IMPLEMENTATION OF KRUSKAL’S ALGORITHM

XI CHEN

1. Project Summary

Implement Kruskal’s algorithm for minimal spanning tree on computer using your favorite programming language (C/C++, Java, Python, Visual Basic, ...). This project requires the understanding of the relevant concepts in graph theory and some programming skills.

2. Kruskal’s Algorithm

We only consider simple graphs, i.e., without loops and multiple edges.

Given a simple connected weighted graph $G$ of vertices $V$, edges $E$ and weights $w : E \rightarrow \mathbb{R}$, we may use Kruskal’s algorithm to find a spanning tree of minimum weight:

Step 1. We let $T$ be the subgraph containing all vertices $V$ and no edges and we sort the edges in order from the smallest weight to the largest weight:

$$e_1, e_2, \ldots, e_m.$$

We let $k = 1$.

Step 2. If $T + e_k$ does not contain a cycle, we replace $T$ by $T + e_k$.

Step 3. Replace $k$ by $k + 1$. If $k > m$, we are done and $T$ is a minimal spanning tree; otherwise, go to Step 2.

For example, let us use Kruskal’s algorithm to find a minimal spanning tree of the weighted graph

```
      B
     /|
    / |
   /  |
 A--\-C
    |  |
    |  |
    F--D
```

First, we list the edges in the order of increasing weights:

$$AB, BD, BC, CD, CE, AF, CF, EF, BF, BE.$$ 

Then the algorithm gives us (step by step):

Date: 2016.07.20.
3. Suggestions on Implementation

Usually, a simple weighted graph $G$ with $n$ vertices (labeled from 1 to $n$) is represented by an $n \times n$ matrix, i.e., a two dimensional array $g_{ij}$ in many computer languages. We set $g_{ij}$ to 0 if $i$ and $j$ are not adjacent and to the weight of the edge $ij$ if they are (assuming that all weights are positive). So in the example above, the matrix representing the graph is

$$G = \begin{bmatrix}
0 & 1 & 0 & 0 & 0 & 6 \\
1 & 0 & 3 & 0 & 10 & 9 \\
0 & 3 & 0 & 4 & 5 & 7 \\
0 & 0 & 4 & 0 & 2 & 0 \\
0 & 10 & 5 & 2 & 0 & 8 \\
6 & 9 & 7 & 0 & 8 & 0
\end{bmatrix}$$
with nodes A-F labeled by 1-6, respectively. And the matrix representing the minimal spanning tree is

\[
T = \begin{bmatrix}
0 & 1 & 0 & 0 & 0 & 6 \\
1 & 0 & 3 & 0 & 0 & 0 \\
0 & 3 & 0 & 4 & 0 & 0 \\
0 & 0 & 4 & 0 & 2 & 0 \\
0 & 0 & 0 & 2 & 0 & 0 \\
6 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

Your program, if correctly implemented, should spit out the matrix (3.2), when fed the matrix (3.1).

The tricky part is checking whether \( T + e_k \) contains a cycle. For that purpose, I suggest the introduction of an \( n \times n \) matrix \( A \), which tracks the number of connected components of \( T \) at any time.

The matrix \( A = \{a_{ij}\} \) is given by \( a_{ij} = 1 \) if there is a path from \( i \) to \( j \) in \( T \) and \( a_{ij} = 0 \) otherwise. At the beginning, \( T \) is the zero matrix and

\[
A = \begin{bmatrix}
1 & 1 & 1 & \cdots & 1 \\
1 & \ddots & & & \\
& & & & \\
& & & & \\
& & & & \\
\end{bmatrix}
\]

When we try to add an edge \( e_k = ij \) to \( T \), we check the value \( a_{ij} \). If \( a_{ij} = 1 \), it means that \( T + e_k \) will contain a cycle and \( e_k \) cannot be added. Otherwise, we replace \( T \) by \( T + e_k \) and update \( A \) accordingly.

In the end, \( A \) should be a matrix with all entries 1. If it is not, it means that the graph \( G \) we start with is not connected.