Notebook - Maratona de Programação

Cabo HDMI, VGA, USB

Contents

1 Graph

1.1 Bellman Ford

Complexity: O(n * m) | n = |nodes|, m = |edges|
Finds shortest paths from a starting node to all nodes of the graph. The node can have negative cycle and belman-ford will detected

```
// TITLE: Bellman Ford
// COMPLEXITY: O(n * m) | n = |nodes|, m = |edges|
// DESCRIPTION: Finds shortest paths from a starting node to all nodes of the graph. The node can have negative cycle and belman-ford will detected

// a and b vertices, c cost
// [{a, b, c}, {a, b, c}]
vector<tuple<int, int, int>> edges;
int N;

void bellman_ford(int x){
    for (int i = 0; i < N; i++){
        dist[i] = oo;
    }
    dist[x] = 0;

    for (int i = 0; i < N - 1; i++){
        for (auto [a, b, c]: edges){
            if (dist[a] == oo) continue;
            dist[b]= min(dist[b], dist[a] + w);
        }
    }
}
// return true if has cycle
bool check_negative_cycle(int x){
    for (int i = 0; i < N; i++){
        dist[i] = oo;
    }
    dist[x] = 0;

    for (int i = 0; i < N - 1; i++){
        for (auto [a, b, c]: edges){
            if (dist[a] == oo) continue;
            dist[b]= min(dist[b], dist[a] + w);
        }
    }

    for (auto [a, b, c]: edges){
        if (dist[a] == oo) continue;
        if (dist[a] + w < dist[b]){
            return true;
        };
    }
    return false;
}
```

1.2 Topological Sort

Complexity: O(N + M), N: Vertices, M: Arestas
Retorna no do grapho em ordem topologica, se a quantidade de nos retornada nao for igual a quantidade de nos e impossivel

```
// TITLE: Topological Sort
// COMPLEXITY: O(N + M), N: Vertices, M: Arestas
// DESCRIPTION: Retorna no do grapho em ordem topologica, se a quantidade de nos retornada nao for igual a quantidade de nos e impossivel

typedef vector<vector<int>> Adj_List;
typedef vector<int> Indegree_List; // How many nodes depend on him
typedef vector<int> Order_List;    // The order in which the nodes appears

Order_List kahn(Adj_List adj, Indegree_List indegree)
{
    queue<int> q;
    // priority_queue<int> q; // If you want in lexicografic order
    for (int i = 0; i < indegree.size(); i++) {
        if (indegree[i] == 0)
            q.push(i);
    }
    vector<int> order;

    while (not q.empty()) {
        auto a = q.front();
        q.pop();

        order.push_back(a);
        for (auto b: adj[a]) {
            indegree[b]--;
            if (indegree[b] == 0)
                q.push(b);
        }
    }
    return order;
}

int32_t main()
{

    Order_List = kahn(adj, indegree);
    if (Order_List.size() != N) {
        cout << "IMPOSSIBLE" << endl;
    }
    return 0;
}
```

2 String

2.1 Z function

Complexity: Z function complexity
z function

```
// TITLE: Z function
// COMPLEXITY: Z function complexity
// DESCRIPTION: z function

void z_function(string& s)
{
    return;
}
```

3 Set

3.1 Ordered Set

Complexity: O(log(n))

```
// TITLE: Ordered Set
// COMPLEXITY: O(log(n))
// DESCRIPION: Set but you can look witch elements is in position (k)

```

```cpp
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

#define ordered_set tree<int, null_type,less<int>, rb_tree_tag,tree_order_statistics_node_update>

int32_t main() {
    ordered_set o_set;

    o_set.insert(5);
    o_set.insert(1);
    o_set.insert(2);
    // o_set = {1, 2, 5}
    5 == *(o_set.find_by_order(2));
    2 == o_set.order_of_key(4); // {1, 2}
}
```

3.2 Set

Complexity: Insertion Log(n)
Keeps elements sorted, remove duplicates, upper_bound, lower_bound, find, count

```cpp
// TITLE: Set
// COMPLEXITY: Insertion Log(n)
// Description: Keeps elements sorted, remove duplicates, upper_bound, lower_bound, find, count

int main() {
  set<int> set1;

  set1.insert(1);       // O(log(n))
  set1.erase(1);        // O(log(n))

  set1.upper_bound(1);  // O(log(n))
  set1.lower_bound(1);  // O(log(n))
  set1.find(1);         // O(log(n))
  set1.count(1);        // O(log(n))

  set1.size();          // O(1)
  set1.empty();         // O(1)

  set1.clear()          // O(1)
  return 0;
}
```

3.3 Multiset

Complexity: O(log(n))
Same as set but you can have multiple elements with same values

```cpp
// TITLE: Multiset
// COMPLEXITY: O(log(n))
// DESCRIPTION: Same as set but you can have multiple elements with same values

int main() {
  multiset<int> set1;
}
```