// init
void dinic (int n1, int s1, int t1) {
      n = n1; s = s1; t = t1;
      adj.clear ();
      level.clear ();
      ptr.clear ();
      adj.resize (n);
      level.resize (n);
      ptr.resize (n);
}

// add an edge to the residual graph
void add_edge (int v, int u, long long cap) {
      // add an edge V -> U with CAP
      // add an back-edge U -> V with 0
      edges.push_back (flow_edge (v, u, cap));
      edges.push_back (flow_edge (u, v, 0));
      // undirected graph which represents
      // the residual graph without any capacity
      adj [v].push_back (pos);
      adj [u].push_back (pos + 1);
      pos += 2;
```

```
}

bool bfs () {
	while (!q.empty ()) {
		int v = q.front ();
		q.pop ();
		for (int id : adj [v]) {
			// if there is no capacity left and
			// the vertex is already visited just continue
			if (edges [id].cap - edges [id].flow < 1)
				continue;
			if (level [edges [id].u] != -1)
				continue;
			// normal bfs
			level [edges [id].u] = level [v] + 1;
			q.push (edges [id].u);
		}
	}
	// return true if the sink T is reached
	return level [t] != -1;
}

long long dfs (int v, long long pushed) {
	if (pushed == 0)
		return 0;
	if (v == t)
		return pushed;

	// for all neightbors of v
	for (int &cid = ptr [v]; cid < (int)adj [v].size (); cid++) {
		// ID  is the id of edge which connects V -> U
		int id = adj [v][cid];
		// u is the back-edge of V -> U
		int u = edges [id].u;

		// if V -> U isnt on the bfs spanning tree
		// or it doesnt have capacity left just continue
		if (level [v] + 1 != level [u] || edges [id].cap - edges [id].flow < 1)
			continue;
		// find the min_edge capacity on the bfs spanning tree using dfs
		long long tr = dfs (u, std::min (pushed, edges [id].cap - edges
[id].flow));
		// if there isnt flow left, continue
		if (tr == 0)
			continue;
		// else update the flow residual graph
		edges [id].flow += tr;
		edges [id ^ 1].flow -= tr;
		// return TR
		return tr;
	}
	// return 0 if there isnt capacity left
	return 0;
}

long long flow () {
	long long f = 0;
	while (true) {
		// all level are -1
		std::fill (level.begin (), level.end (), -1);
		// source level is 0
		level [s] = 0;
		q.push (s);
		if (!bfs ())
			break;
		// all ptr are 0
		std::fill (ptr.begin (), ptr.end (), 0);
```

```
            // pushed will be the min-edge or max-flow in the
            // bfs spanning tree, which was found by the dfs
            while (long long pushed = dfs (s, flow_inf)) {
                  f += pushed;
            }
      }
      return f;
}

void input () {
}

void solve () {
}


int main () {
      // necessary to add the (S)ource and the (T)sink using add_edge ();
      return 0;
}
```

**MST KRUSKAL**

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

void input ();
void solve ();

typedef std::pair <int, int> pi;
typedef std::pair <int, pi> pipi;
typedef std::vector <int> vi;
typedef std::vector <pipi> vpi;

const int MAXN = 1e2 + 1;

vpi edges;
vi p, rank, size;
int n, m;
int visited [MAXN];
std::priority_queue <pi> pq;

int find (int i) {
      return (p [i] == i) ? i : p [i] = find (p [i]); // path compression
}

int same (int i, int j) {
      return find (i) == find (j);
}

int add (int i, int j) {
      if (!same (i, j)) {
            int x = find (i);
            int y = find (j);
            if (rank [x] > rank [y]) {
                  p [y] = x;
                  size [x] += size [y];
            }
            else {
                  p [x] = y;
                  size [y] += size [x];
                  if (rank [x] == rank [y])
                        rank [y]++;
            }
            return 1;
      }
      return 0;
}
```

```
void kruskal () {
	std::sort (edges.begin (), edges.end ());
	long long ans = 0;
	for (int i = 0; i < m; i++) {
		if (add (edges [i].second.first, edges [i].second.second)) {
			//printf ("{%d, %d}\n", edges [i].second.first, edges
[i].second.second);
			ans += edges [i].first;
		}
	}
	printf ("%lld\n", ans);
}

void aux (int s) {
	visited [s] = 1;
	for (int i = 0; i < m; i++)
		if (edges [i].second.first == s && !visited [edges [i].second.second])
			pq.push ({-edges [i].first, -edges [i].second.second});
		else if (edges [i].second.second == s && !visited [edges
[i].second.first])
			pq.push ({-edges [i].first, -edges [i].second.first});
}

void prim (int s) {
	long long ans = 0;

	aux (s);
	while (!pq.empty ()) {
		pi top = pq.top (); pq.pop ();
		printf ("{%d, %d}\n", -top.first, -top.second);
		if (!visited [-top.second]) {
			ans += -top.first;
			aux (-top.second);
		}
	}
	printf ("%lld\n", ans);
}

void input () {
	scanf ("%d%d", &n, &m);
	// n++; if starts from 1
	edges.clear ();
	size.clear ();
	rank.clear ();
	p.clear ();
	size.assign (n, 1);
	rank.assign (n, 0);
	p.assign (n, 0);
	for (int i = 0; i < n; i++)
		p [i] = i;
	for (int i = 0; i < m; i++) {
		int a, b, c; scanf ("%d%d%d", &a, &b, &c);
		edges.push_back ({c, {a, b}});
	}
	solve ();
}

void solve () {
	prim (0);
}

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

**TOPOLOGICAL SORT DAG**

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

void input ();
void solve ();

typedef std::vector <int> vi;
typedef std::vector <vi> vvi;

const int MAXN = 1e3;

int n, m;
int visited [MAXN];
int inc [MAXN], out [MAXN];
vi ans;
vvi g;

// the idea of topological sort is to always remove the TOP first, which could be
more than 1
// but everytime that a TOP is removed we get the next TOP
// TOP == vertices which has no incoming edges

void dfs (int x) {
      visited [x] = 1;
      for (int v : g [x])
            if (!visited [v])
                  dfs (v);
      ans.push_back (x);
}

void bfs () {
      std::queue <int> q;
      for (int i = 0; i < n; i++)
            if (!inc [i])
                  q.push (i);

      while (!q.empty ()) {
            int x = q.front (); q.pop ();
            ans.push_back (x);
            for (int v : g [x])
                  if (--inc [v] == 0)
                        q.push (v);
      }
}

void input () {
      scanf ("%d%d", &n, &m);
      g.clear ();
      g.resize (n);
      for (int i = 0; i < m; i++) {
            int a, b; scanf ("%d%d", &a, &b);
            g [a].push_back (b);
            out [a]++;
            inc [b]++;
      }
}

void solve () {
      /*
      for (int i = 0; i < n; i++)
            if (!visited [i])
                  dfs (i);
      for (int i = ans.size () - 1; i >= 0; i--)
            printf ("%d ", ans [i]);
```

```
      puts ("");
      */

      bfs ();
      for (int i = 0; i < ans.size (); i++)
            printf ("%d ", ans [i]);
      puts ("");
}

int main () {
      input ();
      solve ();
      return 0;
}
```

**SEGMENT TREE**

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

const int MAXN = 1e5 + 1;
int n;
int arr [MAXN * 2];

// ***** RANGE SUM QUERY *****

// use arr as the seg_tree and the input array
// all positions from N to 2 * N are already filled with the leaf values
void build () {
  for (int i = n - 1; i > 0; i--)
    arr [i] = arr [i << 1] + arr [(i << 1) + 1];
}

void update (int p, int value) {
  p += n; // p + n == leaf nodes indexs
  // arr [p] = value == updating the leaf index
  // arr [p >> 1] == father node
  // arr [p] + arr [p ^ 1] == pair that have the same father (odd and even number)
  for (arr [p] = value; p > 1; p >>= 1)
    arr [p >> 1] = arr [p] + arr [p ^ 1];
}

int rq (int l, int r) { // [L, R[
  int res = 0;
  l += n;
  r += n;
  for (; l < r; l >>= 1, r >>= 1) {
    if (l & 1)
      res += arr [l++];
    if (r & 1)
      res += arr [--r];
  }

  return res;
}

int main () {
  scanf ("%d", &n);

  for (int i = 0; i < n; i++)
    scanf ("%d", arr + i + n);

  build ();
  puts ("printing the tree:");
  for (int i = 1; i < 2 * n; i++)
    printf ("%d ", arr [i]);
  puts ("");
```

```
  printf ("rq [1, 2]: %d\n", rq (3, 8));
  return 0;
}
```

**FENWICK TREE**

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

#define LSOne(x) (x & (-x))

typedef std::vector <int> vi;
vi ft;
int n;
int q;

void insert (int num, int freq) {
  int x = num;
  // o num vai de num e se atualiaza com num += (LSOne (num))
  // ate que preencher todas as posicoes possiveis com
  // ft [num] += freq
  for (; num < ft.size (); num += (num & (-num))) {
    ft [num] += freq;
    printf ("num: %d, freq: %d, ft: %d\n", num, freq, ft [num]);
  }
}

int rsq(int num) { // returns RSQ(1, b)
  int sum = 0;
  // o num faz o processo inverso do insert assim,
  // como no insert ele vai crescendo, na busca ele
  // diminui
  for (; num; num -= LSOne(num)) {
    printf ("num: %d, ft[num]: %d\n", num, ft[num]);
    sum += ft[num];
  }
  return sum;
}

int rsq(int a, int b) { // returns RSQ(a, b)
  // se a == 1 retorna rsq (b) - 0
  // se a != 1 retorna rsq (b) - rsq (a - 1)
  // assim "excluindo" a parte do range antes de 'a'
  return rsq(b) - (a == 1 ? 0 : rsq(a - 1));
}

int main () {
  scanf ("%d", &n);
  ft.assign (n + 1, 0);

  for (int i = 1; i <= n; i++) {
    int a;
    puts ("NEW");
    scanf ("%d", &a);
    insert (i, a);
  }

  for (int i = 1; i < ft.size (); i++)
    printf ("%d ", ft [i]);
  puts ("");

  printf ("rsq : %d\n", rsq (3, 3));
  return 0;
}
```

# COMPUTATIONAL GEOMETRY

**BASIC**

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

struct Point {
  int x = 0, y = 0;

  Point operator- (Point a) {
    Point ans;
    ans.x = x - a.x;
    ans.y = y - a.y;
    return ans;
  }

  // lexographically
  bool operator< (Point a) {
    if (x == a.x)
      return y < a.y;
    return x < a.x;
  }
};

Point center;

// which quarter
int quarter (Point a) {
  if (a.x >= 0 && a.y >= 0)
    return 1;
  if (a.x <= 0 && a.y >= 0)
    return 2;
  if (a.x <= 0 && a.y <= 0)
    return 3;
  return 4;
}

// cros product
int cross (Point a, Point b) {
  return a.x * b.y - a.y * b.x;
}

// return a point which represents a vector
Point toVector (Point a, Point b) {
  Point ans;
  ans.x = b.x - a.x;
  ans.y = b.y - a.y;
  return ans;
}

// positive == left, 0 == colinear, negative == right
int checkTurn (Point a, Point b, Point c) {
  return cross (toVector (a, b), toVector (a, c));
}

// compare function by polar angle
bool cmp (Point a, Point b) {
  a = a - center; b = b - center; // relative points from a pivot
  if (quarter (a) == quarter (b))
    return cross (a, b) > 0;

  return quarter (a) < quarter (b);
}

// convex hull Andrew`s monotone chain
std::vector <Point> ch_monotone (std::vector <Point> pts) {
  int k = 0;
  std::vector <Point> ans (pts.size () * 2);
```

```
  std::sort (pts.begin (), pts.end ());

  puts ("SORTED:");
  for (int i = 0; i < pts.size (); i++)
    printf ("{%d, %d} ", pts [i].x, pts [i].y);
  puts ("");

  // lower hull
  for (int i = 0; i < pts.size (); i++) {

    // more than 2 points and while notConvex remove last point
    while (k >= 2 && checkTurn (ans [k - 2], ans [k - 1], pts [i]) < 0)
      k--;
    ans [k++] = pts [i];
  }
  printf ("k: %d\n", k);
  for (int i = 0; i < k; i++)
    printf ("{%d, %d} ", ans [i].x, ans [i].y);
  puts ("");
  // upper hull
  for (int i = (int)pts.size () - 2, t = k + 1; i >= 0; i--) {
    // k >= t to not erase the lower hull ans while notConvex remove last point
    while (k >= t && checkTurn (ans [k - 2], ans [k - 1], pts [i]) < 0)
      k--;
    ans [k++] = pts [i];
  }

  ans.resize (k);
  return ans;
}


int main () {
  std::vector <Point> input, hull;

  input.resize (7);
  input [0].x = 0; input [0].y = 0;
  input [1].x = 0; input [1].y = 8;
  input [2].x = 1; input [2].y = 6;
  input [3].x = 3; input [3].y = 1;
  input [4].x = 6; input [4].y = 6;
  input [5].x = 8; input [5].y = 0;
  input [6].x = 8; input [6].y = 8;

  puts ("INPUT:");
  for (int i = 0; i < input.size (); i++)
    printf ("{%d, %d} ", input [i].x, input [i].y);
  puts ("");

  hull = ch_monotone (input);

  puts ("CH:");
  for (int i = 0; i < hull.size (); i++)
    printf ("{%d, %d} ", hull [i].x, hull [i].y);
  puts ("");

  return 0;
}
```

**INTEGER LINE**

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

typedef long long ll;

struct point {ll x, y;};
struct line {
```

```
  ll a, b, c; //ax - by = c
  bool operator <(const line& rhs) const {
    return std::make_tuple(a, b, c) < std::make_tuple(rhs.a, rhs.b, rhs.c);
  }
};

ll gcd(ll a, ll b) {
  while (b) {
    a %= b;
    std::swap(a, b);
  }
  return a;
}

line pointsToLine(point p1, point p2) {
  ll a = p1.y - p2.y;
  ll b = p1.x - p2.x;
  ll c = p1.y * p2.x - p2.y * p1.x;

  ll d = gcd(gcd(a, b), c);
  a/=d, b/=d, c/=d;

  if (a < 0) a*=-1, b*=-1, c*=-1;

  return {a, b, c};
}

bool areParallel(line l1, line l2) {
  return (l1.a == l2.a) && (l1.b == l2.b);
}

bool areSame(line l1, line l2) {
  return areParallel(l1, l2) && (l1.c == l2.c);
}
```

**INTERSECT**

```
struct point {int x, y;};

struct segment {point a, b;};

point tovec(point a, point b) {
  return {b.x-a.x, b.y-a.y};
}

int cross(point a, point b) {
  return a.x*b.y - a.y*b.x;
}

bool ccw(point p, point q, point r) {
  return cross(tovec(p, q), tovec(p, r)) > 0;
}

bool collinear(point p, point q, point r) {
  return cross(tovec(p, q), tovec(p, r)) == 0;
}

bool in_bouding_box(point p, point q, point r) {
  int maxX = std::max(q.x, r.x), minX = std::min(q.x, r.x);
  int maxY = std::max(q.y, r.y), minY = std::min(q.y, r.y);
  return p.x <= maxX && p.x >= minX && p.y <= maxY && p.y >= minY;
}

bool segment_intersect(segment s, segment t) {
  if (collinear(s.a, t.a, t.b) && in_bouding_box(s.a, t.a, t.b)) return true;
  if (collinear(s.b, t.a, t.b) && in_bouding_box(s.b, t.a, t.b)) return true;
  if (collinear(t.a, s.a, s.b) && in_bouding_box(t.a, s.a, s.b)) return true;
```

```
  if (collinear(t.b, s.a, s.b) && in_bouding_box(t.b, s.a, s.b)) return true;

  if (ccw(s.a, s.b, t.a) != ccw(s.a, s.b, t.b)
      && ccw(t.a, t.b, s.a) != ccw(t.a, t.b, s.b))
  {
    return true;
  }

  return false;
}
```

**CALOPSITA**

```
#define EPS 1e-9

struct point {
      double x, y;
      point() { x = y = 0.0; }
      point(double _x, double _y) : x(_x), y(_y) {}
      double norm() { return hypot(x, y); }
      point normalized() {
            return point(x,y)*(1.0/norm());
      }
      double angle() { return atan2(y, x); }
      double polarAngle() {
            double a = atan2(y, x);
            return a < 0 ? a + 2*acos(-1.0) : a;
      }
      bool operator < (point other) const {
            if (fabs(x - other.x) > EPS) return x < other.x;
            else return y < other.y;
      }
      bool operator == (point other) const {
            return (fabs(x - other.x) < EPS && (fabs(y - other.y) < EPS));
      }
      point operator +(point other) const {
            return point(x + other.x, y + other.y);
      }
      point operator -(point other) const {
            return point(x - other.x, y - other.y);
      }
      point operator *(double k) const {
            return point(x*k, y*k);
      }
};
double dist(point p1, point p2) {
      return hypot(p1.x - p2.x, p1.y - p2.y);
}
double inner(point p1, point p2) {
      return p1.x*p2.x + p1.y*p2.y;
}
double cross(point p1, point p2) {
      return p1.x*p2.y - p1.y*p2.x;
}
bool ccw(point p, point q, point r) {
      return cross(q-p, r-p) > 0;
}
bool collinear(point p, point q, point r) {
      return fabs(cross(p-q, r-p)) < EPS;
}
point rotate(point p, double rad) {
      return point(p.x * cos(rad) - p.y * sin(rad),
                   p.x * sin(rad) + p.y * cos(rad));
}
double angle(point a, point o, point b) {
      return acos(inner(a-o, b-o) / (dist(o,a)*dist(o,b)));
}
```

```
point proj(point u, point v) {
	return v*(inner(u,v)/inner(v,v));
}
bool between(point p, point q, point r) {
	return collinear(p, q, r) && inner(p - q, r - q) <= 0;
}
point lineIntersectSeg(point p, point q, point A, point B) {
	double c = cross(A-B, p-q);
	double a = cross(A, B);
	double b = cross(p, q);
	return ((p-q)*(a/c)) - ((A-B)*(b/c));
}
bool parallel(point a, point b) {
	return fabs(cross(a, b)) < EPS;
}
bool segIntersects(point a, point b, point p, point q) {
	if (parallel(a-b, p-q)) {
		return between(a, p, b) || between(a, q, b)
			|| between(p, a, q) || between(p, b, q);
	}
	point i = lineIntersectSeg(a, b, p, q);
	return between(a, i, b) && between(p, i, q);
}
point closestToLineSegment(point p, point a, point b) {
	double u = inner(p-a, b-a) / inner(b-a, b-a);
	if (u < 0.0) return a;
	if (u > 1.0) return b;
	return a + ((b-a)*u);
}
struct circle{
	point c;
	double r;
	circle() { c = point(); r = 0; }
	circle(point _c, double _r) : c(_c), r(_r) {}
	double area() { return acos(-1.0)*r*r; }
	double chord(double rad) { return  2*r*sin(rad/2.0); }
	double sector(double rad) { return 0.5*rad*area()/acos(-1.0); }
	bool intersects(circle other) {
		return dist(c, other.c) < r + other.r;
	}
	bool contains(point p) { return dist(c, p) <= r + EPS; }
	pair<point, point> getTangentPoint(point p) {
		double d1 = dist(p, c), theta = asin(r/d1);
		point p1 = rotate(c-p,-theta);
		point p2 = rotate(c-p,theta);
		p1 = p1*(sqrt(d1*d1-r*r)/d1)+p;
		p2 = p2*(sqrt(d1*d1-r*r)/d1)+p;
		return make_pair(p1,p2);
	}
	vector< pair<point,point> > getTangentSegs(circle other) {
		vector<pair<point, point> > ans;
		double d = dist(other.c, c);
		double dr = abs(r - other.r), sr = r + other.r;
		if (dr >= d) return ans;
		double u = acos(dr / d);
		point dc1 = ((other.c - c).normalized())*r;
		point dc2 = ((other.c - c).normalized())*other.r;
		ans.push_back(make_pair(c + rotate(dc1, u), other.c + rotate(dc2, u)));
		ans.push_back(make_pair(c + rotate(dc1, -u), other.c + rotate(dc2, -
u)));
		if (sr >= d) return ans;
		double v = acos(sr / d);
		dc2 = ((c - other.c).normalized()) * other.r;
		ans.push_back({c + rotate(dc1, v), other.c + rotate(dc2, v)});
		ans.push_back({c + rotate(dc1, -v), other.c + rotate(dc2, -v)});
		return ans;
	}
```

```
      pair<point, point> getIntersectionPoints(circle other){
            assert(intersects(other));
            double d = dist(c, other.c);
            double u = acos((other.r*other.r + d*d - r*r) / (2*other.r*d));
            point dc = ((other.c - c).normalized()) * r;
            return make_pair(c + rotate(dc, u), c + rotate(dc, -u));
      }
};

circle circumcircle(point a, point b, point c) {
      circle ans;
      point u = point((b-a).y, -(b-a).x);
      point v = point((c-a).y, -(c-a).x);
      point n = (c-b)*0.5;
      double t = cross(u,n)/cross(v,u);
      ans.c = ((a+c)*0.5) + (v*t);
      ans.r = dist(ans.c, a);
      return ans;
}

int insideCircle(point p, circle c) {
      if (fabs(dist(p , c.c) - c.r)<EPS) return 1;
      else if (dist(p , c.c) < c.r) return 0;
      else return 2;
} //0 = inside/1 = border/2 = outside

circle incircle( point p1, point p2, point p3 ) {
      double m1=dist(p2, p3);
      double m2=dist(p1, p3);
      double m3=dist(p1, p2);
      point c = (p1*m1+p2*m2+p3*m3)*(1/(m1+m2+m3));
      double s = 0.5*(m1+m2+m3);
      double r = sqrt(s*(s-m1)*(s-m2)*(s-m3))/s;
      return circle(c, r);
}
```

**POLYGON AREA**

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

void input ();
void solve ();

struct Point {
  int x = 0, y = 0;

  Point operator- (Point a) {
    Point ans;
    ans.x = x - a.x;
    ans.y = y - a.y;
    return ans;
  }

  // lexographically
  bool operator< (Point a) {
    if (x == a.x)
      return y < a.y;
    return x < a.x;
  }
};

typedef std::pair <float, float> pi;
typedef std::vector <Point> vpi;

int n;
vpi arr;
```

```
Point center;

int quarter (Point a) {
  if (a.x >= 0 && a.y >= 0)
    return 1;
  if (a.x <= 0 && a.y >= 0)
    return 2;
  if (a.x <= 0 && a.y <= 0)
    return 3;
  return 4;
}

// cros product
int cross (Point a, Point b) {
  return a.x * b.y - a.y * b.x;
}

// compare function by polar angle
bool cmp (Point a, Point b) {
  a = a - center; b = b - center; // relative points from a pivot
  if (quarter (a) == quarter (b))
    return cross (a, b) > 0;

  return quarter (a) < quarter (b);
}

void input () {
      scanf ("%d", &n);
      for (int i = 0; i < n; i++) {
            int a, b; scanf ("%d%d", &a, &b);
            arr.push_back ({a, b});
      }
      solve ();
}

void solve () {
      //std::random_shuffle (arr.begin (), arr.end ());
      std::sort (arr.begin (), arr.end (), cmp);
      arr.push_back ({arr [0].x, arr [0].y});
      float ans = 0;
      for (int i = 0; i < n; i++) {
            ans += (float)arr [i].x * (float)arr [i + 1].y;
            ans -= (float)arr [i].y * (float)arr [i + 1].x;
      }
      printf ("%f\n", ans / 2.0);
}

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

**SEGMENT INTERSECTION**

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

void input ();
void solve ();

typedef long long ll;
typedef long double ld;

const double EPS = 1e-9;

typedef struct point {
      ld x, y;
```

```
	point () {}
	point (ld x, ld y) : x (x), y (y) {}

	point operator- (point& a) {
		return {x - a.x, y - a.y};
	}

	bool operator< (point& a) {
		if (x == a.x)
			return y < a.y;
		return x < a.x;
	}

	bool operator> (point& a) {
		if (x == a.x)
			return y > a.y;
		return x > a.x;
	}

	bool operator== (point &a) {
		return x == a.x && y == a.y;
	}

	bool operator>= (point &a) {
		if (x == a.x)
			return y > a.y || y == a.y;
		return x > a.x || x == a.x;
	}

	bool operator<= (point &a) {
		if (x == a.x)
			return y < a.y || y == a.y;
		return x < a.x || x == a.x;
	}

} point;

typedef struct segment {
	point a, b;

	segment () {}
	segment (point a, point b) : a (a), b (b) {}
} segment;

typedef struct line {
	ld a, b, c;

	line () {}
	line (ld a, ld b, ld c) : a (a), b (b), c (c) {}
} line;

line pointsToLine(point p1, point p2) {
	line l;
	if (fabs(p1.x-p2.x) < EPS)
		l = {1.0, 0.0, -p1.x};
	else {
		double a = -(double)(p1.y-p2.y) / (p1.x-p2.x);
		l = {a , 1.0, -(double)(a*p1.x) - p1.y};
	}
	return l;
}


point to_vec (const point& a, const point& b) {
	return {a.x - b.x, a.y - b.y};
}
```

```
ld dot (point a, point b) {
      return a.x*b.x + a.y*b.y;
}

ld cross (point a, point b) {
      return a.x*b.y - a.y*b.x;
}

int check_turn (point a, point b, point c) {
  return cross (to_vec (a, b), to_vec (a, c)) > 0;
}

bool collinear(point p, point q, point r) {
  return cross(to_vec(p, q), to_vec(p, r)) == 0;
}

bool in_bouding_box(point p, point q, point r) {
  ld maxX = std::max(q.x, r.x), minX = std::min(q.x, r.x);
  ld maxY = std::max(q.y, r.y), minY = std::min(q.y, r.y);
  return p.x <= maxX && p.x >= minX && p.y <= maxY && p.y >= minY;
}

bool segment_intersect(segment s, segment t) {
  if (collinear(s.a, t.a, t.b) && in_bouding_box(s.a, t.a, t.b)) return true;
  if (collinear(s.b, t.a, t.b) && in_bouding_box(s.b, t.a, t.b)) return true;
  if (collinear(t.a, s.a, s.b) && in_bouding_box(t.a, s.a, s.b)) return true;
  if (collinear(t.b, s.a, s.b) && in_bouding_box(t.b, s.a, s.b)) return true;

  if (check_turn(s.a, s.b, t.a) != check_turn(s.a, s.b, t.b)
      && check_turn(t.a, t.b, s.a) != check_turn(t.a, t.b, s.b))
    return true;

  return false;
}

bool areParallel(line l1, line l2) {       // check coefficients a & b
      return (fabs(l1.a-l2.a) < EPS) && (fabs(l1.b-l2.b) < EPS);
}

bool are_same(line l1, line l2) {           // also check coefficient c
      return areParallel(l1 ,l2) && (fabs(l1.c-l2.c) < EPS);
}

point line_intersect (line l1, line l2) {
      point p;
      p.x = (l2.b*l1.c - l1.b*l2.c) / (l2.a*l1.b - l1.a*l2.b);
      if (fabs(l1.b) > EPS)
            p.y = -(l1.a*p.x + l1.c);
      else
            p.y = -(l2.a*p.x + l2.c);

      return p;
}

point arr [4];

void input () {
      for (int i = 0; i < 4; i++)
            scanf ("%Lf%Lf", &(arr + i)->x, &(arr + i)->y);

      solve ();
}

void solve () {
      if (segment_intersect ({arr [0], arr [1]}, {arr [2], arr [3]})) {
            line l1 = pointsToLine (arr [0], arr [1]);
            line l2 = pointsToLine (arr [2], arr [3]);
```

```
            //printf ("{%Lf, %Lf, %Lf}\n", l1.a, l1.b, l1.c);
            //printf ("{%Lf, %Lf, %Lf}\n", l2.a, l2.b, l2.c);
            if (are_same (l1, l2)) {
                  std::sort (arr, arr + 2);
                  std::sort (arr + 2, arr + 4);

                  point a, b;

                  if (arr [0] <= arr [2]) { // 0---2
                        if (arr [1] <= arr [3]) { // 0---2--1--3
                              a = arr [2];
                              b = arr [1];
                        }
                        else { // 0---2--3--1
                              a = arr [2];
                              b = arr [3];
                        }
                  }
                  else { // 2---0
                        if (arr [1] >= arr [3]) { // 2---0--3--1
                              a = arr [0];
                              b = arr [3];
                        }
                        else { // 2---0--1--3
                              a = arr [0];
                              b = arr [1];
                        }
                  }

                  if (a == b)
                        printf ("%.10Lf %.10Lf\n", a.x, a.y);
                  else {
                        printf ("%.10Lf %.10Lf\n", a.x, a.y);
                        printf ("%.10Lf %.10Lf\n", b.x, b.y);
                  }
            }
            else {
                  point p = line_intersect (l1, l2);
                  printf ("%.10Lf %.10Lf\n", p.x, p.y);
            }
      }
      else
            puts ("Empty");
}

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

# STRINGS

**LCS**

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

void input ();
void solve ();

typedef std::pair <int, int> pi;
typedef std::vector <int> vi;

const int MAXN = 1e2 + 1;

std::string str1, str2;
int dp [MAXN][MAXN];
pi t [MAXN][MAXN];
```

```
vi t1, t2;

int lcs (int i, int j) {
	if (i >= str1.size () || j >= str2.size ())
		return 0;

	if (dp [i][j])
		return dp [i][j];


	if (str1 [i] == str2 [j]) {
		dp [i][j] = lcs (i + 1, j + 1) + 1;
		t [i][j] = std::make_pair (i + 1, j + 1);
	}
	else {
		int a = lcs (i + 1, j), b = lcs (i, j + 1);

		if (a > b) {
			dp [i][j] = a;
			t [i][j] = std::make_pair (i + 1, j);
		}
		else {
			dp [i][j] = b;
			t [i][j] = std::make_pair (i, j + 1);
		}
	}

	return dp [i][j];
}

void input () {
	std::cin >> str1 >> str2;
	solve ();
}

void solve () {
	printf ("%d\n", lcs (0, 0));

	int a = 0, b = 0;
	while (a < str1.size () && b < str2.size ()) {
		//printf ("{%d, %d} = {%c, %c}\n", a, b, str1 [a], str2 [b]);
		if (str1 [a] == str2 [b]) {
			t1.push_back (t [a][b].first);
			t2.push_back (t [a][b].second);
		}

		pi p = t [a][b];
		a = p.first;
		b = p.second;
	}

	for (int i = 0; i < t1.size (); i++)
		printf ("%d ", t1 [i]);
	puts ("");
	for (int i = 0; i < t2.size (); i++)
		printf ("%d ", t2 [i]);
	puts ("");
}

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

**KMP**

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

```
const int MAXP = 100; // max pattern size

int lsp [MAXP]; // suffix which is a prefix

void build_lps (std::string a) { // a == pattern
  int i = 1, j = 0;
  lps [0] = 0;

  while (i < a.size ()) {
    if (a[i] == a [j]) { // match
      j++;
      lps [i++] = j;
    }
    else { // missmatch
      // if j == 0 cant go back anymore
      // so lps [i] doesnt belongs to a suffix
      // which is a prefix
      // else => can go back so there's still chance
      if (j == 0)
        lps [i++] = 0;
      else
        j = lps [j - 1];
    }
  }
}

void kmp_search (std::string a, std::string b) { // a == text, b == pattern
  int i = 0; j = 0;
  build_lps (b);

  while (i < a.size () - b.size ()) { // until text.size - pattern.size
    if (a [i] == b [j]) { //match
      i++; j++;
      if (j == b.size ()) { // pattern match
        j = lps [j - 1];
      }
    }
    else {
      if (j == 0)
        i++;
      else
        j = lps [j - 1];
    }
  }
}

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

# MISCELANIA

**SIEVE ERASTHOTENES**

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

const int MAXN = 1e7 + 1;

std::vector <long long> isp (MAXN, true), p, spf (MAXN);

void sieve () {
  isp [0] = false;
  isp [1] = false;

  for (int i = 2; i <= MAXN; i++) {
    if (isp [i]) {
      p.push_back (i);
```

```
      spf [i] = i;
    }

    for (long long j = 0; (j < p.size ()) && (i * p [j] < MAXN) && (p [j] <= spf
[i]); j++) {
      isp [i * p [j]] = false;
      spf [i * p [j]] = p [j];
    }
  }
}

int main () {
  int n; scanf ("%d", &n);
  sieve ();

  for (int i = 0; i < n; i++)
    printf ("%d ", p [i]);
  puts ("");

  return 0;
}
```