Data Structures and Algorithms – (COMP SCI 2C03)
Winter, 2021
Assignment-III

Due at 11:59pm on April 12th, 2021

- **No late assignment accepted.**
- Make sure to submit a version of your assignment ahead of time to avoid last minute uploading issues.
- Submit one assignment solution as a PDF file per group on Avenue.
- If the solution submitted by two groups is the same. Both teams will get a zero mark on the assignment.
- Present your algorithms in Java or Pseudocode (Pseudocode is preferred).
- It is advisable to start your assignment early.

This assignment consists of 8 questions, and is worth 40 marks.

1. How would you use Prim's and Kruskal's algorithms to compute the minimum spanning forest of an edge-weighted graph that is not connected? For this question only outline the procedure and discuss both algorithms as well as how your approach would differ depending on the algorithm used. No need for an implementation. Note: A minimum spanning forest is a collection of MSTs computed for each connected component.[5 marks]
   Answer: Kruskal's algorithm computes the minimum spanning forest (MSF); that is, when the input graph is disconnected, it computes the MST for each component. Prim's algorithm, however, fails to compute

the MSF for a disconnected graph. Below, is the outline to compute the MSF using Prim's algorithm. Note that, the below outline also works if Prim's algorithm is replaced by Krushkal's algorithm.

(a) Compute the connected components of the edge weighted graph G using Union-Find, so that the id array can be used to identify its various connected components. Quick find approach is more suitable here, as the primary purpose of the id array is to quickly find if the two vertices belong to the same component.

(b) Let c be the number of connected components. Then the graph G can be partitioned into sub graphs representing the connected components $G_1, G_2, \ldots, G_c$.

(c) Initially the minimum spanning forest (MSF) F is empty.

(d) Execute Prim's algorithm on G_i, where $1 \leq i \leq c$, to compute the MST T_i and add it to F; that is, $F = F \cup \{T_i\}$

(e) Return F.

2. Given an MST for an edge-weighted graph G, suppose that an edge in G that does not disconnect G is deleted. Describe how to find an MST of the new graph in time proportional to E. [4 marks]
Answer: If the edge to be deleted is not part of the original MST, then nothing needs to be changed - the original MST still holds.

   If the edge to be deleted is part of the original MST (T), after deletion T is divided into two disconnected MSTs T_1 and T_2. We partition the set of vertices V into two set S and S', where S is the set consisting of all and only the vertices in T_1, and S' consists of all the remaining vertices in V; that is, all the vertices in T_2. By the cut property the minimum weighted crossing edge from S to S' will be part of the MST for G. We know that such an edge exists, as G remains connected even after the edge deletion. Therefore, to compute an MST for G, we examine each crossing edge between S and S', and choose the edge with the minimum weight. Since there are at most E edges to examine, this proposed solution runs in time proportional to E.

3. Given a digraph with positive edge weights, and two distinguished subsets of vertices S and T, find a shortest path from any vertex in S to any vertex in T. Your algorithm should run in time proportional to

$E \log V$, in the worst case. Only provide an outline/pseudocode for the algorithm [4 marks for the algorithm, 1 marks for explaining it runs in $E \log V$]
Answer: The outline of the algorithm is below:

(a) Create a new vertex s with edges of weight 0 from s to each vertex in S.

(b) Execute Dijkstra's algorithm with s as the source. Dijkstra's algorithm computes the shortest path from s to every other vertex in G, and stores these distances in the $distTo[]$ array of length V.

(c) To find the shortest path from some vertex in S to some vertex in T, we simply scan the $distTo[]$ array to compute the minimum over all the shortest path values from s to all vertices $v \in T$. This step takes at most $O(V)$ time.

Dijkstra's algorithm for the graph including s and the zero weighted edges from s to all $v \in S$ requires at most $O(2E \log(V+1)) = O(E \log V)$. Hence, the total running time of the algorithm is of the order $E \log V + V$. Since $|E| \geq V$, the proposed algorithm runs in $O(E \log V)$. Therefore, the above solution runs in time proportional to $E \log V$.

4. Given a weighted digraph, find a monotonic shortest path from s to every other vertex. A path is monotonic if the weight of every edge on the path is either strictly increasing or strictly decreasing. The path should be simple (no repeated vertices). Only provide an outline/pseudocode for the algorithm. No need to give an implementation. Hint : Relax edges in ascending order and find a best path; then relax edges in descending order and find a best path. [4 marks]
Answer: The outline of the algorithm is as follows:

(a) We want to initially relax all edges in ascending order to get a monotonic increasing shortest path.

(b) We then relax edges in descending order to get a monotonic decreasing shortest path.

(c) As the shortest path will be whichever path has the shortest weight, we would return that path. If a monotonic path does not exist, we may return false.

5. Develop a version of key-indexed counting that uses only a constant amount of extra space. Your developed version need not be stable. [4 marks]
Answer: The solution is given in Figure 1.

```
// Sort an array arr[] of N integers between 0 and R - 1
void count_sort(int arr[]) {
    int N = arr.length;
    int[] count = new int[R];

    for(int i=0; i<N; i++) {
        count[a[i]]++;
    }

    int write = 0;
    for(int r=0; r<R; r++) {
        while(count[r] > 0) {
            arr[write++] = r;
            count[r]--;
        }
    }
}
```

Figure 1: Solution for Q5

6. Write an algorithm that, given two strings of equal length n, determines whether one is a cyclic rotation of the other, such as example and ampleex in time $O(n)$. A general explanation of why your algorithm runs in $O(n)$ time will suffice. You are not required to compute $T(n)$ and proof that $T(n) \in O(n)$. Note: A cyclic rotation of a string $s = uv$ is a string $s' = vu$. When $u = \varepsilon$, then $s = s'$. [5 marks for algorithm, 1 marks for explanation of it being in $O(n)$].
Answer: Algorithm 1 presents the solution for this problem. Algorithm 1 computes β_w which requires $O(2n) = O(n)$ time. Apart from this, the algorithm has only one `while` loop, executing statements taking constant time, and executing at most n times. Therefore, Algorithm 1 runs in $O(n)$ time.

7. A tandem repeat of a base string u in a string s is a substring of s having at least two consecutive copies of u (non-overlapping). Develop a linear-time algorithm that, given two strings u and s, returns the index of the beginning of the longest tandem repeat of u in s. For example, your algorithm should return 3 when $u = abcab$ and

Algorithm 1 Algorithm to check if strings u, v are cyclic rotations

```
procedure isRotation(u, v)
    w = uv                                          ▷ We assume |w| ≥ 2
    β_w ← Border array of w
    b ← β_w[2n]                                     ▷ b stores the max. border of w.
    i ← 0              ▷ We assume any string s is indexed from 0 to |s| − 1.
    while (i ≤ n − b) do
        if (u[i + b] ≠ v[i]) then
            return false
        else
            i = i + 1
    return true
```

$s = abcabcababcababcababcab$. [5 marks for algorithm, 1 marks for explaining that it is linear-time]

Answer: Algorithm 2 presents the solution for this problem. KMP requires $O(N + M)$ time, where $N = |s|$ and $M = |u|$. The maximum length of $patList$ is $N - M$. Since Algorithm 2 has only one while loop executing statements taking constant time, and since this loop executes at most $N - M$ times, the running time of Algorithm 2 is $O(N + M)$.

8. (i) Give the optimal Huffman code for the symbols a, b, c, d, e, f, g, h having frequencies $1, 1, 2, 3, 5, 8, 13, 21$ respectively. Observe that the frequencies listed are the first 8 Fibonacci numbers. [3 marks]
Answer: The Huffman prefix-free code trie is given in Figure 2. The code for each symbols is given below.
$A = 1111111$
$B = 1111110$
$C = 111110$
$D = 11110$
$E = 1110$
$F = 110$
$G = 10$
$H = 0$

   (ii) Generalize your answer for [i] to find the optimal Huffman code for n symbols, having their frequencies as the first n Fibonacci numbers, for a general n; that is, the i-th symbol has frequency

Algorithm 2 Algorithm to identify the longest tandem repeat of u in s

procedure LONGESTTANDEMREPEAT(s, u)
  $maxI = -1$ ▷ stores the starting index of the global max. TR of u
  $curI = -1$ ▷ stores the starting index of the local max. TR of u
  $maxC = 0$ ▷ count of repeats in global max. TR of u
  $curC = 0$ ▷ count of repeats in local max. TR of u ▷ We assume patList is an array consisting of all occurrences of u, in s
  $patList \leftarrow KMP(s, u)$
  if $(|patList| = 0)$ **then**
    return -1
  ▷ We assume any string s is indexed from 0 to $|s| - 1$.
  $curI = maxI = patList[0]; i = 1$
  $maxC = 1; curC = 1$
  while $i < |patList|$ **do**
    if $(patList[i] = curI + curC * |u|)$ **then**
      $curC = curC + 1$
      if $maxC < curC$ **then** $maxC = curC; maxI = curI$
    else if $(patList[i] < curI + curC * |u|)$ **then** ▷ do nothing
    else
      $curI = patList[i]; curC = 1$
    $i = i + 1$
  return $maxI$

F_i, where $1 \leq i \leq n$. [3 marks]
Answer: Let $H[n]$ denote the Huffman code for the symbol whose frequency is equal to the n-th fibonacci number, then we have:
$H[n] = 0$
$H[n-1] = 10$
$H[n-2] = 110$
$\vdots$
$H[n-i] = 1^i0$
$\vdots$
$H[2] = 1^{n-2}0$
$H[1] = 1^{n-1}$

Note that depending on how you arrange the nodes having frequencies, $F_1 = F_2 = 1$, you might get t$H[1] = 1^{n-2}0$ and $H[2] = 1^{n-1}$. Apart

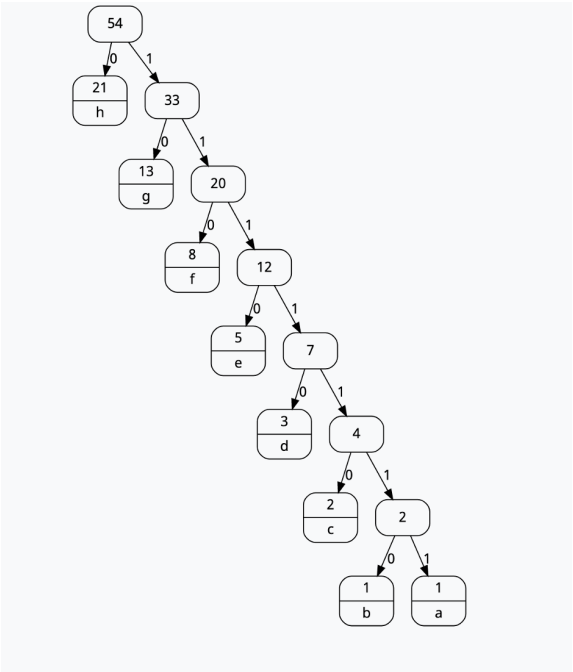

Figure 2: Trie for Q8a solution

from this, the prefix-free codes for other symbols remain the same as shown above.