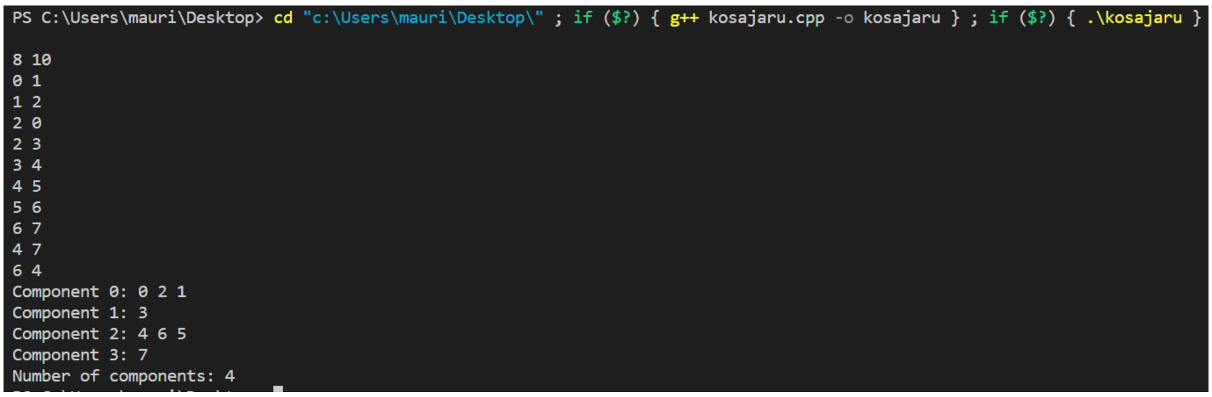
Repositorio:

https://github.com/mauriciocarazas/Competitive-Programming-/blob/main/KOSARAJU'S%20ALGORITHM.cpp

```cpp
//kosajaru´s algorithm

#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>
#include <string>
#include <stack>
#include <queue>
#include <limits.h>

#define MAX_N 20001
#define ll long long int
using namespace std;


//Here dies the macros
int n, m;   //n = number of nodes, m = number of edges


struct Node {
  vector < int > adj; //adjacency list
  vector < int > rev_adj; //reverse adjacency list
};

Node g[MAX_N]; //graph

stack < int > S;        //stack for Kosaraju's algorithm
bool visited[MAX_N];    //visited array for Kosaraju's algorithm

int component[MAX_N];  //component[i] = component of node i
vector < int > components[MAX_N]; //components[i] = list of nodes in component i
int numComponents; //number of components

void dfs_1(int x) { //first dfs for Kosaraju's algorithm
```

```
  visited[x] = true; //mark node as visited
  for (int i = 0; i < g[x].adj.size(); i++) { //for each neighbor
    if (!visited[g[x].adj[i]]) dfs_1(g[x].adj[i]); //if not visited, visit it
  }
  S.push(x); //push node to stack when done
}

void dfs_2(int x) { //second dfs for Kosaraju's algorithm
  cout << x << " ";  //print node
  component[x] = numComponents; //assign component to node
  components[numComponents].push_back(x); //add node to component
  visited[x] = true; //mark node as visited
  for (int i = 0; i < g[x].rev_adj.size(); i++) { //for each neighbor
    if (!visited[g[x].rev_adj[i]]) dfs_2(g[x].rev_adj[i]); //if not visited,
visit it
  }
}

void Kosaraju() { //Kosaraju's algorithm
  for (int i = 0; i < n; i++) //for each node
    if (!visited[i]) dfs_1(i); //if not visited, visit it

  for (int i = 0; i < n; i++) //for each node
    visited[i] = false; //mark all nodes as unvisited

  while (!S.empty()) { //while stack is not empty
    int v = S.top(); //get top of stack
    S.pop(); //pop top of stack
    if (!visited[v]) { //if node is not visited
      cout << "Component " << numComponents << ": "; //print component number
      dfs_2(v); //visit node and all nodes connected to it
      numComponents++; //increment number of components
      cout << endl;
    }
  }
}

int main() {

  cin >> n >> m; //read in number of nodes and edges
  int a, b; //nodes
  while (m--) { //for each edge
```

```cpp
    cin >> a >> b;
    g[a].adj.push_back(b); //add edge to adjacency list
    g[b].rev_adj.push_back(a); //add edge to reverse adjacency list
  }

  Kosaraju(); //run Kosaraju's algorithm
  cout << "Number of components: " << numComponents << endl;
  //Complexity: O(V + E)

  return 0;
}
```



```
PS C:\Users\mauri\Desktop> cd "c:\Users\mauri\Desktop\" ; if ($?) { g++ kosajaru.cpp -o kosajaru } ; if ($?) { .\kosajaru }

8 10
0 1
1 2
2 0
2 3
3 4
4 5
5 6
6 7
4 7
6 4
Component 0: 0 2 1
Component 1: 3
Component 2: 4 6 5
Component 3: 7
Number of components: 4
```