Lab 11: Graph Algorithms

Michael Morikawa

May 26, 2020

Lab Questions

Question 1 Does Dijkstra's algorithm perform a DFS or BFS on a graph? Explain.
It is pretty close to doing a BFS on the graph, since it considers the vertices adjacent to whatever vertex it is at. However, the next vertex that is processed is not the next one on the same level or the first in the next level; what gets processed next is the vertex with the current shortest distance from the source vertex.

Question 2 Lab question 2: Explain the concept of "Transitive Closure" on a digraph. Provide a reason for performing a transitive closure on a digraph.
Transitive closure is the process of connecting vertices that have a path between them by adding a new edge into the graph that connects the two. A reason for performing transitive closure on a graph is whenever you are only concerned with whether or not you could reach a certain vertex from another. For example, this could be used to find if there are a series of one or more flights to get from A to B.

Source Code

main.cpp

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

typedef std::vector<std::vector<int>> Matrix;

class PQComp
{
public:
    bool operator()(std::pair<int, int> a, std::pair<int, int> b)
    {
        return a > b;
    }
};

Matrix transitiveClosure(Matrix g);
void shortestPath(Matrix g, int src, int dest);
void printPath(const std::vector<int> &parent, int dest);
void printMatrix(const Matrix &input);

int main()
```

```
{
    Matrix transitiveGraph = {{0, 0, 0, 1, 1},
                              {1, 0, 1, 0, 0},
                              {0, 0, 0, 1, 0},
                              {0, 0, 0, 0, 1},
                              {0, 0, 1, 0, 0}};

    Matrix dijkstraGraph = {{0, 0, 0, 5, 10},
                            {3, 0, 4, 0, 0},
                            {0, 0, 0, 2, 0},
                            {0, 0, 0, 0, 3},
                            {0, 0, 6, 0, 0}};

    std::cout << "Starter matrix/graph: \n";
    printMatrix(transitiveGraph);
    std::cout << "\nUpdated matrix/graph: \n";
    Matrix update = transitiveClosure(transitiveGraph);
    printMatrix(update);
    std::cout << '\n';
    shortestPath(dijkstraGraph, 1, 4);
}

Matrix transitiveClosure(Matrix g)
{
    Matrix output = g;
    for (int k = 0; k < g.size(); k++)
    {
        for (int i = 0; i < g.size(); i++)
        {
            for (int j = 0; j < g.size(); j++)
            {
                if (g[i][k] == 1 && g[k][j] == 1)
                {
                    output[i][j] = 1;
                }
            }
        }
    }
    return output;
}

void printMatrix(const Matrix &input)
{
    for (auto row : input)
    {
        std::cout << "[";
        for (auto elem : row)
        {
            std::cout << elem << "  ";
        }
        std::cout << "]\n";
    }
}
```

```
void shortestPath(Matrix g, int src, int dest)
{
    typedef std::pair<int, int> int_pair;
    std::vector<std::pair<int, int>> distance;
    std::vector<int> parent(g.size());
    parent[src] = -1;
    for (int i = 0; i < g.size(); i++)
    {
        distance.push_back(std::make_pair(INT_MAX, i));
    }
    distance[src].first = 0;
    std::priority_queue<int_pair, std::vector<int_pair>, std::greater<int_pair>> Q;
    Q.push(distance[src]);
    int u;
    std::vector<bool> isInQueue(g.size(), true);
    while (!Q.empty())
    {
        u = Q.top().second;
        isInQueue[u] = false;
        if (u == dest)
        {
            break;
        }
        Q.pop();
        for (int z = 0; z < g.size(); z++)
        {
            if (g[u][z] != 0 && isInQueue[z])
            {
                if (distance[u].first + g[u][z] < distance[z].first)
                {
                    distance[z].first = distance[u].first + g[u][z];
                    Q.push(distance[z]);
                    parent[z] = u;
                }
            }
        }
    }
    std::cout << "Distance from vertex " << src
              << " to vertex " << dest << ": "
              << distance[dest].first
              << "\nExtra Credit: \n";
    printPath(parent, dest);
}

void printPath(const std::vector<int> &parent, int dest)
{
    int i = dest;
    std::stack<int> path;
    path.push(dest);

    while (parent[i] != -1)
    {
        path.push(parent[i]);
        i = parent[i];
```

```
    }

    std::cout << "Path taken: ";

    while (!path.empty())
    {
        std::cout << path.top() << " ";
        path.pop();
    }
    std::cout << '\n';
}
```

Output