Lab08-Graph Exploration

CS214-Algorithm and Complexity, Xiaofeng Gao & Lei Wang, Spring 2021.

∗ If there is any problem, please contact TA Yihao Xie.

∗ Name: Zilong Li   Student ID: 518070910095   Email: logcreative-lzl@sjtu.edu.cn

1. Given an undirected graph $G = (V, E)$. Prove the following propositions.

   (a) Let e be a maximum-weight edge on some cycle of connected graph $G = (V, E)$. Then there is a minimum spanning tree of G that does not include e. Moreover, there is no minimum spanning tree of G that includes e if e is the unique maximum-weight edge on the cycle.

   (b) Let T and T' are two different minimum spanning trees of G. Then T' can be obtained from T by repeatly substitute one edge in $T \backslash T'$ by one edge in $T' \backslash T$ and meanwhile the result after each subsitution is still a minimum spanning tree.

2. Let $G = (V, E)$ be a connected, undirected graph. Give an $O(|V| + |E|)$-time algorithm to compute a path in G that traverses each edge in E exactly once in each direction. Describe how you can find your way out of a maze if you are given enough coins to apply your algorighm.

3. Consider the maze shown in Figure 1. The black blocks in the figure are blocks that can not be passed through. Suppose the block are explored in the order of right, down, left and up. That is, to go to the next block from (X, Y), we always explore $(X, Y + 1)$ first, and then $(X + 1, Y)$,$(X, Y - 1)$ and$(X - 1, Y)$ at last. Answer the following subquestions:

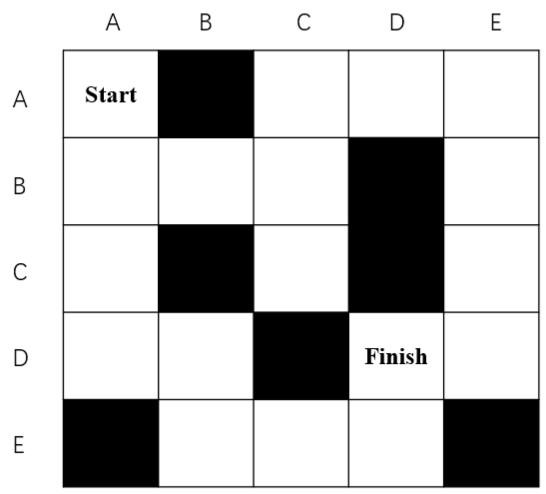


Figure 1: The blocks in the maze.

   (a) Give the sequence of the blocks explored by using DFS to find a path from the "start" to the "finish".

   Solution. The DFS order is:

   AA, AB, BB, CB, CC, CA, DA, EA, EB, EC, ED, DD

   which is shown in Figure 2.

And the path found is:

```
AA, AB, BB, CB, CA, DA, EA, EB, EC, ED, DD
```

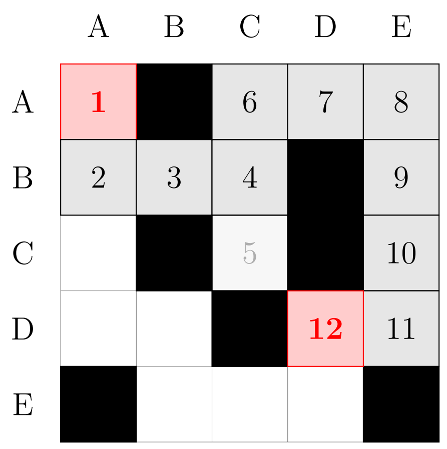


Figure 2: DFS Explored Sequence

□

(b) Give the sequence of the blocks explored by using BFS to find the <u>shortest</u> path from the “start” to the “finish”.

Solution. The BFS order is:

```
AA, AB, BB, AC, CB, AD, CC, CA, BD, DA, BE, EA, CE, EB, DE, EC, DD
```

which is shown in Figure 3.
And the shortest path is:

```
AA, AB, AC, AD, BD, BE, CE, DE, DD
```

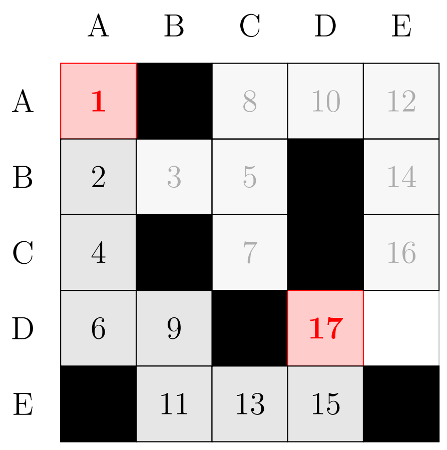


Figure 3: BFS Explored Sequence

□

(c) Consider a maze with a larger size. Discuss which of BFS and DFS will be used to find one path and which will be used to find the shortest path from the start block to the finish block.

Solution. DFS will be used to find one path. DFS could reach every connected node, which is proved in the lecture. Then, DFS will certainly find a path that reaches the **Finish** from **Start**.

BFS will be used to find the shortest path. When EJECT happens, BFS will INJECT the node unvisited and adjacent to the ejected node with 1 increment on the distance.

By mathematical induction, the node with distance d will be ejected and the following nodes with distance $d + 1$ will be injected, with the basic step starting from $d = 0$ where there is only the **Start** in the queue. Figure 4 shows the process.

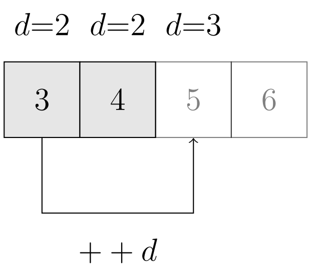


Figure 4: INJECT

Then nodes are ordered by the distance to INJECT to the queue and EJECT from the queue. Then before **Finish** is INJECT to the queue, there is no shorter path to **Finish**. And after **Finsh** is EJECT from the queue, the path is always farther than the path when injected.

Thus, BFS will find the shortest path from **Start** to **Finish**.

□

4. Given a directed graph G, whose vertices and edges information are introduced in data file "SCC.in". Please find its number of Strongly Connected Components with respect to the following subquestions.

(a) Read the code and explanations of the provided C/C++ source code "SCC.cpp", and try to complete this implementation.

Solution. The count of Strong Connected Components is

```
202
```

□

(b) Visualize the above selected Strongly Connected Components for this graph G. Use the *Gephi* or other software you preferred to draw the graph. (If you feel that the data provided in "SCC.in" is not beautiful, you can also generate your own data with more vertices and edges than G and draw an additional graph. Notice that results of your visualization will be taken into the consideration of Best Lab.)

Solution. The visualization using Gephi is shown in Figure 5.

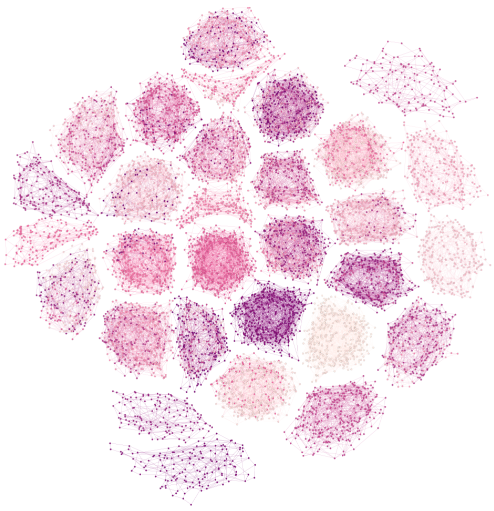
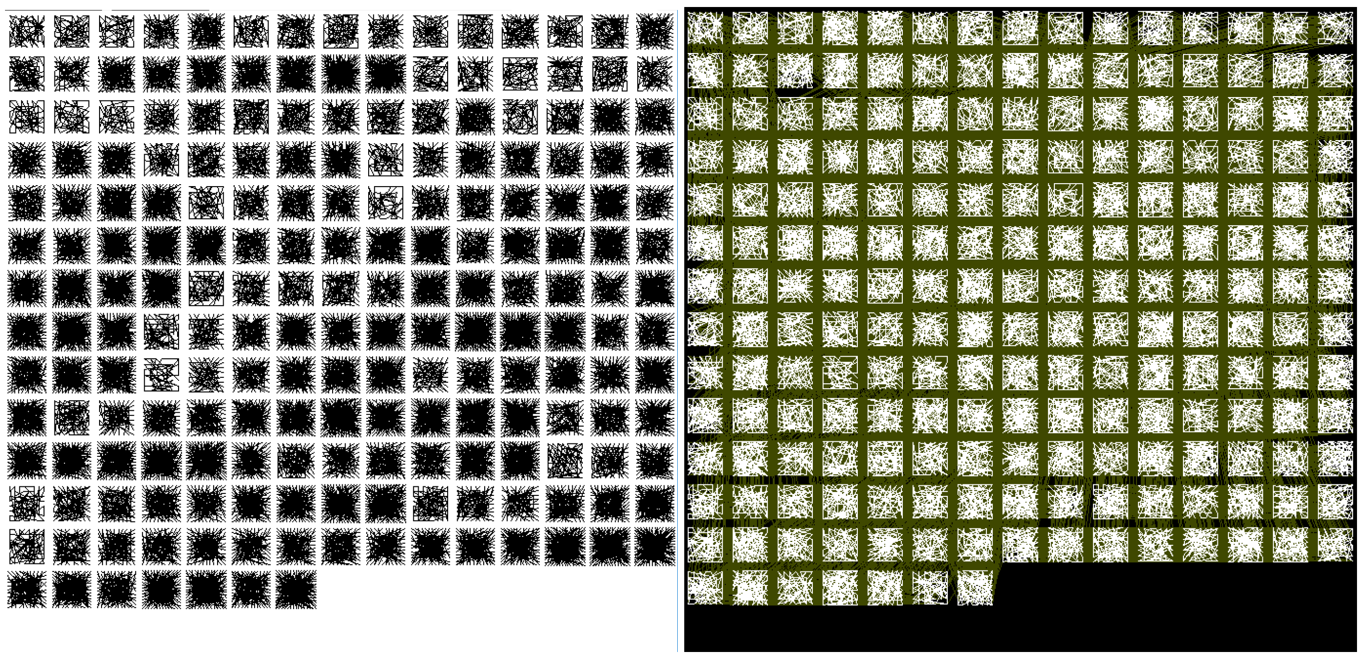


Figure 5: Using Gephi

Additionally, the project that I completed in the Principles and Practice of Problem Solving course is also used to visualize the graph. GraphGenDecomp program visualize the Strong Connected Components (SCC) and all edges in Figure 6. You can get more information about this program on GraphGenDecomp on GitHub. Basically, it uses spirial funciton to organize nodes into a centralized form for every sub-graph.


(a) SCC only

(b) with crossing edges

Figure 6: Using GraphGenDecomp

□

A Code for Problem 4

`SCC.cpp` is the main code for counting and getting the strong connected components.

```cpp
#include <vector>
#include <iostream>
#include <fstream>
using namespace std;
//Please put this source code in the same directory with SCC.in
//And do NOT change the file name.
/*
This function computes the number of Strongly Connected Components in a graph
Args:
    n: The number of nodes in the graph. The nodes are indexed as 0~n-1
    edge: The edges in the graph. For each element (a,b) in edge, it means
          there is a directed edge from a to b
          Notice that there may exists multiple edge and self-loop
Return:
    The number of strongly connected components in the graph.
*/

class Digraph {
public:
    Digraph(int _n) {
        n = _n;
        G = vector<vector<int>>(n);
    }
    void addEdge(pair<int, int> e) {
        G[e.first].push_back(e.second);
    }
    Digraph reverse() {
        Digraph GR(n);
        for (int i = 0; i < n; ++i)
            for (int end : G[i])
                GR.addEdge(make_pair(end, i));
        return GR;
    }
    vector<int> dfs() {
        visited = vector<bool>(n, false);
        visiting = vector<int>();
        for (int i = 0; i < n; ++i)
            if(!visited[i])
                dfs(i);
        return visiting;
    }
    int SCC(vector<int> order) {
        int count = 0;
        visited = vector<bool>(n, false);
        ofstream fscc;
        fscc.open("partition.txt");
        for (int node : order) {
            if (!visited[node]) {
                visiting = vector<int>();
                dfs(node);
                ++count;
                for (int v : visiting)
                    fscc << v << ' ';
                fscc << endl;
            }
        }
        fscc.close();
        return count;
    }
private:
    int n;
    vector<vector<int>> G;
    vector<bool> visited;
    vector<int> visiting;
    void dfs(int node) {
        visited[node] = true;
        for (int adj : G[node])
            if(!visited[adj])
                dfs(adj);
        visiting.push_back(node);
    }
};

int SCC(int n, vector<pair<int,int>>& edge) {
    Digraph G(n);
    for (auto e : edge)
        G.addEdge(e);
    Digraph GR = G.reverse();
    vector<int> visiting = GR.dfs();
    reverse(visiting.begin(), visiting.end());
    return G.SCC(visiting);
}
//Please do NOT modify anything in main(). Thanks!
int main()
{
    int m,n;
    vector<pair<int,int>> edge;
    ifstream fin;
    ofstream fout;
    fin.open("SCC.in");
    cout<<fin.is_open()<<endl;
    fin>>n>>m;
    cout<<n<<" "<<m<<endl;
    int tmp1,tmp2;
    for(int i=0;i<m;i++)
    {
        fin>>tmp1>>tmp2;
        edge.push_back(pair<int,int>(tmp1,tmp2));
    }
    fin.close();
    int ans=SCC(n,edge);
    fout.open("SCC.out");
    fout<<ans<<'\n';
    fout.close();
    return 0;
}
```

`datapre.py` is the data preprocessing code to adapt Gephi and GraphGenDecomp.

```python
import pandas as pd
import os
indata = pd.read_table(os.path.dirname(__file__) + "/SCC.in",sep=' ')
adapter=""
for l in indata.index.to_list():
    adapter += '<' + str(indata[indata.columns[0]][l]) + ',' + str(indata[indata.columns[1]][l]) + ",1>\n"
with open("main.txt",'w') as file_object:
    file_object.write(adapter)
edges="source,target\n"
for l in indata.index.to_list():
    edges += str(indata[indata.columns[0]][l]) + ',' + str(indata[indata.columns[1]][l]) + "\n"
with open("edges.csv",'w') as file_object:
    file_object.write(edges)
nodes="id,label\n"
for i in range(0,int(indata.columns[0])):
    nodes+=str(i)+','+str(i)+'\n'
with open("nodes.csv",'w') as file_object:
    file_object.write(nodes)
```

Remark: Please include your .pdf, .tex, .cpp files for uploading with standard file names.