CSCB63 Assignment 2

Due: 11.59pm, Sunday Feb 24, 2019

1. (18 marks) Consider the following scenario. A set of machines are to process jobs of varying priorities. Each machine has it's own set of jobs stored in a priority queue. At times machines go down and we want to merge the priority queue of the down machine with one that is running. Once the down machine is up and running again, we'd like to split the priority queue of a machine to give the newly up and running machine a priority queue of jobs to complete. We will model this problem as special variation of a *heap* called a *dangling-heap*.

   A *dangling-heap* of height h is a tree consisting of

   - 2^h nodes
   - a root with exactly one child, which is the root of a complete binary tree of the remaining $2^h - 1$ nodes.
   - the root and (updated Feb. 14) and the complete binary tree satisfy a priority condition: the element at any node is the maximum of all the elements in the subtree rooted at that node (like a *heap*).
   - Note that a dangling-heap of **height 0 consists of a single node**. $\Leftarrow$ for this question, things work out slightly more neatly with this definition of height.

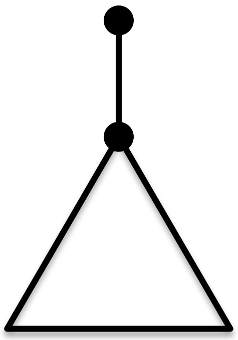

Figure 1: The heap has $2^h - 1$ nodes, so the dangling-heap has 2^h nodes

   A *dangling-heap forest* is a sequence of dangling-heaps $D_0, D_1, \ldots, D_m$ such that

   - height(D_{i-1}) $\leq$ height(D_i), for $0 < i \leq m$.
   - There are at most 2 dangling-heaps of any height.
   - There are at least $i + 1$ dangling-heaps of height at most i, for $0 \leq i \leq m$.
   - There are at most $i + 2$ dangling-heaps of height at most i, for $0 \leq i \leq m$.
   - The element at the root of D_{i-1} $\leq$ the element at the root of D_i, for $0 < i \leq m$.

   (a) (3 marks) Prove that if $D_0, D_1, \ldots, D_m$ is a dangling-heap forest, then it contains at least 2^m nodes and fewer than 2^{m+1} nodes.

(b) (3 marks) Give an efficient algorithm that given a tree of height h satisfying the heap conditions, constructs a dangling-heap forest $D_0, \ldots, D_h$ with the same nodes.

(c) (3 marks) Give an efficient algorithm that, given a dangling-heap forest $D_0, \ldots, D_m$, constructs a tree of height m with the same nodes satisfying the heap conditions.

(d) (3 marks) Give an efficient algorithm for splitting a dangling-heap of height h into two dangling-heaps of height $h-1$.

(e) (3 marks) Give an efficient algorithm for merging two dangling-heaps of height $h-1$ into a dangling-heap of height h.

(f) (3 marks) Give an algorithm that takes a sequence $D_0, \ldots, D_m$ of dangling-heaps that satisfies the first four properties of a dangling-heap forest and constructs a dangling-heap forest with the same nodes. Your algorithm should run in $O(m^2)$ time.

Explain why your algorithms are correct and run in the required times.

2. (10 marks) Two trees are edge-disjoint if there is no edge appearing in both of them.

(a) (3 marks) Let G be a graph with 2 edge-disjoint spanning trees. What is the least number of vertices, n', that G can have? Give an example of a graph on n' vertices which has 2 edge-disjoint spanning trees.

(b) (4 marks) For general $n \geq n'$, describe a graph on n vertices and a weight function such that the graph has exactly two edge-disjoint minimum spanning trees. Explain why your construction is correct.

(c) (4 marks)

Prove that if a graph has fewer than $2k$ vertices, then it cannot have k spanning trees such that every pair of them is edge-disjoint, when $k \geq 3$.

3. (12 marks) The classic Sudoku game involves a grid of 81 squares consisting of a 9×9 grid with 3×3 regions. The rules of the game are simple: each of the nine blocks has to contain all the numbers $1-9$ within its squares. Each number can only appear once in a row, column or box. The initial starting configuration contains a set of numbers that if correctly defined leads to only a single unique solution.

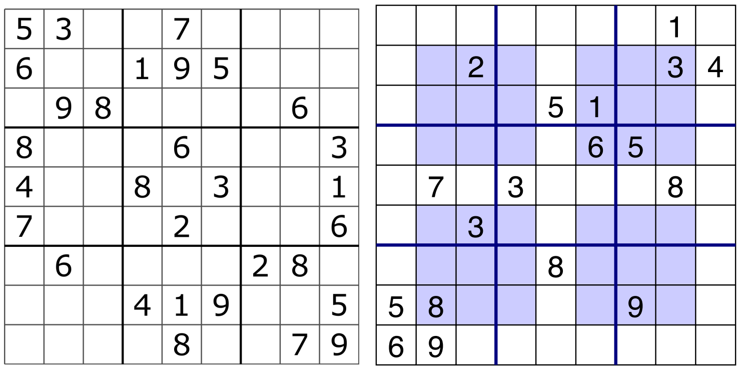

Figure 2: Sudoku and Hyper Sudoku

A variation of Sudoku is Hyper Sudoku. This variation uses the classic 9×9 grid with 3×3 regions, but defines four additional interior 3×3 regions in which the numbers $1 - 9$ must appear exactly once (the shaded regions in the example above).[1]

Write a program to solve 9×9 Hyper Sudoku puzzles. You may find it simpler to start by writing a Classic Sudoku solver first. Input specifications can be found in the starter code as well as test files and instructions on how to test your code.

Every solution is not equal; some take longer than others. Your program will be graded on whether if finds a solution and how quickly it finds a solution (if one exists). Try to think about choosing a good graph traversal algorithm (that you have seen in class) and how to minimize the number of vertices visited to optimize the time to find a solution.

Marking Scheme.

If your code looks like a general algorithm (as opposed to customizing for my test cases), you get 5 marks to start. In addition:

- If your code compiles: additional marks up to 5 for tests passed. But watch this: each test case gets 2 seconds only on the BV computers. Timing out is a failure. Each test case is timed separately.
- Additional marks up to 2 marks for speed - a faster solution will get full marks here. A slower solution will get 0 marks.
- If your code does not compile: 0 more marks.

If your code does not look like a general algorithm, 0 to 4 marks depending on how far off it is.

[1] images and description from wikipedia