GRAPH THEORY

----------------------------------------------------

**BFS**

|  |
| --- |
| void **bfs**(int u) {     queue <int> q;     q.push(u);     vis[u] = 1;     while (!q.empty()) {         u = q.front();         q.pop();         printf("%d ", u);         for (int i = 0; i < g[u].size(); i++) {             int v = g[u][i];             if (!vis[v]) q.push(v), vis[v] = 1;         }     } } |

----------------------------------------

**DFS**

|  |
| --- |
| void **dfs\_visit**(int u) {     vis[u] = 1;     printf("%d ", u);     for (int v = 0; v < g[u].size(); v++)         if (!vis[g[u][v]]) dfs\_visit(g[u][v]); } |

----------------------------------------

**Count All Paths**

|  |
| --- |
| void **countAllPaths**(int u, int d, list<int> \*adj, vector <bool> visited, int &totalPaths) {     visited[u] = true;     if (u == d) totalPaths++;     else {         for (auto i = adj[u].begin(); i != adj[u].end(); i++)             if (!visited[\*i]) countAllPaths(\*i, d, adj, visited, totalPaths);     }     visited[u] = false; }  int **countPaths**(int nVertex, list<int> \*adj, int s, int d) {     vector <bool> visited(nVertex, false);     int totalPaths = 0;     countAllPaths(s, d, adj, visited, totalPaths);     return totalPaths; } |

|  |
| --- |
|  |

----------------------------------------

**Count All Trees**

|  |
| --- |
| void **DFS\_again**(int u, vector <int> adj[], vector <bool> &visited) {     visited[u] = true;     for (int v = 0; v < adj[u].size(); v++)         if (!visited[adj[u][v]])             DFS\_again(adj[u][v], adj, visited); }  int **count\_TREE**(vector <int> adj[], int nVertex) {     vector <bool> visited(nVertex, false);     int cnt = 0;     for (int u = 0; u < nVertex; u++) { // DFS         if (!visited[u]) {             DFS\_again(u, adj, visited);             cnt++;         }     }     return cnt; } |

----------------------------------------

**Detect Cycle**

|  |
| --- |
| bool **isCycleUtil**(int v) {     if (!vis[v]) {         vis[v] = true, recStack[v] = true;         for (auto i = g[v].begin(); i != g[v].end(); i++) {             if (!vis[\*i] && isCycleUtil(\*i)) return true;             else if (recStack[\*i]) return true;         }     }     recStack[v] = false;     return false; }  bool **isCyclic**() {     vis.assign(nodes, false);     recStack.assign(nodes, false);     for (int i = 0; i < nodes; i++)         if (isCycleUtil(i)) return true;     return false; } |

-----------------------------------------

**Flood Fill**

|  |
| --- |
| int dx[] = {+0, +0, +1, -1, -1, +1, -1, +1}; int dy[] = {-1, +1, +0, +0, +1, +1, -1, -1};  void **floodfill**(int i, int j, int cnt) {     if (i < 0 || j < 0 || i >= row || j >= column) return; // base case      if (grid[i][j] == '.' && !vis[i][j]) {         vis[i][j] = true;         grid[i][j] = (cnt + '0');         for (int m = 0; m < 8; m++) {             int x = i + dx[m];             int y = j + dy[m];             floodfill(x, y, cnt);         }      } } int **cnt**(0); for (int i = 0; i < row; i++) {     for (int j = 0; j < column; j++) {         if (grid[i][j] == '.' && !vis[i][j]) {             Cnt++, floodfill(i, j, cnt);         }     } } |

----------------------------------------

**Dijkstra by Matrix**

|  |
| --- |
| inline int **minDistance**() {  int minValue = INT\_MX, minIndex = 0;  for (int i = 0; i < nodes; i++)  if (!vis[i] && dis[i] < minValue)  minValue = dis[i], minIndex = i;  return minIndex; }  void **dijkstra**(int src) {  for (int i = 0; i < nodes; i++)  dis[i] = INT\_MX, vis[i] = false;  dis[src] = 0;  for (int i = 0; i < nodes-1; i++) {  int u = minDistance();  vis[u] = true;  for (int v = 0; v < nodes; v++) {  if (!vis[v] && g[u][v] && dis[u] != INT\_MX && dis[u] + g[u][v] < dis[v])  dis[v] = dis[u] + g[u][v];  }  }  for (int i = 0; i < nodes; i++) printf("%d - %d\n", i, dis[i]); } |

-------------------------------------

**Dijkstra**

|  |
| --- |
| struct **node** {     int u, w;     node(int u, int w) { this->u = u, this->w = w; }     bool operator < (const node& p) const { return w < p.w; } };  void **dijkstra**(int src) {     for (int i = 1; i <= nodes; i++) dis[i] = inf;     priority\_queue <node> q;     q.push(node(src, 0));     dis[src] = 0;      while (!q.empty()) {         node top = q.top(); q.pop();         int u = top.u;         for (int i = 0; i < (int) g[u].size(); i++) {             int v = g[u][i].first;             int w = g[u][i].second;             if (dis[u] + w < dis[v]) {                 dis[v] = dis[u] + w;                 q.push(node(v, dis[v]));             }         }     } }  void **print\_dis**(int src) {     for (int i = 1; i <= nodes; i++) {         printf("%d - %d = %d\n", src, i, dis[i]);     } } |

-------------------------------------

**Kruskal MST**

|  |
| --- |
| struct **edge** {  int u, v, w;  bool operator < (const edge& p) const {  return w < p.w;  } };  vector <edge> g; int parent[MX]; int nodes, edges;  inline int **find**(int r) {  return (parent[r] == r) ? r : find(parent[r]); }  inline void **kruskal\_mst**() {  sort(g.begin(), g.end());  for (int i = 0; i < nodes; i++) parent[i] = i;  int cnt = 0, tcost = 0, u, v, w;  for (int i = 0; i < edges; i++) { // edges = g.size()  u = find(g[i].u);  v = find(g[i].v);  if (u != v) {  cout << g[i].u << " - " << g[i].v << " " << g[i].w << "\n";  parent[u] = v;  cnt++;  tcost += g[i].w;  if (cnt == nodes-1) break;  }  }  printf("\nminimum cost: %d\n", tcost); } |

-------------------------------------

**Prim's MST by Matrix**

|  |
| --- |
| inline void **printMST**() {     printf("Edge \t cost\n");     for (int i = 1; i < nodes; i++) {         printf("%d - %d \t %d\n", parent[i], i, g[i][parent[i]]);     } }  inline int **minNODE**() {     int min = INT\_MAX, minINDEX;     for (int i = 0; i < nodes; i++) {         if (!vis[i] && dis[i] < min) {             min = dis[i], minINDEX = i;         }     }     return minINDEX; }    PTO  inline void **prim\_MST**() {     for (int i = 0; i < nodes; i++) {         vis[i] = false, dis[i] = INT\_MAX;     }     dis[0] = 0;     parent[0] = -1;     for (int i = 0; i < nodes-1; i++) {         int u = minNODE();         vis[u] = true;         for (int v = 0; v < nodes; v++) {             if (g[u][v] && !vis[v] && g[u][v] < dis[v])                  parent[v] = u, dis[v] = g[u][v];         }     }     printMST(); } |

---------------------------------------

**Prim's MST**

|  |
| --- |
| inline void **Prim\_MST**() {     vector <int> dis(nodes, INT\_MAX);     vector <bool> vis(nodes, false);     vector <int> parent(nodes, -1);     int source = 0;     dis[0] = 0;       priority\_queue <dual, vector <dual>, greater <dual> > q;     q.push(make\_pair(0, source));       while (!q.empty()) {  int u = q.top().second;  q.pop();  vis[u] = true;  for (int x = 0; x < (int) g[u].size(); x++) {      int v = g[u][x].first;      int w = g[u][x].second;      if (!vis[v] && dis[v] > w) {  dis[v] = w;  q.push(make\_pair(dis[v], v));  parent[v] = u;      }  }     }     for (int i = 1; i < nodes; i++)  printf("%d - %d \t%d\n", parent[i], i, dis[i]); } |

-----------------------------------

**Strongly Connected Component**

|  |
| --- |
| vector <int> g1[mx], g2[mx]; bool vis[mx]; stack <int> s1;  void **dfs1**(int u) {     vis[u] = true;     for (int v = 0; v < g1[u].size(); v++) {         if (!vis[g1[u][v]]) {             dfs1(g1[u][v]);         }     }     s1.push(u); } void **dfs2**(int u) {     vis[u] = true;     cout << u << " ";     for (int v = 0; v < g2[u].size(); v++) {         if (!vis[g2[u][v]]) {             dfs2(g2[u][v]);         }     } }  // main class for (int i = 0; i < n; i++) {     if (!vis[i]) {        dfs1(i);     } } memset(vis, 0, sizeof(vis)); while (!s1.empty()) {    int u = s1.top();    s1.pop();    if (!vis[u]) {       dfs2(u);       cout << endl;    } } |

-----------------------------------

**Topological Sort**

|  |
| --- |
| stack <int> s;  void **printStack**() {     for (int i = 0; i < nodes; i++)         cout << s.top() << " ", s.pop(); }  void **topologicalSortUtil**(int v) {     vis[v] = true;     for (auto i = g[v].begin(); i != g[v].end(); i++)         if (!vis[\*i]) topologicalSortUtil(\*i);     s.push(v); }  void **topologicalSort**() {     vis.assign(nodes, false);     for (int i = 0; i < nodes; i++)         if (!vis[i]) topologicalSortUtil(i); } |

-----------------------------------

**Top Sort**

|  |
| --- |
| inline void **bfs**() {     queue <int> q;     for (int i = 1; i <= x; i++)         if (!vis[i]) q.push(i);      while (!q.empty()) {         int v = q.front(), q.pop();         printf("%d ", v);         for (auto i: g[v]) {             vis[i]--;             if (!vis[i]) q.push(i);         }     } } |

-----------------------------------

**BellmanFord**

|  |
| --- |
| bool **BellmanFord**(int src) {     for (int i = 1; i <= nodes;) dis[i++] = inf;     dis[src] = 0;      // relax all edges     for (int i = 1; i <= nodes; i++) {         for (int j = 1; j <= edges; j++) {             uu = g[j].u;             vv = g[j].v;             ww = g[j].w;             if (dis[uu] != inf && dis[uu] + ww < dis[vv]) {                 dis[vv] = dis[uu] + ww;             }         }     }      // check negative-weight cycles     for (int i = 1; i <= edges; i++) {         uu = g[i].u;         vv = g[i].v;         ww = g[i].w;         if (dis[uu] != inf && dis[uu] + ww < dis[vv]) return true;     }     return false; }  // Main class if (BellmanFord(src\_node)) printf("Negative weight cycle detected."); else {     for (int i = 1; i <= nodes; i++) {         printf("%d - %d = %d\n", src\_node, i, dis[i]);     } } |

---------------------------------------

**Floyd Warshall**

|  |
| --- |
| void **floyd\_warshall**() {     for (int k = 1; k <= nodes; k++) {         dis[k][k] = 0;         for (int i = 1; i <= nodes; i++) {             for (int j = 1; j <= nodes; j++) {                 if (dis[i][k] != inf && dis[k][j] != inf && dis[i][k] + dis[k][j] < dis[i][j]) {                     dis[i][j] = dis[i][k] + dis[k][j];                 }             }         }     } }  void **print\_distance\_matrix**() {     for (int i = 1; i <= nodes; i++) {         for (int j = 1; j <= nodes; j++) {             if (dis[i][j] == inf) printf("inf ");             else printf("%d ", dis[i][j]);         }         printf("\n");     } } |

------------------------------------

**K-th shortest path**

|  |
| --- |
| struct **node** {     int u, w;     node(int u, int w) {         this->u = u;         this->w = w;     };     bool operator < (const node& p) const {         return w > p.w;     }; };  int t, nodes, edges, vis[mx]; vector <pii> g[mx];  int **dijkstra**(int x, int y, int k) {     memset(vis, 0, sizeof(vis));     priority\_queue <node> q;     q.push(node(x, 0));      while (!q.empty()) {         node top = q.top();         q.pop();         int u = top.u;         int w = top.w;         if (vis[u] == k) continue;         vis[u]++;         if (vis[u] == k && u == y) return w;                  for (int i = 0; i < (int) g[u].size(); i++) {             int v = g[u][i].first;             int nw = g[u][i].second;             if (vis[v] < k) q.push(node(v, w+nw));         }     }     return -1; }  // Main class printf("%d\n", dijkstra(x, y, k)); |

------------------------------------

|  |
| --- |
| Created by: Amirul Islam      github.com/shiningflash |

DYNAMIC PROGRAMMING

----------------------------------------------------

**Fibonacci**

|  |
| --- |
| int lookup[le];  ll **fib**(int n) {     if (lookup[n] == nil) {         if (n <= 1) lookup[n] = n;         else lookup[n] = fib(n-1) + fib(n-2);     }     return lookup[n]; } |

----------------------------------------------------

**Binomial Coefficient**

|  |
| --- |
| ll **bioCOEFF**(int n, int k) {     ll res = 1;     if (k > n-k) k = n-k;     for (int i = 0; i < k; i++) {         res \*= (n-i);         res /= (i+1);     }     return res; } |

----------------------------------------------------

**Coin Change**

|  |
| --- |
| int **coin\_chnage**() {     dp[0] = 1;     for (int i = 0; i < m; i++)         for (int j = a[i]; j <= n; j++)             dp[j] += dp[j-a[i]];     return dp[n]; } |

----------------------------------------------------

**Knapsack 1**

|  |
| --- |
| ll val[N], w[N], dp[N]; ll **knapsack**(int n, int W) {     CLEAR(dp);     for (int i = 0; i < n; i++)         for (int j = W; j >= w[i]; j--)             dp[j] = max(val[i] + dp[j-w[i]], dp[j]);     return dp[W]; } |

----------------------------------------------------

**Knapsack 2**

|  |
| --- |
| int w[] = {10, 20, 30}; int val[] = {60, 100, 120}; int cnt = 0;  int **knapsack**(int W, int n) {     cnt++;     cout << W << " " << n << endl;     if (n == 0 || W == 0) return 0;     if (w[n-1] > W) return knapsack(W, n-1);     else return max(val[n-1] + knapsack(W-w[n-1], n-1), knapsack(W, n-1)); }  // Main method cout << knapsack(W, n) << " " << cnt << endl; |

----------------------------------------------------

**Knapsack 3**

|  |
| --- |
| int w[] = {10, 20, 30}; int val[] = {60, 100, 120};  int **knapsack**(int W, int n) {     int dp[n+1][W+1];     for (int i = 0; i <= n; i++) dp[i][0] = 0;     for (int i = 0; i <= W; i++) dp[0][i] = 0;     for (int i = 1; i <= n; i++) {         for (int j = 1; j <= W; j++) {             if (w[i-1] <= W)                 dp[i][j] = max(val[i-1] + dp[i-1][j-w[i-1]], dp[i-1][j]);             else                 dp[i][j] = dp[i-1][j];         }     }     return dp[n][W]; }  // Main method int n = 3, W = 50; cout << knapsack(W, n) << endl; |

----------------------------------------------------

**Gold Mine Problem**

|  |
| --- |
| int gold[10000][10000]; int dp[10000][10000];  int **getMaxGold**(int m, int n) {     for (int i = 0; i < m; dp[i][0] = gold[i][0], i++);     for (int i = 1; i < m; i++) {         for (int j = 0; j < n; j++) {             int right = dp[j][i-1];             int rightUP = (i == 0) ? 0 : dp[j-1][i-1];             int rightDN = (i == m-1) ? 0 : dp[j+1][i-1];             dp[j][i] = gold[j][i] + max(right, max(rightUP, rightDN));         }     }     int sol = dp[m-1][n-1];     for (int i = 0; i < m; sol = max(sol, dp[i][n-1]), i++);     return sol; } |

----------------------------------------------------

**LCS**

|  |
| --- |
| int cost[105][105], path[105][105], lcs[105]; int t, l1, l2, lcs\_len;  inline void **LCS\_LENGTH**() {     for (int i = 1; i <= l1; cost[i++][0] = 0;     for (int j = 1; j <= l2; cost[0][j++] = 0;    for (int i = 1; i <= l1; i++) {         for (int j = 1; j <= l2; j++) {             if (s1[i-1] == s2[j-1]) {                 cost[i][j] = cost[i-1][j-1] + 1;                 path[i][j] = 1;             }             else if (cost[i-1][j] >= cost[i][j-1]) {                 cost[i][j] = cost[i-1][j];                 path[i][j] = 2;             }             else {                 cost[i][j] = cost[i][j-1];                 path[i][j] = 3;             }         }     }     lcs\_len = cost[l1][l2]; }  inline void **LCS**() {     int i = l1, j = l2, k = lcs\_len-1;     while (k >= 0) {         if (path[i][j] == 1) {             lcs[k] = s1[i-1];             i--; j--; k--;         }         else if (path[i][j] == 2) i--;         else if (path[i][j] == 3) j--;     } }  inline void **LCS\_PRINT**() {     if (lcs\_len <= 0) cout << ":(";     else         for (int i = 0; i < lcs\_len; i++)             cout << (char) lcs[i];     cout << endl; } |

----------------------------------------------------

**Longest Palindromic Subsequence**

|  |
| --- |
| // DP Solution :: TC - O(N^2) :: SC - O(N) int **lps\_1D**(string s, int n) {     int dp[n];     for (int i = n-1; i >= 0; i--) {         int back\_up = 0;         for (int j = i; j < n; j++) {             if (i == j) dp[j] = 1;             else if (s[i] == s[j]) {                 dp[j] = back\_up + 2;                 back\_up = dp[j] - 2;             }             else {                 back\_up = dp[j];                 dp[j] = max(dp[j-1], dp[j]);             }         }     }     return dp[n-1]; } |

----------------------------------------------------

**Longest Palindromic Substring**

|  |
| --- |
| // TC - O(N^2) :: SC - O(1) void **longestPalSubStr\_Efficient**(string s) {     int n = s.length(), start = 0, maxLen = 1;     for (int i = 1; i < n; i++) {         // longest even length palindrome         // center point as i-1, i         int low = i-1, high = i;         while (low >= 0 && high < n && s[low] == s[high]) {             if (high - low + 1 > maxLen) start = low, maxLen = high - low + 1;             Low--, high++;         }         //longest odd palindrome         // center point as i         int low = i-1; high = i+1;         while (low >= 0 && high < n && s[low] == s[high]) {             if (high - low + 1 > maxLen) start = low, maxLen = high - low + 1;             Low--, high++;         }     }     printLPS(s, start, maxLen); } |

----------------------------------------------------

**Maximum Subarray Sum**

|  |
| --- |
| int **maxSubArrSum**(int a[], int sz) {     int cur\_max = a[0], global\_max = a[0];     for (int i = 1; i < sz; i++) {         cur\_max = max(a[i], cur\_max + a[i]);         global\_max = max(global\_max, cur\_max);     }     return global\_max; }  int **maxSubArrSum\_withStartEnd**(int a[], int sz) {     int cur\_max = 0, global\_max = INT\_MIN, start(0), end(0), s(0);     for (int i = 0; i < sz; i++) {         cur\_max = cur\_max + a[i];         if (global\_max < cur\_max) global\_max = cur\_max, start = s, end = i;         if (cur\_max < 0) cur\_max = 0, s = i + 1;     }     printf("\nMax Sub Array Sum = %d\nStart = %d\tEnd = %d\n", global\_max, start, end); } |

----------------------------------------------------

**Subset Sum**

|  |
| --- |
| // complexity: Pseudo-polynomial bool **isSubsetSumDP**(int a[], int n, int sum) {     bool subset[n+1][sum+1];     for (int i = 0; i <= n; i++) subset[i][0] = true;     for (int i = 0; i <= sum; i++) subset[0][i] = false;     for (int i = 1; i <= n; i++) {         for (int j = 1; j <= sum; j++) {             if (j < a[i-1]) subset[i][j] = subset[i-1][j];             else subset[i][j] = subset[i-1][j] || subset[i-1][j-a[i-1]];         }     }     return subset[n][sum]; } |

----------------------------------------------------

DATA STRUCTURE

----------------------------------------------------

**Disjoint Set Unit**

|  |
| --- |
| // return the representation of r int **Find**(int r) {  if (par[r] == r) return r;  return par[r] = Find(par[r]); }  // make Union void **Union**(int u, int v) {  par[u] = v; }  // at the beginning, everyone's representative is it itself void **init**(int n) {  for (int i = 1; i <= n; i++) par[i] = i; }  // main method scanf("%d %d", &a, &b); int u = Find(a); int v = Find(b); if (u == v) printf("already friend\n"); else Union(u, v); |

----------------------------------------------------

**SQRT Decomposition**

|  |
| --- |
| void **getsum**(int a[], int n, int b[], int m, int l, int r) {     int sum = 0, lb = l/m, rb = r/m;     if (lb == rb) for (int i = l; i <= r; sum += a[i++]);     else {         for (int i = l, end = (lb+1)\*m-1; i <= end; sum += a[i++]);         for (int i = lb+1; i < rb; sum += b[i++]);         for (int i = rb\*m; i <= r; sum += a[i++]);     }     printf("%d\n", sum); }  void **sqrt\_decomposition**(int a[], int n) {     int m = sqrt(n), q, l, r;     m += !(m\*m == n);     int b[m];     memset(b, 0, sizeof(b));     for (int i = 0; i < n; i++) b[i/m] += a[i];      for (scanf("%d", &q); q--; ) {         scanf("%d %d", &l, &r);         getsum(a, n, b, m, l, r);     } } |

----------------------------------------------------

**Segment Tree**

|  |
| --- |
| int arr[mx], tree[mx\*4];  void **init**(int node, int b, int e) {     if (b == e) tree[node] = arr[b];     else {         int left = node << 1, right = left | 1, mid = (b+e) >> 1;         init(left, b, mid);         init(right, mid+1, e);         tree[node] = tree[left] + tree[right];     } }  int **query**(int node, int b, int e, int i, int j) {     if (i > j) return 0;     if (i == b && j == e) return tree[node];     int left = node << 1, right = left | 1, mid = (b+e) >> 1;     return query(left, b, mid, i, min(mid, j)) + query(right, mid+1, e, max(mid+1, i), j); }   void **update**(int node, int b, int e, int i, int val) {     if (b == e) tree[node] = val;     else {         int left = node << 1, right = left | 1, mid = (b+e) >> 1;         if (i <= mid) update(left, b, mid, i, val);         else update(right, mid+1, e, i, val);         tree[node] = tree[left] + tree[right];     } } |

----------------------------------------------------

**Lazy propagation**

|  |
| --- |
| int arr[mx];  struct **info** {  int sum, prop; } tree[mx \* 4];  void **init**(int node, int b, int e) {  if (b == e) {  tree[node].sum = arr[b];  return;  }  int left = node << 1;  int right = (node << 1) + 1;  int mid = (b + e) >> 1;  init(left, b, mid);  init(right, mid + 1, e);  tree[node].sum = tree[left].sum + tree[right].sum; } |

|  |
| --- |
| int **query**(int node, int b, int e, int i, int j, int carry = 0) {  if (i > e || j < b) return 0;  if (b >= i && e <= j) return tree[node].sum + carry \* (e - b + 1);  int left = node << 1;  int right = (node << 1) + 1;  int mid = (b + e) >> 1;  int ls = query(left, b, mid, i, j, carry + tree[node].prop);  int rs = query(right, mid + 1, e, i, j, carry + tree[node].prop);  return ls + rs; }  void **update**(int node, int b, int e, int i, int j, int val) {  if (i > e || j < b) return;  if (b >= i && e <= j) {  tree[node].sum += ((e - b + 1) \* val);  tree[node].prop = val;  return;  }  int left = node << 1;  int right = (node << 1) + 1;  int mid = (b + e) >> 1;  update(left, b, mid, i, j, val);  update(right, mid + 1, e, i, j, val);  tree[node].sum = tree[left].sum + tree[right].sum + (e - b + 1) \* tree[node].prop; } |

----------------------------------------------------

**Trie Tree**

|  |
| --- |
| struct **node** {     bool isWord;     node\* next[26+1];      node() {  isWord = false;  for (int i = 0; i < 26; next[i++] = NULL;     } } \* root;  void **insert**(char\* str, int len) {     node\* cur = root;     for (int i = 0; i < len; i++) {  int id = str[i] - 'a';  if (cur->next[id] == NULL) cur->next[id] = new node();  cur = cur->next[id];     }     cur->isWord = true; }  bool **search**(char\* str, int len) {  node\* cur = root;  for (int i = 0; i < len; i++) {  int id = str[i] - 'a';  if (cur->next[id] == NULL) return false;  cur = cur->next[id];  }  return cur->isWord; } |

|  |
| --- |
| void **del**(node\* cur) {     for (int i = 0; i < 26; i++)  if (cur->next[i]) del(cur->next[i]);     delete(cur); } |

----------------------------------------------------

**Binary Index Tree**

|  |
| --- |
| const ll sz = 2e7+7; ll a[sz], b[sz];   void **update**(ll id, ll n) {  while (id <= n) {  b[id] += 1;  id += (id & -id);  } }   ll **query**(ll id) {  ll **sum** (0);  while (id >= 1) {  sum += b[id];  id -= (id & -id);   }  return sum; }  // Main method for (ll i = n-1; i >= 0; i--) {     if (a[i]) {  x += query(a[i]);  update(a[i] + 1, mx);     } } |

----------------------------------------------------

|  |
| --- |
| Created by: Amirul Islam      github.com/shiningflash |

COMMON ALGORITHM

----------------------------------------------------

**Binary Search**

|  |
| --- |
| int **binary\_search**(int arr[], int size, int key) {     int lo = 0, hi = size - 1, mid;     while (lo <= hi) {         mid = (lo + hi) >> 1;         if (key == arr[mid]) return mid;         if (key > arr[mid]) lo = mid + 1;         else hi = mid - 1;     }     return -1; }  int **searchLowerBound**(int arr[], int size, int key) {     int lo = 0, hi = size - 1, mid, indx = -1;     while (lo <= hi) {         mid = (lo + hi) >> 1;         if (key == arr[mid]) indx = mid, hi = mid - 1;         else if (key > arr[mid]) lo = mid + 1;         else hi = mid - 1;     }     return lo; } |

----------------------------------------------------

**Bisection**

|  |
| --- |
| double **mysqrt**(int n) {     int **cnt**(0);     double lo = 0.0, hi = n, mid;     while (hi - lo > .0000001) {         cnt++;         mid = (lo + hi) / 2.0;         if (mid \* mid > n) hi = mid;         else lo = mid;     }     printf("\nIteration needed: %d times\n", cnt);     return mid; } |

----------------------------------------------------

**Sieve (prime generator)**

|  |
| --- |
| // a[0] == 0, prime, a[1] == 1, not prime void **sieve**() {     a[0] = a[1] = 1;     for (int i = 2; i\*i <= mx; i++) {         if (!a[i]) {             for (int j = i<<1; j <= mx; j+=i) a[j] = 1;         }     } } |

----------------------------------------------------

**Bitwise sieve**

|  |
| --- |
| int a[(mx >> 5) + 2];  bool **Check**(int n, int pos) {     return (bool) (n & (1 << pos)); }  int **Set**(int n, int pos) {     return n = n | (1 << pos); }  void **sieve**() {     int rt = (int) sqrt(mx);     for (int i = 3; i <= rt; i += 2) {         if (Check(a[i >> 5], i & 31) == 0) {             for (int j = i \* i, k = i << 1; j <= mx; j += k) {                 a[j >> 5] = Set(a[j >> 5], j & 31);             }         }     }     printf("2");     for (int i = 3; i <= mx; i += 2) {         if (Check(a[i >> 5], i & 31) == 0) printf(" %d", i);     }     printf("\n"); } |

----------------------------------------------------

**Fermat little theorem**

|  |
| --- |
| // complexity - O (log(n)) int **Fermat\_Little\_Theorem**(int a, int n, int p) {     int res = 1;     a = a % p; // update if a >= p     while (n > 0) {         if (n & 1) res = (res \* a) % p;         n = n >> 1;         a = (a \* a) % p;     }     return res; }  bool **isPrime**(int n, int k) {     // corner cases  if (n < 2 || n == 4) return false;  if (n == 2 || n == 3) return true;       // try k times     while (k > 0) {         // pick a number in [2 .... n-2]         int a = 2 + rand() % (n-4);         if (\_\_gcd(n, a) != 1) return false;         if (Fermat\_Little\_Theorem(a, n-1, n) != 1) return false;         k--;     }  return true; } |

----------------------------------------------------

**Modular arithmetic**

|  |
| --- |
| int n = 100, m = 97, fact = 1; for (int i = 1; i <= n; i++) {     fact = ( (fact % m) \* (i % m) ) % m; } |

----------------------------------------------------

**Big Mod**

|  |
| --- |
| int **modolo**(int n, int p) {     if (p == 0) return 1;     if (p % 2 == 0) {         ll ret = modolo(n, p>>1);         return ((ret%m) \* (ret%m)) % m;     }     else return ((n%m) \* (modolo(n, p-1) % m)) % m; } |

----------------------------------------------------

**Two Pointer**

|  |
| --- |
| bool **hasPairSum**(int a[], int n, int key) {  for (int i = 0, j = n-1; i < j; ) {  if (a[i] + a[j] == key) return true;  else if (a[i] + a[j] > key) j--;  else i++;  }  return false; } |

----------------------------------------------------

**Meet in the middle**

|  |
| --- |
| int a[n], b[n], c[n], d[n]; for (int i = 0; i < n; i++) {     scanf("%d %d %d %d", &a[i], &b[i], &c[i], &d[i]); }  int X[n\*n], Y[n\*n]; for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  X[i\*n+j] = a[i] + b[j];  Y[i\*n+j] = c[i] + d[j];  } }  ll sum = 0; sort(Y, Y+n\*n); for (int i = 0; i < n\*n; i++) {  int lowpos = lower\_bound(Y, Y+n\*n, -X[i]) - Y;  if (Y[lowpos] == -X[i]) {  sum += ((upper\_bound(Y, Y+n\*n, -X[i])-Y) - lowpos);  } } printf("%lld\n", sum); |

----------------------------------------------------

BIT Magic

-------------------------------------------

|  |
| --- |
| x >>= 1; // Divide by 2 x <<= 1; // Multiply by 2 ch |= ' '; // Upper case to lower case ch &= '\_'; // Lower case to upper |

-------------------------------------------

**Clear all bits from LSB to ith bit**

|  |
| --- |
| mask = ~((1 << i+1 ) - 1); x &= mask; |

-------------------------------------------

**Clearing all bits from MSB to i-th bit**

|  |
| --- |
| mask = (1 << i) - 1; x &= mask; |

-------------------------------------------

**Checking if given 32 bit integer is power of 2**

|  |
| --- |
| int **isPowerof2**(int x)  {      return (x && !(x & x-1));  } |

-------------------------------------------

**Count set bits in integer**

|  |
| --- |
| int **countSetBits**(int x) {      int count = 0;      while (x)  {          x &= (x-1);          count++;      }      return count;  } |

-------------------------------------------

**Generate all possible subset**

|  |
| --- |
| void **possibleSubset**(int A[], int N) {     for (int i = 0; i < (1 << N); i++) {         for (int j = 0; j < N; j++) {             if (i & (1 << j)) {                 cout << A[j] << " ";             }         }         cout << endl;     } } |

-------------------------------------------

**i'th bit set, check, clear, flip**

|  |
| --- |
| set bit      : S |= (1<<i) check bit    : S & (1<<i) clear bit    : S &= ~(1<<i) flip bit     : S ^= ~(1<<i) |

|  |
| --- |
| Created by: Amirul Islam      github.com/shiningflash |