Lab08-Graph Exploration

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

∗ If there is any problem, please contact TA Yihao Xie.
∗ Name:Zirui Liu Student ID: 519021910343 Email: L.prime@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.

   Solution. (a):
   We suppose that T is a spanning tree of minimum weight, and there exists a cycle C so that e is the heaviest edge in C and $e \in E(T)$. Let e connects two points u, v and let $P = C-e$, so that p is also a path from u to v. Now $T-e$ are consists of two parts, as T_1 and T_2 while one contains u and the other one contains v. Since P is a path from u to v, there exists an edge $f \in E(P)$ with one end in T_1 and the other one in T_2. By assumption we have $w(f) < w(e)$, now $T-e+f$ is a spanning tree with weight less than T, which made a contradiction. So the original problem can be solved. For the next question, if e is the unique maximum-weight edge on the cycle, there would always be another path having less weight that can replace e, so in contradiction, there is no minimum spanning tree of G that includes e if e is the unique maximum-weight edge on the cycle.

   (b):
   We approve by contradiction. If this is not true, assume after one replacement, we got a new T_i that is not a minimum spanning tree, assume that we just use x_i to replace x, then we can use x_i to replace x at first, for original T, so we can get a better structure for T. The T we modified uses less weight to connect all points, lesser then T. So T is not a minimum spanning tree, contradiction. In conclusion, 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.

   Solution. 1. We first perform a DFS for G, starting at an arbitrary vertex.(The path required by the problem can be obtained in the order which DFS explores in the graph.)
   2. When exploring an edge (u, v) that goes to an unvisited vertex, the edge (u, v) is included for the first time in the path.
   3. When DFS backtracks to u again after v is made BLACK, the edge (u, v) (which in more precise way is that the edge connecting u and v not considering the direction) is included for the second time in the path, which this time is in the opposite direction. from v to u.
   4. When DFS explores an edge (u, v) that goes to a pre-visited node (GRAY or BLACK), we add (u, v) and (v, u) to the path. In this way we can make sure that each edge is added to

the path exactly twice.

□

3. Consider the maze shown in Figure **??**. 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:

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

   (b) Give the sequence of the blocks explored by using BFS to find the shortest path from the "start" to the "finish".

   (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.

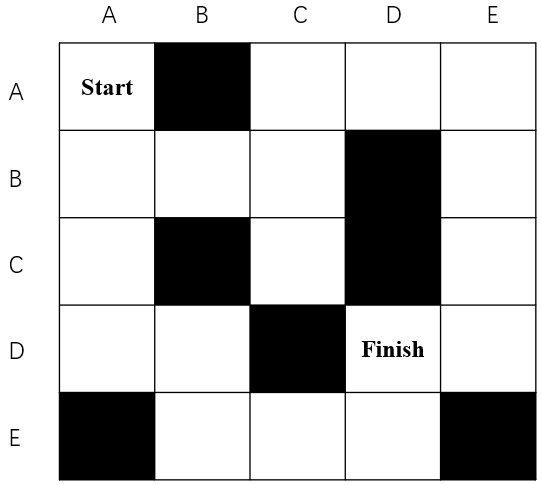

图 1: The blocks in the maze.

Solution. (a):
We give the sequence by (x, y).

(A, A)
-> (B, A)
-> (B, B)
-> (B, C)
-> (C, C)
-> (B, C)
-> (A, C)
-> (A, D)
-> (A, E)
-> (B, E)
-> (C, E)
-> (D, E)

-> (D, D)

(b):

(A, A)
(B, A)
(B, B)
(C, A)
(B, C)
(D, A)
(C, C)
(D, B)
(A, C)
(E, B)
(A, D)
(E, C)
(A, E)
(E, D)
(B, E)
(D, D)

(c):
DFS will be used to find one path and BFS will be used to find the shortest path from the start block to the finish block.It's because DFS can get straight to the destination, DFS with backtrack CAN always find a way if there is a way. Otherwise the way does not exist. But DFS can't not get all the different ways od path, so it cannot know which way is the shortest path. But BFS is different, BFS can get through all the ways possible to get to the end, so BFS can get the shortest way. But if we only need to find if there is a way from the start to the end, we would only have to apply DFS.

□

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.

   (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.)

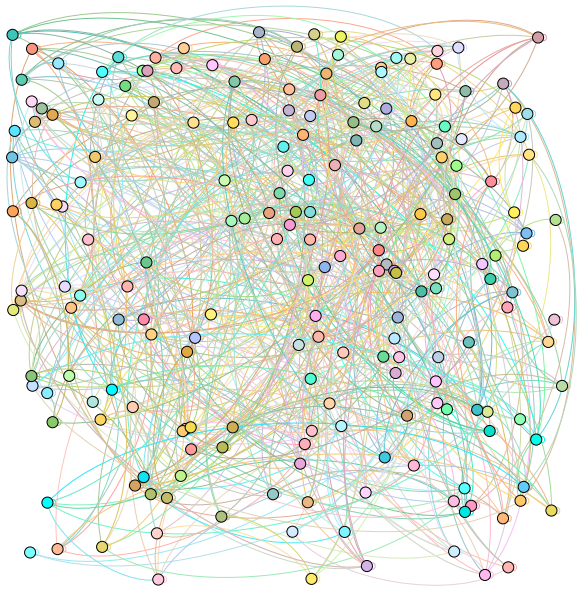

图 2: picture

Solution. □

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

1 Second appendix: SCC.cpp

Input c++ source1:

```
#include <vector>
#include <iostream>
#include <fstream>
#include <stack>
using namespace std;
const int maxn = 1e5;
//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.
*/
bool visit[maxn];
stack<int> stack1;
vector<int> Graph[maxn];
int DFN[maxn], Low[maxn], connect[maxn];
int index_ = 0, num = 0;
void calculate(int u)
{
    Low[u] = DFN[u] = ++index_;
    visit[u] = true;
    stack1.push(u);
    for (int v : Graph[u])
    {
        if (DFN[v] == 0)
        {
            calculate(v);
            Low[u] = min(Low[u], Low[v]);
        }
        else if (visit[v])
        {
            Low[u] = min(Low[u], DFN[v]);
        }

    }
    if (Low[u] == DFN[u])
    {
        num++;
        while (visit[u])
        {
            int v = stack1.top();
```

```
            stack1.pop();
            visit[v] = false;
            connect[v] = num;
        }
    }
}
int SCC(int n, const vector<pair<int, int>> &edge)
{
    for (auto iterator = edge.cbegin(); iterator != edge.cend(); iterator++)
    {
        Graph[iterator->first].push_back(iterator->second);
    }
    for (int u = 0; u < n; u++)
    {
        if (connect[u] == 0)
        {
            calculate(u);
        }
    }
    return num;
}
//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);
    cout<<"Test Answer: "<<ans<<endl;
    fout.open("SCC.out");
    fout << ans << '\n';
    fout.close();
    return 0;
}
```