Competitive Programming Notebook

Daniel Santiago Lopez Daza
dlopezd@unal.edu.co

Tomás David Rodriguez Agudelo
trodrigueza@unal.edu.co

Eduard Joel Ostos Castro
eostos@unal.edu.co

Last updated: July 26, 2024

Contents

1 Template

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

#define int long long
#define pb push_back
#define pii pair<int, int>
#define vi vector<int>
#define vb vector<bool>
#define vpi vector<pair<int, int>>
#define vvi vector<vector<int>>
#define ff first
#define ss second
#define all(container) container.begin(), container.end()

void solve()
{
}

int32_t main()
{
#ifndef ONLINE_JUDGE
  freopen("input.txt", "r", stdin);
  freopen("out.txt", "w", stdout);
  freopen("error.txt", "w", stderr);
#endif
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
  int t; cin >> t;
  while(t--) solve();
}
```

2 Macros

1. **Debugging:**

   (a) `assert(x)`: If x is false, the program will terminate. ej:

   ```
   assert(1 == 2);
   // Will terminate the program

   ```

2. **Container related Macros:**

   (a) **all(x)**

   ```
   #define all(x) x.begin(), x.end()

   ```

   (b) **pb**

   ```
   #define pb push_back

   ```

3. **Types:**

   (a) **int**

   ```
   #define int long long

   ```

   (b) **vi, vb and vpi**

   ```
   #define vi vector<int>
   #define vb vector<bool>
   #define vpi vector<pair<int, int>>

   ```

   (c) **pi**

   ```
   #define pi pair<int, int>

   ```

   (d) **ff and ss**

   ```
   #define ff first
   #define ss second

   ```

3 Container Classes and Initialization

1. **Vector:**

   (a) Class:

   ```
   template < class T,
   class Alloc = allocator<T>> class vector;

   ```

   (b) Initialization:

      i. `vector<int> v`: Declares a vector of integers.
      ii. `vector<int> v(n)`: Declares a vector of integers of size n.
      iii. `vector<int> v(n, x)`: Declares a vector of integers of size n, with all elements initialized to x.
      iv. `vector<int> v = {1, 2, 3, 4}`: Declares a vector of integers with the elements 1, 2, 3 and 4.
      v. `vector<int> v {1, 2, 3, 4}`: Declares a vector of integers with the elements 1, 2, 3 and 4.

2. **Set:**

   (a) Class:

```cpp
template < class T,
class Compare = less<T>, \\ View Interesting Classes Chapter for more information
class Alloc = allocator<T>> class set;

```

(b) Initialization:

i. set<int> s: Declares a set of integers.

ii. set<int> s {1, 2, 3, 4}: Declares a set of integers with the elements 1, 2, 3 and 4.

iii. set<int> s = {1, 2, 3, 4}: Declares a set of integers with the elements 1, 2, 3 and 4.

(c) **Map:**

i. Class:

```cpp
    template <class Key, class T, class Compare = less<Key>, class Alloc = allocator<pair<const Key, T>>> class map;

```

ii. Initialization:

A. map<int, int> m: Declares a map with integer keys and values.

B. map<int, int> m {{1, 2}, {3, 4}}: Declares a map with the key-value pairs {1, 2} and {3, 4}.

C. map<int, int> m = {{1, 2}, {3, 4}}: Declares a map with the key-value pairs {1, 2} and {3, 4}.

(d) **Multiset:**

i. Class:

```cpp
    template <class T, class Compare = less<T>, class Alloc = allocator<T>> class multiset;

```

ii. Initialization:

A. multiset<int> ms: Declares a multiset of integers.

B. multiset<int> ms {1, 2, 2, 3, 4}: Declares a multiset of integers with the elements 1, 2, 2, 3, and 4.

C. multiset<int> ms = {1, 2, 2, 3, 4}: Declares a multiset of integers with the elements 1, 2, 2, 3, and 4.

(e) **Unordered Set:**

i. Class:

```cpp
    template <class Key, class Hash = hash<Key>, class Pred = equal_to<Key>, class Alloc = allocator<Key>> class unordered_set;

```

ii. Initialization:

A. unordered_set<int> us: Declares an unordered set of integers.

B. unordered_set<int> us {1, 2, 3, 4}: Declares an unordered set of integers with the elements 1, 2, 3, and 4.

C. unordered_set<int> us = {1, 2, 3, 4}: Declares an unordered set of integers with the elements 1, 2, 3, and 4.

(f) **Unordered Map:**

i. Class:

```cpp
    template <class Key, class T, class Hash = hash<Key>, class Pred = equal_to<Key>, class Alloc = allocator<pair<const Key, T>>> class unordered_map;

```

ii. Initialization:

A. unordered_map<int, int> um: Declares an unordered map with integer keys and values.

B. unordered_map<int, int> um {{1, 2}, {3, 4}}: Declares an unordered map with the key-value pairs {1, 2} and {3, 4}.

C. unordered_map<int, int> um = {{1, 2}, {3, 4}}: Declares an unordered map with the key-value pairs {1, 2} and {3, 4}.

(g) **Order Statistics Tree (using tree from <ext/pb_ds/assoc_container.hpp>):**

i. Class:

```cpp
    #include <ext/pb_ds/assoc_container.hpp>
    using namespace __gnu_pbds;
    template <class Key, class Mapped, class Cmp_Fn = std::less<Key>, class Tag = rb_tree_tag, class Node_Update = null_node_update, class Alloc = std::allocator<char>> class tree;

```

ii. Initialization:

A. tree<int, null_type, less<int>, rb_tree_tag, tree_order_statisti
ost: Declares an order statistics tree of integers.

B. tree<int, null_type, less<int>, rb_tree_tag, tree_order_statisti
ost {1, 2, 3, 4}: Declares an order statistics tree of integers with the elements 1, 2, 3, and 4.

C. tree<int, null_type, less<int>, rb_tree_tag, tree_order_statisti
ost = {1, 2, 3, 4}: Declares an order statistics tree of integers with the elements 1, 2, 3, and 4.

4 Functions

Containers Functions

1. **Vector:**

   (a) `push_back(x)`: Insert element x at the end of the vector. ej:

   ```
   vector<int> v;
   v.push_back(5);
   // New value of v = {5}

   ```

   Note: Amortized complexity of O(1).

   (b) `pop_back()`: Remove the last element of the vector. ej:

   ```
   vector<int> v ({5, 6, 7});
   v.pop_back();
   // New value of v = {5, 6}

   ```

   Note: Complexity of O(1).

   (c) `size()`: Return the number of elements in the vector. ej:

   ```
   vector<int> v ({5, 6, 7});
   v.size();
   // Will return 3

   ```

   Note: Complexity of O(1).

   (d) `at(i)`: Access the element at index i. ej:

   ```
   vector<int> v ({5, 6, 7});
   v.at(1);
   // Will return 6

   ```

   Note: Complexity of O(1).

   (e) `clear()`: Remove all elements from the vector. ej:

   ```
   vector<int> v ({5, 6, 7});
   v.clear();
   // New value of v = {}

   ```

   Note: Complexity of O(n).

2. **Map:**

   (a) `insert({key, value})`: Insert key-value pair in the map. ej:

   ```
   map<int, int> m;
   m.insert({5, 10});
   // New value of m = { {5, 10} }

   ```

   Note: Complexity of O($log_2 n$).

   (b) `erase(key)`: Erase element with the given key from the map. ej:

   ```
   map<int, int> m ({ {5, 10}, {6, 20} });
   m.erase(5);
   // New value of m = { {6, 20} }

   ```

   Note: Complexity of O($log_2 n$).

   (c) `find(key)`: Find element with the given key in the map. ej:

   ```
   map<int, int> m;
   m.find(5);
   // Will return m.end() if key is not in the map

   ```

   Note: Complexity of O($log_2 n$).

   (d) `lower_bound(key)`: Find the first element that is not less than the given key. ej:

   ```
   map<int, int> m ({ {5, 10}, {6, 20} });
   m.lower_bound(6);
   // Will return an iterator to the element {6, 20}

   ```

   Note: Complexity of O($log_2 n$). **Note:** Uses Binary Search under the hood.

   (e) `upper_bound(key)`: Find the first element that is greater than the given key. ej:

   ```
   map<int, int> m ({ {5, 10}, {6, 20} });
   m.upper_bound(6);
   // Will return an iterator to the end of the map

   ```

   Note: Complexity of O($log_2 n$). **Note:** Uses Binary Search under the hood.

3. **Multiset:**

   (a) `insert(x)`: Insert element x in the multiset. ej:

   ```
   multiset<int> ms;
   ms.insert(5);
   // New value of ms = {5}

   ```

   Note: Complexity of O($log_2 n$).

   (b) `erase(x)`: Erase all instances of element x from the multiset. ej:

   ```
   multiset<int> ms ({5, 6, 7, 5});
   ms.erase(5);
   // New value of ms = {6, 7}

   ```

Note: Complexity of O($log_2 n$).

(c) `find(x)`: Find an instance of element x in the multiset. ej:

```cpp
multiset<int> ms;
ms.find(5);
// Will return ms.end() if x is not in the multiset

```

Note: Complexity of O($log_2 n$).

(d) `lower_bound(x)`: Find the first element that is not less than x. ej:

```cpp
multiset<int> ms ({5, 6, 7});
ms.lower_bound(6);
// Will return an iterator to the element 6

```

Note: Complexity of O($log_2 n$). **Note:** Uses Binary Search under the hood.

(e) `upper_bound(x)`: Find the first element that is greater than x. ej:

```cpp
multiset<int> ms ({5, 6, 7});
ms.upper_bound(6);
// Will return an iterator to the element 7

```

Note: Complexity of O($log_2 n$). **Note:** Uses Binary Search under the hood.

4. **Unordered Set:**

(a) `insert(x)`: Insert element x in the unordered set. ej:

```cpp
unordered_set<int> us;
us.insert(5);
// New value of us = {5}

```

Note: Average complexity of O(1), worst-case complexity of O(n).

(b) `erase(x)`: Erase element x from the unordered set. ej:

```cpp
unordered_set<int> us ({5, 6, 7});
us.erase(5);
// New value of us = {6, 7}

```

Note: Average complexity of O(1), worst-case complexity of O(n).

(c) `find(x)`: Find element x in the unordered set. ej:

```cpp
unordered_set<int> us;
us.find(5);
// Will return us.end() if x is not in the unordered set

```

Note: Average complexity of O(1), worst-case complexity of O(n).

(d) `bucket_count()`: Return the number of buckets in the unordered set. ej:

```cpp
unordered_set<int> us ({5, 6, 7});
us.bucket_count();
// Will return the number of buckets

```

Note: Complexity of O(1).

(e) `load_factor()`: Return the load factor of the unordered set. ej:

```cpp
unordered_set<int> us ({5, 6, 7});
us.load_factor();
// Will return the load factor

```

Note: Complexity of O(1).

5. **Unordered Map:**

(a) `insert({key, value})`: Insert key-value pair in the unordered map. ej:

```cpp
unordered_map<int, int> um;
um.insert({5, 10});
// New value of um = { {5, 10} }

```

Note: Average complexity of O(1), worst-case complexity of O(n).

(b) `erase(key)`: Erase element with the given key from the unordered map. ej:

```cpp
unordered_map<int, int> um ({ {5, 10}, {6, 20} });
um.erase(5);
// New value of um = { {6, 20} }

```

Note: Average complexity of O(1), worst-case complexity of O(n).

(c) `find(key)`: Find element with the given key in the unordered map. ej:

```cpp
unordered_map<int, int> um;
um.find(5);
// Will return um.end() if key is not in the unordered map

```

Note: Average complexity of O(1), worst-case complexity of O(n).

(d) `bucket_count()`: Return the number of buckets in the unordered map. ej:

```cpp
unordered_map<int, int> um ({ {5, 10}, {6, 20} });
um.bucket_count();
// Will return the number of buckets

```

Note: Complexity of O(1).

(e) `load_factor()`: Return the load factor of the unordered map. ej:

```cpp
unordered_map<int, int> um ({ {5, 10}, {6, 20} });
um.load_factor();
// Will return the load factor

```

Note: Complexity of O(1).

6. **Order Statistics Tree (using `tree` from `<ext/pb_ds/assoc_container.hpp>`):**

   (a) `insert(x)`: Insert element x in the order statistics tree. ej:

   ```
   #include <ext/pb_ds/assoc_container.hpp>
   using namespace __gnu_pbds;
   tree<int, null_type, less<int>, rb_tree_tag,
     tree_order_statistics_node_update> ost;
   ost.insert(5);
   // New value of ost = {5}
   ```

   Note: Complexity of O($log_2 n$).

   (b) `erase(x)`: Erase element x from the order statistics tree. ej:

   ```
   #include <ext/pb_ds/assoc_container.hpp>
   using namespace __gnu_pbds;
   tree<int, null_type, less<int>, rb_tree_tag,
     tree_order_statistics_node_update> ost ({5, 6, 7});
   ost.erase(5);
   // New value of ost = {6, 7}
   ```

   Note: Complexity of O($log_2 n$).

   (c) `find_by_order(k)`: Find the k-th smallest element in the order statistics tree. ej:

   ```
   #include <ext/pb_ds/assoc_container.hpp>
   using namespace __gnu_pbds;
   tree<int, null_type, less<int>, rb_tree_tag,
     tree_order_statistics_node_update> ost ({5, 6, 7});
   ost.find_by_order(1);
   // Will return an iterator to the element 6
   ```

   Note: Complexity of O($log_2 n$).

   (d) `order_of_key(x)`: Find the number of elements strictly less than x in the order statistics tree. ej:

   ```
   #include <ext/pb_ds/assoc_container.hpp>
   using namespace __gnu_pbds;
   tree<int, null_type, less<int>, rb_tree_tag,
     tree_order_statistics_node_update> ost ({5, 6, 7});
   ost.order_of_key(6);
   // Will return 1
   ```

   Note: Complexity of O($log_2 n$).

   (e) `clear()`: Remove all elements from the order statistics tree. ej:

   ```
   #include <ext/pb_ds/assoc_container.hpp>
   using namespace __gnu_pbds;
   tree<int, null_type, less<int>, rb_tree_tag,
     tree_order_statistics_node_update> ost ({5, 6, 7});
   ost.clear();
   // New value of ost = {}
   ```

   Note: Complexity of O(n).

7. **Set:**

   (a) `insert(x)`: Insert element x in the set. ej:

   ```
   set<int> s;
   s.insert(5);
   // New value of s = {5}
   ```

   Note: Complexity of O($log_2 n$).

   (b) `erase(x)`: Erase element x from the set. ej:

   ```
   set<int> s ({5, 6, 7});
   s.erase(5);
   // New value of s = {6, 7}
   ```

   Note: Complexity of O($log_2 n$).

   (c) `find(x)`: Find element x in the set. ej:

   ```
   set<int> s;
   s.find(5);
   // Will return s.end() if x is not in the set
   ```

   Note: Complexity of O($log_2 n$).

   (d) `lower_bound(x)`: Find the first element that is not less than x. ej:

   ```
   set<int> s ({5, 6, 7});
   s.lower_bound(6);
   // Will return an iterator to the element 6
   ```

   Note: Complexity of O($log_2 n$). **Note:** Uses Binary Search under the hood.

   (e) `upper_bound(x)`: Find the first element that is greater than x. ej:

   ```
   set<int> s ({5, 6, 7});
   s.upper_bound(6);
   // Will return an iterator to the element 7
   ```

   Note: Complexity of O($log_2 n$). **Note:** Uses Binary Search under the hood.

5 Functions

Graphs

1. **DFS:**

```
#include <iostream>

struct Node {
    int data;
    Node* left;
    Node* right;

    Node(int value) : data(value), left(nullptr), right(nullptr) {}
};

```

Preorder:

```
void preorder(Node* node) {
  if (node == nullptr) return;
  std::cout << node->data << " ";
  preorder(node->left);
  preorder(node->right);
}

```

```
void inorder(Node* node) {
  if (node == nullptr) return;
  inorder(node->left);
  std::cout << node->data << " ";
  inorder(node->right);
}

```

```
void postorder(Node* node) {
  if (node == nullptr) return;
  postorder(node->left);
  postorder(node->right);
  std::cout << node->data << " ";
}

```

Dfs:

```
    #include <iostream>
#include <vector>

using namespace std;

void DFSUtil(int v, vector<bool> &visited, const vector<vector<int>>
    &adj) {
    visited[v] = true;
    cout << v << " ";

    for (int i : adj[v]) {
        if (!visited[i]) {
            DFSUtil(i, visited, adj);
        }
    }
}

void DFS(int V, const vector<vector<int>> &adj) {
    vector<bool> visited(V, false);

    for (int i = 0; i < V; ++i) {
        if (!visited[i]) {
            DFSUtil(i, visited, adj);
        }
    }
}

int main() {
    int V = 5;
    vector<vector<int>> adj(V);

    adj[0].push_back(1);
    adj[0].push_back(2);
    adj[1].push_back(3);
    adj[2].push_back(4);

    cout << "DFS starting from vertex 0:\n";
    DFS(V, adj);

    return 0;
}


```

2. BFS:

```
  #include <iostream>
#include <vector>
#include <queue>

using namespace std;

void BFS(int start, int V, const vector<vector<int>> &adj) {
    vector<bool> visited(V, false);
    queue<int> q;

    visited[start] = true;
    q.push(start);

    while (!q.empty()) {
        int v = q.front();
        q.pop();
        cout << v << " ";

```

```
        for (int i : adj[v]) {
            if (!visited[i]) {
                visited[i] = true;
                q.push(i);
            }
        }
    }
}

int main() {
    int V = 5;
    vector<vector<int>> adj(V);

    adj[0].push_back(1);
    adj[0].push_back(2);
    adj[1].push_back(3);
    adj[2].push_back(4);

    cout << "BFS starting from vertex 0:\n";
    BFS(0, V, adj);

    return 0;
}


```

3. Kruskal:

```
#include <iostream>
#include <vector>
#include <algorithm>

  using namespace std;

  struct Edge {
    int src, dest, weight;
  };

struct Graph {
  int V, E;
  vector<Edge> edges;
};

struct subset {
  int parent;
  int rank;
};

int find(subset subsets[], int i) {
  if (subsets[i].parent != i) {
    subsets[i].parent = find(subsets, subsets[i].parent);
  }
  return subsets[i].parent;
}

void Union(subset subsets[], int x, int y) {
  int xroot = find(subsets, x);
  int yroot = find(subsets, y);

  if (subsets[xroot].rank < subsets[yroot].rank) {
    subsets[xroot].parent = yroot;
  } else if (subsets[xroot].rank > subsets[yroot].rank) {
    subsets[yroot].parent = xroot;
  } else {
    subsets[yroot].parent = xroot;
    subsets[xroot].rank++;
  }
}

void KruskalMST(Graph& graph) {
  vector<Edge> result;
  int V = graph.V;
  sort(graph.edges.begin(), graph.edges.end(), [](Edge a, Edge b) {
      return a.weight < b.weight;
      });

  subset* subsets = new subset[(V * sizeof(subset))];
  for (int v = 0; v < V; ++v) {
    subsets[v].parent = v;
    subsets[v].rank = 0;
  }

  for (Edge e : graph.edges) {
    int x = find(subsets, e.src);
    int y = find(subsets, e.dest);

    if (x != y) {
      result.push_back(e);
      Union(subsets, x, y);
    }
  }

  cout << "Edges in the MST:\n";
  for (Edge e : result) {
    cout << e.src << " -- " << e.dest << " == " << e.weight << endl;
  }

  delete[] subsets;
}

int main() {
  Graph graph;
  graph.V = 4;
  graph.E = 5;
  graph.edges = {
    {0, 1, 10},
    {0, 2, 6},
    {0, 3, 5},
    {1, 3, 15},
    {2, 3, 4}
```

```
    };

    KruskalMST(graph);

    return 0;
}


```

4. Prims:

```
  #include <iostream>
#include <vector>
#include <climits>

using namespace std;

int minKey(const vector<int>& key, const vector<bool>& mstSet, int V)
    {
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (!mstSet[v] && key[v] < min)
            min = key[v], min_index = v;

    return min_index;
}

void PrimMST(const vector<vector<int>>& graph, int V) {
    vector<int> parent(V);
    vector<int> key(V, INT_MAX);
    vector<bool> mstSet(V, false);

    key[0] = 0;
    parent[0] = -1;

    for (int count = 0; count < V - 1; count++) {
        int u = minKey(key, mstSet, V);
        mstSet[u] = true;

        for (int v = 0; v < V; v++)
            if (graph[u][v] && !mstSet[v] && graph[u][v] < key[v])
                parent[v] = u, key[v] = graph[u][v];
    }

    cout << "Edge \tWeight\n";
    for (int i = 1; i < V; i++)
        cout << parent[i] << " - " << i << " \t" << graph[i][parent[i
    ]] << " \n";
}

int main() {
    int V = 5;
    vector<vector<int>> graph = {
        {0, 2, 0, 6, 0},
        {2, 0, 3, 8, 5},
        {0, 3, 0, 0, 7},
        {6, 8, 0, 0, 9},
        {0, 5, 7, 9, 0}
    };

    PrimMST(graph, V);

    return 0;
}


```

5. Dijkstra:

```
  #include <iostream>
#include <vector>
#include <queue>
#include <climits>

using namespace std;

void dijkstra(const vector<vector<pair<int, int>>> &graph, int src) {
    int V = graph.size();
    vector<int> dist(V, INT_MAX);
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<
    pair<int, int>>> pq;

    pq.push({0, src});
    dist[src] = 0;

    while (!pq.empty()) {
        int u = pq.top().second;
        pq.pop();

        for (auto &neighbor : graph[u]) {
            int v = neighbor.first;
            int weight = neighbor.second;

            if (dist[u] + weight < dist[v]) {
                dist[v] = dist[u] + weight;
                pq.push({dist[v], v});
            }
        }
    }

    cout << "Vertex \t Distance from Source\n";
    for (int i = 0; i < V; ++i)
        cout << i << " \t " << dist[i] << "\n";
}

int main() {
    int V = 5;
    vector<vector<pair<int, int>>> graph(V);

```

```
    graph[0].push_back({1, 10});
    graph[0].push_back({4, 5});
    graph[1].push_back({2, 1});
    graph[1].push_back({4, 2});
    graph[2].push_back({3, 4});
    graph[3].push_back({0, 7});
    graph[3].push_back({2, 6});
    graph[4].push_back({1, 3});
    graph[4].push_back({2, 9});
    graph[4].push_back({3, 2});

    dijkstra(graph, 0);

    return 0;
}


```

6. Bellman:

```
 #include <iostream>
#include <vector>
#include <climits>

using namespace std;

struct Edge {
    int src, dest, weight;
};

void bellmanFord(const vector<Edge> &edges, int V, int src) {
    vector<int> dist(V, INT_MAX);
    dist[src] = 0;

    for (int i = 1; i <= V - 1; ++i) {
        for (const auto &edge : edges) {
            if (dist[edge.src] != INT_MAX && dist[edge.src] + edge.weight < dist[edge.dest]) {
                dist[edge.dest] = dist[edge.src] + edge.weight;
            }
        }
    }

    for (const auto &edge : edges) {
        if (dist[edge.src] != INT_MAX && dist[edge.src] + edge.weight < dist[edge.dest]) {
            cout << "Graph contains negative weight cycle\n";
            return;
        }
    }

    cout << "Vertex \t Distance from Source\n";
    for (int i = 0; i < V; ++i)
        cout << i << " \t " << dist[i] << "\n";
}

int main() {
    int V = 5;
    vector<Edge> edges = {
        {0, 1, -1},
        {0, 2, 4},
        {1, 2, 3},
        {1, 3, 2},
        {1, 4, 2},
        {3, 2, 5},
        {3, 1, 1},
        {4, 3, -3}
    };

    bellmanFord(edges, V, 0);

    return 0;
}

```

6 Strings

1. Knuth Morris Pratt Encontrar todas las ocurrencias de un patrón en un texto.

```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

void computeLPSArray(const string& pat, vector<int>& lps) {
    int length = 0;
    int i = 1;
    lps[0] = 0;

    while (i < pat.size()) {
        if (pat[i] == pat[length]) {
            length++;
            lps[i] = length;
            i++;
        } else {
            if (length != 0) {
                length = lps[length - 1];
            } else {
                lps[i] = 0;
                i++;
            }
        }
    }
}

void KMPSearch(const string& pat, const string& txt) {
    int M = pat.size();
```

```
    int N = txt.size();

    vector<int> lps(M);
    computeLPSArray(pat, lps);

    int i = 0;
    int j = 0;
    while (i < N) {
        if (pat[j] == txt[i]) {
            j++;
            i++;
        }

        if (j == M) {
            cout << "Found pattern at index " << i - j << endl;
            j = lps[j - 1];
        } else if (i < N && pat[j] != txt[i]) {
            if (j != 0) {
                j = lps[j - 1];
            } else {
                i++;
            }
        }
    }
}

int main() {
    string txt = "ABABDABACDABABCABAB";
    string pat = "ABABCABAB";
    KMPSearch(pat, txt);
    return 0;
}
```

2. Rabin-Karp Algoritmo de búsqueda de patrones que utiliza una función hash para encontrar una subcadena en un texto.

```
  #include <iostream>
#include <string>

using namespace std;

#define d 256
const int q = 101; // A prime number

void RabinKarpSearch(const string& pat, const string& txt) {
    int M = pat.size();
    int N = txt.size();
    int i, j;
    int p = 0; // Hash value for pattern
    int t = 0; // Hash value for txt
    int h = 1;

    for (i = 0; i < M - 1; i++)
        h = (h * d) % q;

    for (i = 0; i < M; i++) {
        p = (d * p + pat[i]) % q;
        t = (d * t + txt[i]) % q;
    }

    for (i = 0; i <= N - M; i++) {
        if (p == t) {
            for (j = 0; j < M; j++) {
                if (txt[i + j] != pat[j])
                    break;
            }
            if (j == M)
                cout << "Pattern found at index " << i << endl;
        }

        if (i < N - M) {
            t = (d * (t - txt[i] * h) + txt[i + M]) % q;
            if (t < 0)
                t = (t + q);
        }
    }
}

int main() {
    string txt = "GEEKS FOR GEEKS";
    string pat = "GEEK";
    RabinKarpSearch(pat, txt);
    return 0;
}
```

3. Z Algorithm Encontrar todas las ocurrencias de un patrón en un texto.

```
  #include <iostream>
#include <vector>
#include <string>

using namespace std;

vector<int> calculateZ(const string& s) {
    int n = s.size();
    vector<int> Z(n);
    int L = 0, R = 0, K;

    for (int i = 1; i < n; ++i) {
        if (i > R) {
            L = R = i;
            while (R < n && s[R] == s[R - L])
                R++;
            Z[i] = R - L;
            R--;
        } else {
            K = i - L;
```

```
            if (Z[K] < R - i + 1)
                Z[i] = Z[K];
            else {
                L = i;
                while (R < n && s[R] == s[R - L])
                    R++;
                Z[i] = R - L;
                R--;
            }
        }
    }
    return Z;
}

void ZSearch(const string& text, const string& pattern) {
    string concat = pattern + "$" + text;
    vector<int> Z = calculateZ(concat);

    for (int i = 0; i < Z.size(); ++i) {
        if (Z[i] == pattern.size())
            cout << "Pattern found at index " << i - pattern.size() -
     1 << endl;
    }
}

int main() {
    string text = "GEEKS FOR GEEKS";
    string pattern = "GEEK";
    ZSearch(text, pattern);
    return 0;
}

```

4. Longest Common Subsequence Encuentra la subsecuencia común más larga entre dos secuencias.

```
  #include <iostream>
#include <vector>
#include <string>

using namespace std;

int LCS(const string& X, const string& Y) {
    int m = X.size();
    int n = Y.size();
    vector<vector<int>> L(m + 1, vector<int>(n + 1));

    for (int i = 0; i <= m; i++) {
        for (int j = 0; j <= n; j++) {
            if (i == 0 || j == 0)
                L[i][j] = 0;
            else if (X[i - 1] == Y[j - 1])
                L[i][j] = L[i - 1][j - 1] + 1;
            else
                L[i][j] = max(L[i - 1][j], L[i][j - 1]);
        }
    }
    return L[m][n];
}

int main() {
    string X = "AGGTAB";
    string Y = "GXTXAYB";
    cout << "Length of LCS is " << LCS(X, Y) << endl;
    return 0;
}

```

5. Longest Palindromic Subsequence: Encuentra la subcadena palindrómica más larga dentro de una cadena dada

```
  #include <iostream>
#include <string>

using namespace std;

string longestPalindromicSubstring(const string& s) {
    int n = s.size();
    if (n == 0) return "";

    int start = 0, maxLength = 1;

    vector<vector<bool>> table(n, vector<bool>(n, false));

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

    for (int i = 0; i < n - 1; ++i) {
        if (s[i] == s[i + 1]) {
            table[i][i + 1] = true;
            start = i;
            maxLength = 2;
        }
    }

    for (int k = 3; k <= n; ++k) {
        for (int i = 0; i < n - k + 1; ++i) {
            int j = i + k - 1;

            if (table[i + 1][j - 1] && s[i] == s[j]) {
                table[i][j] = true;

                if (k > maxLength) {
                    start = i;
                    maxLength = k;
                }
            }
        }
    }

```

```
    return s.substr(start, maxLength);
}

int main() {
    string s = "babad";
    cout << "Longest Palindromic Substring is " <<
    longestPalindromicSubstring(s) << endl;
    return 0;
}
```