Report: Compressed Graph Project

Delacroix Nicolas

Masi Lucca

Overall understanding of the project:

The concept of compressed graphs depends a lot on the way we want to use them. Depending on the type of problem we want to solve or the system we want to create, the implementation may differ. In the most common way, a graph is represented by an adjacency matrix of length Number of vertex squared. In our early draft, we wanted to use a matrix of chain list to reduce the unused spaces in the adjacency matrix. However, with the advice of our teacher we chose to use BST (binary search trees). A binary search tree is a data structure that organizes elements in a hierarchical manner and enables efficient searching, insertion, and deletion operations. It follows the binary search property, which states that for any given node in the tree, the values in its left subtree are smaller than the node's value, and the values in its right subtree are greater. Using BST doesn't create a real compressed graph, as the data is not compressed using any compression methods. However, it is more optimized as each node is already sorted in a way and so the use of BST order function is easier.

Our graph is then represented by:

- The number of Nodes in the hole graph.
- A List of Binary search trees (called node) that represent each vertex and the vertex it is linked to.

Each node is composed of:

- A data, wich is the number of vertex and the index-1 of the vertex in the list of nodes of the graph.
- A matched variable used in the matching algorithm
- Addresses to nodes for its left and right child (BST)
- The number of connections the vertex makes.

We extract all the graph information we need from a text file (data.txt). The first line is the total number of vertex (from 1 to + ∞) then it is all the lines of the graph, each line being a vertex, the first column being the number of edges, and then all the vertex it is linked to.

The organization of the project is simple, graph.c and graph.h are the files concerning the gestion of our graph. Same concerning queue.c, queue.h, stack.c, stack.h they are here to implement stacks and queues. Function.c and function.h are the other functions that are not applied directly to the graph. Main.c is the main file calling everything needed to implement our graph.

DFS, BFS:

DFS (Depth-First Search) and BFS (Breadth-First Search) are two fundamental graph traversal algorithms used in graph theory to explore or search through the nodes (vertices) and edges of a graph. Both DFS and BFS have their own strengths and use cases. DFS is often used for tasks such as finding connected components, topological sorting, and detecting cycles in a graph. BFS is commonly used for finding the shortest path, solving puzzles, and traversing a tree or graph level by level. The choice between DFS and BFS depends on the specific requirements and characteristics of the problem at hand.

In our implementation we chose, due to time management, to implement only DFS. Implementing it was challenging as each node is a BST which means that the next vertex is either the left child or the right child. That was our first main problem. Then we had a problem accessing the linked nodes of other nodes. Problem that we resolved as it was only an else if problem.

Topological Sort:

Topological sort is an algorithm used in graph theory to sort the vertices of a directed acyclic graph (DAG) in a linear order such that for every directed edge (u, v), vertex u comes before v in the ordering. In other words, the algorithm produces a sequence of the nodes in such a way that each node comes before any node that depends on it.

The topological sort algorithm works by repeatedly selecting a vertex that has no incoming edges, adding it to the sorted sequence, and removing it from the graph along with all of its outgoing edges. This process continues until no vertices remain in the graph.

In our implementation, we had to duplicate the standards loop operation for topological sort as we must go either to the left child or the right child. After that, the algorithm was pretty simple to implement.

Matching Algorithm:

In graph theory, matching algorithms are used to find specific types of matchings in a graph. A matching is a subset of edges in a graph such that no two edges share a common vertex. Matching algorithms are typically applied to bipartite graphs, which are graphs whose vertices can be divided into two disjoint sets, such that all edges connect a vertex from one set to a vertex in the other set.

However, due to a time management problem we weren't able to implement it.