Graph Library

The library code written can be used by other programs to implement their graph algorithms. The library is a single source file that implements the prototypes found in the graph.h header file.

Creates the graph and maintains it using an Adjacency Matrix, which is a two-dimensional static matrix inside of the Graph struct. Each vertex will have a single int as its value (label), which will have a non-negative value. Each edge will only be between two existing vertices and each will have a positive weight greater than 0. Each vertex may also have an edge to itself (self-loop). Graph supports operations on a directed, weighted graph that may or may not be connected and may have cycles. All edges must have a positive weight (> 0). Graph may have disconnected vertices without any edges.

The implementation of the Adjacency Matrix uses the struct definition in graph.h

```
typedef struct graph_struct {
  int max_vertex;
  int adj_matrix[MAX_VERTICES][MAX_VERTICES];
  int visited[MAX_VERTICES];
} Graph;
```

This struct has three members.

• **max_vertex** is an int and is used to track the highest vertex in the graph

• **adj_matrix** itself is a static two-dimensional array that is used to implement adjacency matrix. Values are initialized to -1and 0 represents existing vertex and value represents an edge

• **visited** is a one-dimensional array that is used to keep track of path.

Functions Implemented

// Initialization Functions

Graph *graph_initialize();

- This function mallocs and initializes a new Graph structure, then returns a pointer to it
- Loops through all of the indexes to initialize them. After that the matrix represents a state with no vertices and no edges
- Returns the pointer to the Graph if successful or NULL on any errors.

// Vertex Operations

int graph_add_vertex(Graph *graph, int v1);

- This function adds vertex number v1 to the graph
- If it already exists, returns 0 and if v1 is not a legal vertex (ie. < 0 or >= MAX_VERTICES), returns error -1
- Returns 0 if successful

int graph_contains_vertex(Graph *graph, int v1);

- This function checks if the graph has vertex number v1 in the graph
- Returns 1 if it exists, on any errors or if v1 is not in the graph, returns 0.

int graph_remove_vertex(Graph *graph, int v1);

- This function removes vertex number v1 from the graph, if there are any edges connected to this vertex, they will also be removed
- If v1 is not in the graph, returns success
- If v1 is not a legal vertex (i.e. < 0 or >= MAX_VERTICES), returns error
- Returns 0 on successful remove, and returns -1 on any errors

// Edge Operations

int graph_add_edge(Graph *graph, int v1, int v2, int wt);

- This function adds an edge from v1 to v2 with weight of wt
- If this edge already exists, it updates to the new weight
- If either v1 or v2 are not in the graph, considers it an error and return -1
- Returns 0 on successful add

int graph_contains_edge(Graph *graph, int v1, int v2);

- This function checks if an edge from v1 to v2 exists
- Returns 1 if an edge exists and returns 0 in any other case.

int graph_remove_edge(Graph *graph, int v1, int v2);

- This function removes an edge from v1 to v2, if either v1 or v2 aren’t in the graph, considers it an error and return -1
- If v1 and v2 exist, but have no edges already, it’s a success and returns 0

// Graph Metrics Operations

int graph_num_vertices(Graph *graph);

- This function returns the number of vertices in the graph

int graph_num_edges(Graph *graph);

- This function returns the number of edges in the graph.

int graph_total_weight(Graph *graph);

- This function returns the sum of all edge weights in the graph

// Vertex Metrics Operations

int graph_get_degree(Graph *graph, int v1);

- This function returns the total degree of edges on Vertex V1, this includes both the in-degree and out-degree combined
- If there is a self-edge, that would count as both in and out, adding 2 to the degree and if there is no vertex V1, it is an error
- Returns the degree (number of edges) on v1 on success and on any errors, return -1.

int graph_get_edge_weight(Graph *graph, int v1, int v2);

- This function returns the weight of the edge from Vertex V1 to Vertex V2
- If there is no vertex V1 or vertex V2, it is an error
- Returns the weight of the v1 -> v2 edge on success and on any errors, return -1.

int graph_is_neighbor(Graph *graph, int v1, int v2);

- This function checks if v1 is connected to v2 (either direction)
- If there is no vertex V1 or vertex V2, it is an error
- Returns 1 if there is an edge from v1->v2 or an edge from v2->v1 and on any errors or if there is no such edge, returns 0

int *graph_get_predecessors(Graph *graph, int v1);

- This function returns all Predecessors of Vertex V1 as a dynamically allocated array and malloc's an array big enough for all predecessor vertices + 1
- The last entry of this array is -1 to indicate no further predecessors exist
- If there is no vertex V1, it is an error or returns pointer to the new array on success

int *graph_get_successors(Graph *graph, int v1);

- This function returns all Successors of Vertex V1 as a dynamically allocated array and mallocs an array big enough for all successor vertices + 1
- The last entry of this array is -1 to indicate no further successors exist
- If there is no vertex V1, it is an error.

// Graph Path Operations

int graph_has_path(Graph *graph, int v1, int v2);

- This function checks to see if there is path from v1 to v2, if there is no vertex V1 or vertex V2, it is an error
- Returns 1 if there is a path from v1 to v2 and on any errors or if there is no such path, returns 0

// Input/Output Operations

void graph_print(Graph *graph);

- This function when called, prints the graph
- The output is an adjacency list, as a graphical representation of the adj matrix

void graph_output_dot(Graph *graph, char *filename);

- This function outputs the graph in a format for the GraphViz program as a file
- The GraphViz (https://www.graphviz.org/) program (specifically dot) can be used to process this kind of a file
- Once finished, a file is opened with fopen, and closed with fclose.

int graph_load_file(Graph *graph, char *filename);

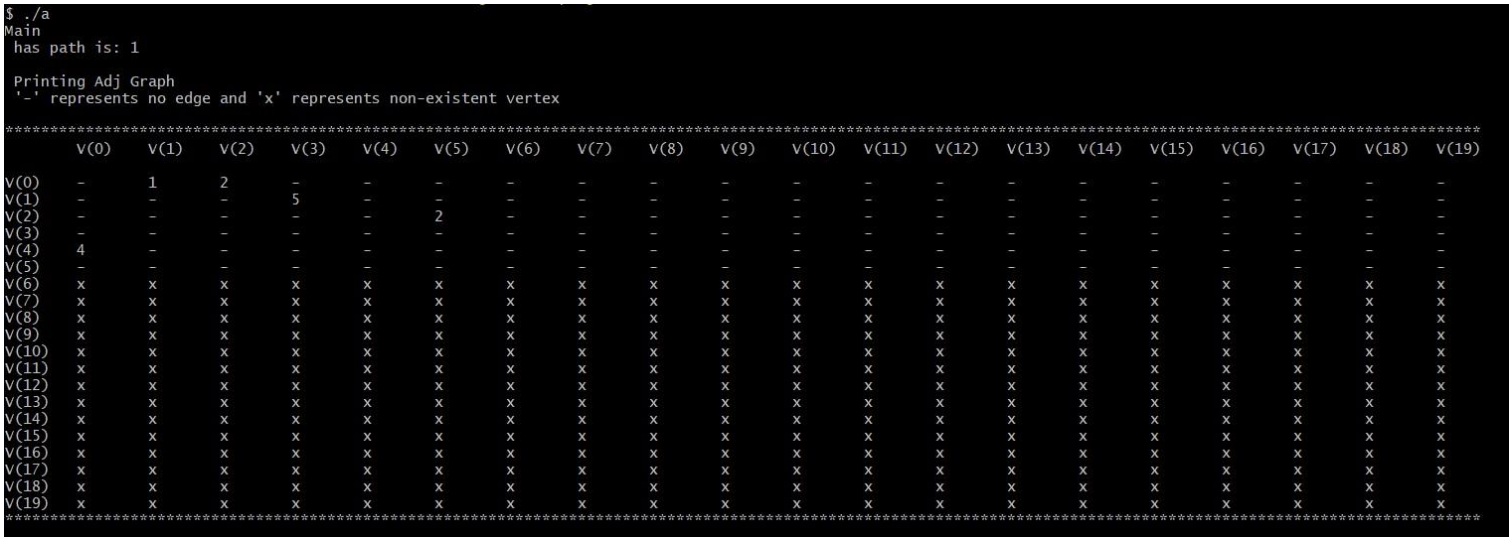
- This function loads a graph from a file
- The format for loading/saving files is as
    - to store a vertex with no edges, putting the vertex number on its own line
    - If there's an edge to store, we put it in this comma separated format, on its own line. Such as vertex1, vertex2, edge_weight

int graph_save_file(Graph *graph, char *filename);

- This function saves the graph to a file
- Uses the same file format as described in the graph_load_file function above.

OUTPUT

This following output is from a testing main file. (main.c)



```
$ ./a
Main
 has path is: 1

 Printing Adj Graph
 '-' represents no edge and 'x' represents non-existent vertex

**********************************************************************************************************************************************************************
       V(0)    V(1)    V(2)    V(3)    V(4)    V(5)    V(6)    V(7)    V(8)    V(9)    V(10)   V(11)   V(12)   V(13)   V(14)   V(15)   V(16)   V(17)   V(18)   V(19)

V(0)   -       1       2       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -
V(1)   -       -       -       5       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -
V(2)   -       -       -       -       -       2       -       -       -       -       -       -       -       -       -       -       -       -       -       -
V(3)   -       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -
V(4)   4       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -
V(5)   -       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -
V(6)   x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x
V(7)   x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x
V(8)   x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x
V(9)   x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x
V(10)  x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x
V(11)  x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x
V(12)  x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x
V(13)  x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x
V(14)  x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x
V(15)  x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x
V(16)  x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x
V(17)  x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x
V(18)  x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x
V(19)  x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x       x
**********************************************************************************************************************************************************************
```