Homework 08: Solutions

Due date: Wednesday, 08/15 at 11:59 pm

Instructions:

Submit a typed or neatly handwritten scan of your responses on Canvas in PDF format.

Note: you will need to submit a separate PDF for each problem.

The solutions to this assignment will be posted immediately after the due date. Therefore you cannot use late days on this assignment.

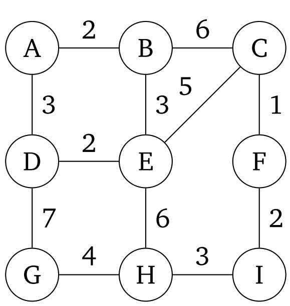
1. Minimum Spanning Trees

Draw the minimum spanning tree that results when running Kruskal's algorithm on the graph below. Please draw the vertices in the same order as in the graph below. Write final state of the array disjoint sets data structure from the run that results from using union-by-*SIZE* with *NO* path compression.

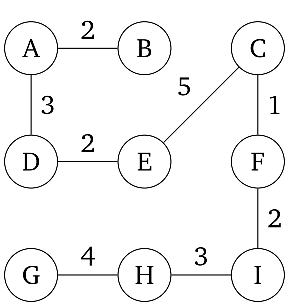
If there is a tie between edges in Kruskal's algorithm, choose the edge for which the endpoints, when written in alphabetical order, are alphabetically first. For example, if (D, A) and (B, C) were tied, you would chose (D,A) first because "AD" is alphabetically before "BC".

If two trees to be unioned have the same size, make the root of the unioned tree the representative element that comes alphabetically first. For example, if a tree rooted at A and a tree rooted at F are to be unioned but have the same size, then A would be used as the root of the combined tree.

When drawing the final array, assume that the mapping from vertex letter to array index is alphabetical: $A \rightarrow 0$, $B \rightarrow 1$, etc.



Solution:



2	0	-9	0	3	2	2	2	2

2. Dynamic Programming

This problem will walk you through the steps of designing a dynamic program. The problem we are solving is the longest palindrome problem: given a string, S, what is the length of the longest palindrome that is a substring of S? A palindrome is a string that reads the same forwards as backwards. For example, "racecar", "eve", and "I", are all palindromes. We also consider the empty string to be a palindrome (of length 0).

(a) First we need to figure out what our subproblems are. Since we are working with strings, a natural subproblem to use is substrings. Let $OPT(i,n)$ denote the length of the longest palindrome in the substring of length n starting at index i. Write an expression for the recursive case of $OPT(i,n)$. (Hint: All palindromes above a certain size have palindromes as substrings).

Solution:

$$OPT(i,n) = \begin{cases} 2 + OPT(i+1, n-2) & \text{if S[i] == S[i + n - 1] and OPT(i+1, n-2) == n-2} \\ \max\{OPT(i, n-1), OPT(i+1, n-1)\} & \text{otherwise} \end{cases}$$

(b) Next we need a base case for our OPT recurrence. Write an expression for the base case(s) of this recurrence. (Hint: Which size strings are always palindromes?)

Solution:

$$OPT(i,n) = n \qquad \text{if n < 2}$$

(c) Now that we have a complete recurrence, we need to figure out which order to solve the subproblems in. Which subproblems does the recursive case $OPT(i,n)$ require to be calculated before it can be solved?

Solution:

Problems of smaller substrings: $OPT(\_, n-1)$ and $OPT(\_, n-2)$.

(d) Given these dependencies, what order should we loop over the subproblem in?

Solution:

In order of increasing n (order of i) does not matter.

(e) We have all of the pieces required to put together a dynamic program now. Write psuedocode for the dynamic program that computes the length of the longest palindromic substring of S.

Solution:

function palindrome(S)
    OPT = new int[S.length][S.length + 1]
    Set OPT[i][n] = n for all i and all $n < 2$.
    for n= 2 …S.length **do**
        for i = 0 …S.length - n - 1 **do**
            if S[i] == S[i + n] and OPT[i+1][n-2] == n-2 **then**
                OPT[i][n] = 2 + OPT[i+1][n-2]
            else
                OPT[i][n] = max { OPT[i][n-1], OPT[i+1][n-1] }
    return OPT[0][S.length]