FIT3155: Lab questions for week 5

Objectives: (1) Continue (if remaining) Ukkonen's suffix tree construction from last week. (2) Disjoint set data structure with smart union algorithms and find with path compression.

1. Continue the implementation of Ukkonen's algorithm. See below the suggested sequence of steps so that you can progressively work towards a full working implementation, while sanity-checking each step.
   (a) Implement Ukkonen's algorithm for suffix tree construction without any optimizations/speed-up/tricks. (Refer slide 16 of the lecture slides.)
   (b) Extend the above implementation by computing suffix links during each phase. (Refer slide 22.)
   (c) Add further, the ability to traverse via the suffix links during suffix extensions in any given phase. (Refer slides 23-26.)
   (d) Enhance this implementation using skip/count trick. (Refer slide 27.)
   (e) If not already done, extend further your implementation using the space-efficient representation of edge-labels. (Refer slide 28.)
   (f) Add to this, the premature stopping criterion. (Refer slide 29.)
   (g) Improve this further to handle rapid leaf extensions. (Refer slides 30-34.)
   (h) Finally, generate the final suffix tree of the input string (by extending the implicit tree one more time by '$' character). (Refer slide 35.)
2. Implement a disjoint set data structure supporting:
   (a) Union-by-size without path compression.
   (b) Union-by-height/rank without path compression.
   (c) Union-by-height with path compression.
3. Reimplement Kruskal's Minimum-Weight Spanning Tree algorithm (you learnt in FIT2004) using union-by-height with path compression. As a part of this exercise, you will have to also write a routine to construct a random weighted undirected graph – think about ways to do this.

4. Another application of this union/find data structure is the generation of mazes. Imagine a $m \times n$ rectangular grid of cells. These cells are numbered row-wise as $1, 2, ...m \times n$, starting from the top-left cell and all the way to the bottom-right cell. The entry point into the maze is the top-left cell and the exit point out of the maze is bottom-right cell. A simple algorithm to generate a maze is to start with an initial set up of this rectangular grid of cells where we imagine there are walls surrounding every cell that prohibit movement between cells (except the walls that permit entry into and exit out of the maze). Using this setup, continually choose a wall at random and knock it down when the two cells separated by a wall are not already connected to each other. If this process is repeated until the entry and exit cells are connected, then we have a maze.

   Using this description, implement an algorithm to generate a random maze. In fact, you can also implement a way to solve that randomly generated maze, using algorithms learnt in FIT2004.

```
-=0=-
 END
-=0=-
```