Lab08-Graph Exploration

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

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

∗ Name: Wendi Chen Student ID: 519021910071 Email: chenwendi-andy@sjtu.edu.cn

1. Given a 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) Set m as an MST of G. If e is not contained in m, then we have proved the conclusion. Otherwise, we can remove e from m and m will be split into two small trees m_1 and m_2. Because e is on some cycle of G, the cycle must connects m_1 and m_2. Thus, there exists another edge $e^{'}$ that connect m_1 and m_2, which constructs a MST that does not include e (Figure 1). Moreover, if e is the unique maximum-weight edge on the cycle, then $|e^{'}| < |e|$, which means that m can not be a MST. Therefore, there is no minimum spanning tree of G that includes e.

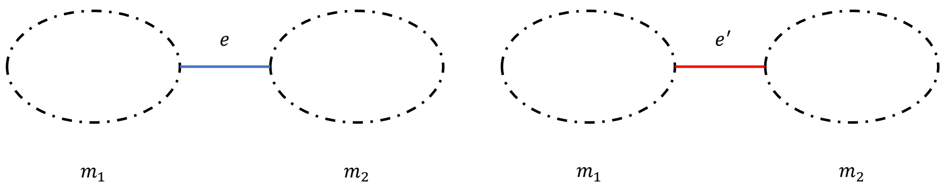

Figure 1: How to Find an Edge for Substitution

(b) What we need to consider is the edges on some cycle of G which are contained in $T^{'}$ but not in $T(T_1)$. We denote them by $e_1, e_2, \ldots, e_n$. Then, let us prove by induction.
We start with $i = 1$. In fact, T_1 is a MST.
We assume $T_i (i \geq 1)$ is a MST. For e_i, we add them to T_i and T_i becomes a graph $T_i^{'}$. Then, e_i will construct a circle in $T_i^{'}$. Because T_i is a MST, for every $e_j \neq e_i$ on the circle, we have $|e_j| \leq |e_i|$. According to problem (a), there must exist $e_k \neq e_i$ so that $|e_k| = |e_i|$. Then, we'll prove that there must exist $e_q \in T_i \backslash T^{'}$ satisfying this condition. If all the edges on the circle have the same weight, it's trivial. Otherwise, if all $e_k \neq e_i$ with $|e_k| = |e_i|$ on the circle is in $T^{'}$, then we can replace one e_k in $T^{'}$ with an edge with less weight, which contradicts the fact that $T^{'}$ is a MST. Thus, we can remove $e_q \in T_i \backslash T^{'}$ from $T_i^{'}$ and get T_{i+1}, which is also a MST.
By doing this recursively, we'll finally get $T^{'}$, and $T_1(T), T_2, \ldots, T_{n+1}(T^{'})$ are all MSTs.

□

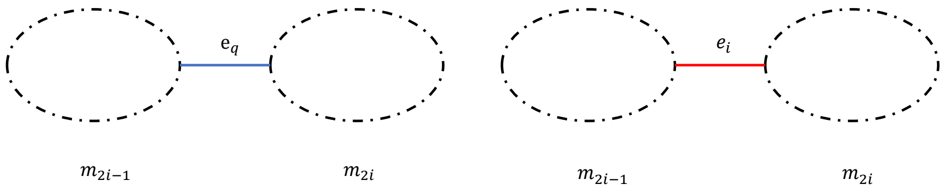

Figure 2: How to Transform One MST to Another

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 a large supply of pennies.

   Solution.

```
Algorithm 1: Traverse(G, v)
Input: $G = (V, E)$ is a connected, undirected graph;$v \in V$
Output: Sequence (not a set) $P$ which represents the path
$f \leftarrow NULL$;
$visited(v) = true$;
for $e = edge(v, u) \in E$ do
    if $count(e) = 0$ then
        $P \leftarrow P \cup \{(v, u)\}$;
        $count(e) \leftarrow count(e) + 1$;
        if $visited(u) = false$ then
            Traverse$(G, u)$;
        else
            $count(e) \leftarrow count(e) + 1$;
            $P \leftarrow P \cup \{(u, v)\}$;
    else if $count(e) = 1$ then
        $f \leftarrow e$;
if $f$ is not $NULL$ then
    $count(f) \leftarrow count(f) + 1$;
    $P \leftarrow P \cup \{f^{-1} = (u, v)\}$;
```

   In this algorithm, every edge will be examined exactly twice and every vertex will be marked. Thus, the time complexity is $O(|V| + |E|)$.
   When we try to find a way out of a maze, we can put some pennies on each edge to represent *count*(e) and then we can traverses each edge in the maze exactly once in each direction to find the route.

   □

3. Consider the maze shown in Figure 3. 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.

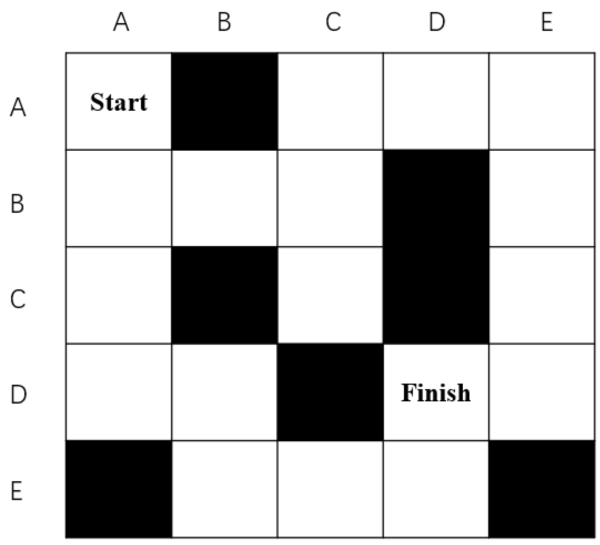

Figure 3: An example of making room for one new element in the set of arrays.

Solution.

(a) The sequence is (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) The sequence is (A, A),(B, A),(B, B),(C, A),(B, C),(D, A),(C, C),(A, C),(D, B),(A, D), (E, B),(A, E),(E, C),(B, E),(E, D),(C, E),(D, D).

(c) If we just want to find one path, we had better use DFS. The reason is that in DFS, we'll explore the block that far from the start block first and we try every route one by one instead of doing this parallelly. Thus, we can explore fewer blocks to get the path. If we want to find the shortest path, we had better use BFS. The reason is that in BFS, every route we try parallelly has the same number of blocks. Then, the route arrives at the finish block first is the shortest path. If we use DFS instead, we may have to find every possible path one by one and do a comparison, which costs far more than BFS.

$\square$

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.)

Solution.

(a) I use *Tarjan Algorithm* to find the number of strongly connected components. Please refer to *SCC.cpp* for details.

(b) First, we dye the vertexes in different strongly connected components with different colors and visualize the graph (Figure 4).

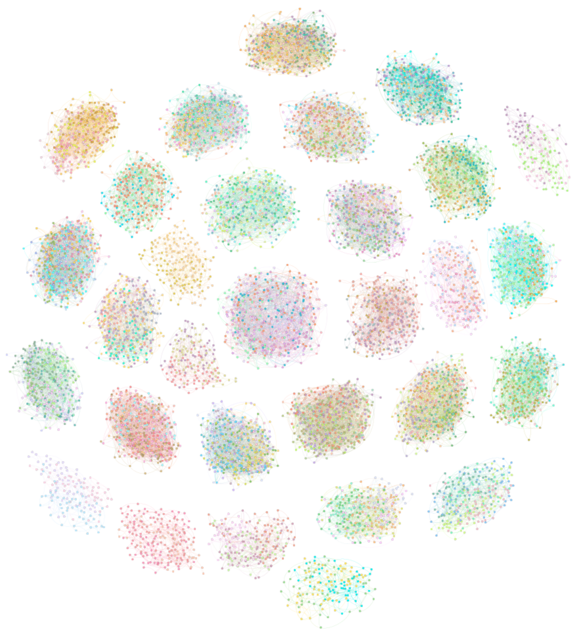

Figure 4: Visualize the Graph with Gephi

Then, I merge all the vertexes in one strongly connected components into a "super node" whose size implies the number of vertexes it contains. The result is much more beautiful (Figure 5). From the result, we can find there are 20 weakly connected components and 202 strongly connected components.

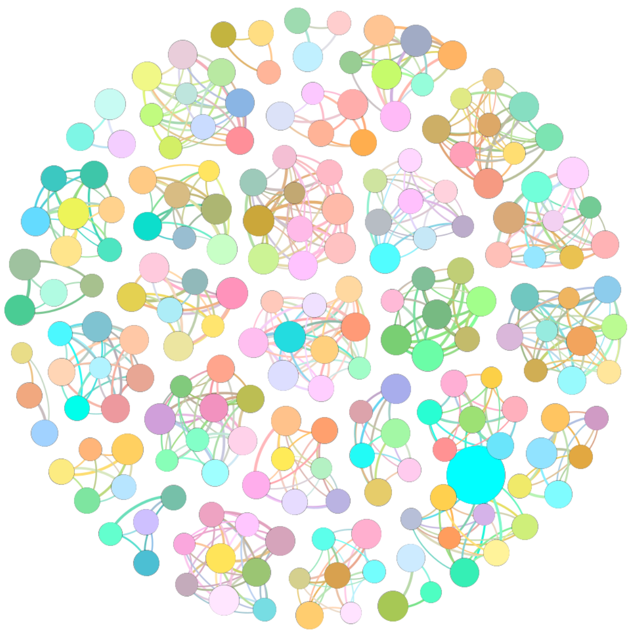
Figure 5: Visualize SCC with Gephi

□

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