Lab08-Graph Exploration

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

∗ If there is any problem, please contact TA Yihao Xie.
∗ Name:Beichen Yu Student ID:519030910245 Email: polarisybc@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) It is obvious that in a circle has n edges, in the spanning tree we must delete at least one edge to confirm that there is only one path between any two vertices. So in that circle, we must choose at least one edge to delete. Because e is the maximum-weight edge, to get a minimum spanning tree, there is no better way than delete e, so begin with deleting e we can get a minimum spanning tree without e. Moreover, if e is the unique maximum-weight edge on the cycle, deleting e is the only optimal solution to get a minimum spanning tree, so there is no minimum spanning tree that includes e.

   (b) Assume that there are n edges in the minimal spanning tree. Denote $w(e)$ as the weight of edge e. Assume $a_1, a_2, \cdots, a_n$ are edges of T, and $b_1, b_2, \cdots, b_n$ are edges of T'. And assume that $w(a_1) \leqslant w(a_2) \leqslant \cdots \leqslant w(a_n); w(b_1) \leqslant w(b_2) \leqslant \cdots \leqslant w(b_n)$.

   Assume i is the first time that $w(a_i) \neq w(b_i)$. And assume that $w(a_i) \geqslant w(b_i)$.

   In the first case, $b_i \in T$. Because $\forall t < i : w(a_t) = w(b_t)$, we can confirm that $\exists j > i$ and $w(a_j) = w(b_i)$. So $w(b_i) = w(a_j) \geqslant w(a_i) \geqslant w(b_i)$. That means $w(a_i) = w(b_i)$.

   In the second case, $b_i \notin T$. If we add b_i into T, it will form a circle somewhere. Obviously, there is an edge $a_j \notin T'$ and $j \geqslant i$. Because T is a minimal spanning tree, $w(b_i)$ is bigger or equal to the weight of any edge in that circle. So we have $a_j \leqslant b_i$. So $w(b_i) \leqslant w(a_i) \leqslant w(a_j) \leqslant w(b_i)$. That means $w(a_i) = w(b_i)$.

   So we confirm that $w(a_1) = w(b_1), w(a_2) = w(b_2), \cdots, w(a_n) = w(b_n)$. That means T' can be obtainedby recursively substitute one edge in T by one edge in T'.

   □

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. We use an algorithm similar to the DFS.

   We need a stack to store some information.

   At the beginning, all of the edges are not signed.

   First, we choose a vertex to begin. We randomly choose an edge connected with the vertex, sign it with the out direction and push the edge into the stack.

Then we go to the next vertex. We do the **going-out operation**: just choose a never signed edge connected with the vertex, sign it with the out direction and push the edge into the stack.

Repeat this operation until finding a vertex that all edges connected with it are signed.Then we do the **going-back operation**: we should pop the stack, and go back along the edge poped from the stack. Sign the edge with the back direction. Then we go back to the last vertex.

If we can find a never signed edge connected with that vertex, repeat the **going-out operation**. Otherwise, repeat the **going-back operation**. Finally, we will go back to the origin vertex, and we find a path in G that traverses each edge in E exactly once in each direction. □

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:

   (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.

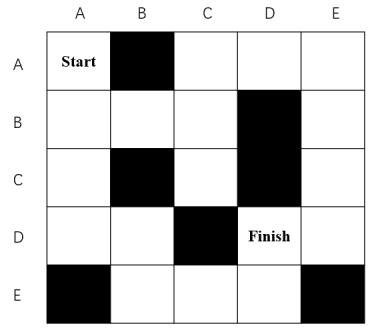

Figure 1: An example of making room for one new element in the set of arrays.

Solution. (a) $(A, A) \to (A, B) \to (A, C) \to (A, D) \to (B, D) \to (B, E) \to (C, E) \to (D, E) \to (D, D)$.

(b) $(A, A) \to (A, B) \to (A, C) \to (B, B) \to (A, D) \to (C, B) \to (B, D) \to (C, C) \to (C, A) \to (B, E) \to (D, A) \to (C, E) \to (E, A) \to (D, E) \to (E, B) \to (D, D)$

The shortest path from the "start" to the "finish" is the same as the result of DFS algorithm: $(A, A) \to (A, B) \to (A, C) \to (A, D) \to (B, D) \to (B, E) \to (C, E) \to (D, E) \to (D, D)$.

(c) If the maze is larger, BFS is a better algorithm. Using DFS, we can search only one path once. If we can't find the "finish", we have go back to the original position. Besides, if using DFS, the number of recursion levels will be very deep. If using BFS, we can approach the "finish" step by step. The larger the maze is, the more efficient BFS is.

□

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) The .cpp code is included in the .zip file. And the answer is 202.

(b) Replace each strongly connected component with a vertex, and use the size and color of the vertex to represent the number of vertices in the original strongly connected component. Then use *Gephi* to draw this figure.

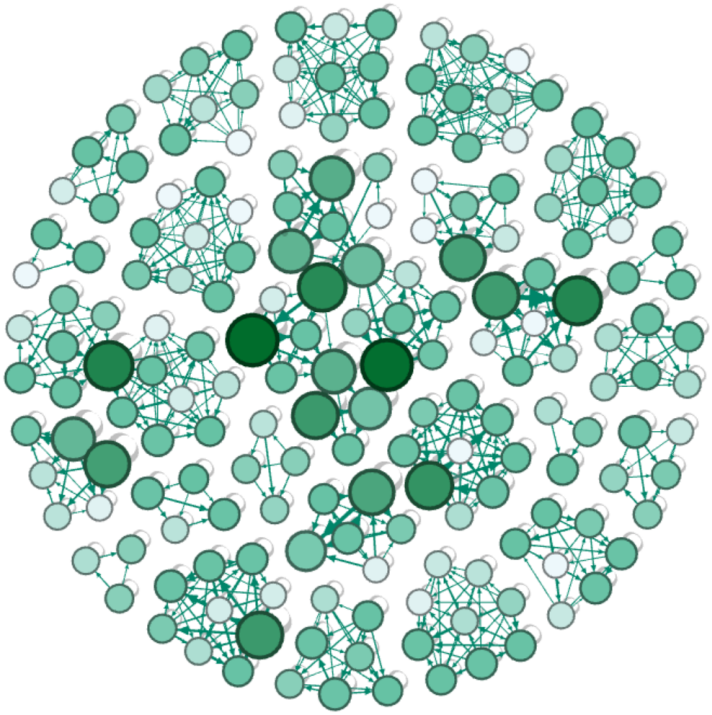

Figure 2: The Strongly Connected Components

□

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