Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Total
10	10	20	9	5	15	5	10	6	10	100

CMPE 224 / 343 – Practice Exam 2

Date: 11.06.2023

Name:
Student ID:

You have 120 minutes for this exam. The exam is closed book, closed notes, except that you are allowed to use an **A4-size, double sided, handwritten** cheat sheet.

1. (10 points) Draw the R-way trie for storing the following string keys. Assume that the value for each string key is the index of that string in the input (e.g. 0 for `Feld`, 1 for `Huhn`, 2 for `Bauer`, etc.).

   `Feld, Huhn, Bauer, Katze, Hund, Hof, Baum, Hahn, Haus`

2. (10 points) Draw the TST that results when the below keys are inserted in that order into an initially empty TST. Assume that the value for each string key is the index of that string in the input list (e.g. 0 for `as`, 1 for `at`, 2 for `be`, etc.).

   `as at be by he in is it of on or to`

3. (20 points) Given the Trie data structure given below, write a method `findKeys(int n)` that returns the list of keys with lengths smaller than a specified length *n* stored in the Trie. Your algorithm should work in linear time, and it should not visit more characters than those necessary. Note: You may use a helper function for recursion. **Make sure to include explanatory comments in your algorithm.**

```
public class TrieST<Value>
{
   private static int R = 256; // radix
   private Node root;          // root of trie

   private static class Node
   {
      private Object val;
      private Node[] next = new Node[R];
   }
```

```
public Iterable<String> findKeys(int n)  {
```

What is the computational complexity of your algorithm, in big-O notation? Explain your answer.

4. (9 points) Answer whether the following statements are true or false. For complete credit, **explain** your answer clearly.

   i) (TRUE: __ / FALSE: __) Kruskal's MST algorithm is an example of a greedy algorithm.

   Explanation:

   ii) (TRUE: __ / FALSE: __) Prim's algorithm is good for sparse graphs, whereas Kruskal's algorithm is good for dense graphs.

   Explanation:

   iii) (TRUE: __ / FALSE: __) For any graph, there is a unique minimum spanning tree.

   Explanation:

5. (5 points) What is the order of growth of Dijkstra's algorithm if we use an ordered array instead of the Priority Queue? Assume there are no self-edges or parallel edges. For full credit, explain your answer.

6. (15 points) Modify Dijkstra's algorithm to find the shortest paths from a given source vertex 's' to all other vertices in a directed weighted graph, but this time, consider a scenario where each edge has an associated *toll* (or *cost*). The objective is to find the shortest path based on both the distance and the minimum toll. In case of multiple paths with the same distance, prioritize the path with the minimum toll.

   Give the algorithm (in pseudocode or in Java) in sufficient detail. Show the necessary modifications to the original Dijkstra algorithm.

   a) Explain your idea in a few sentences:

   b) Give your algorithm in sufficient detail:

   c) What is the computational complexity of your algorithm, in big-O notation? Explain your answer.

7. (5 points) Why would anyone prefer to use string sort algorithms, instead of a general-purpose Quick Sort or Merge Sort algorithm?

8. (10 points) The column on the left is the original input of 24 strings to be sorted; the column on the right are the strings in sorted order; the other 7 columns are the contents at some intermediate step during one of the 3 radix sorting algorithms listed below. Match up each column with the corresponding sorting algorithm. You may use a number more than once.

```
mink  bear  bear  calf  crow  myna  crab  bear  bear
moth  calf  calf  lamb  lamb  crab  toad  crow  calf
crow  crow  crow  hare  deer  lamb  swan  calf  crab
myna  crab  crab  wasp  crab  toad  bear  crab  crow
swan  deer  hare  hawk  hare  mule  deer  deer  deer
wolf  hare  kiwi  ibex  bear  hare  ibex  hare  hare
mule  hawk  deer  bear  kiwi  sole  hoki  hawk  hawk
slug  hoki  hawk  deer  calf  wolf  mule  hoki  hoki
hare  ibex  ibex  mink  hawk  calf  sole  ibex  ibex
bear  kiwi  hoki  lion  ibex  slug  wolf  kiwi  kiwi
kiwi  lion  lion  kiwi  hoki  moth  calf  lion  lamb
calf  lynx  lynx  slug  lion  kiwi  lamb  lynx  lion
hawk  lamb  lamb  toad  lynx  hoki  myna  lamb  lynx
ibex  mink  mink  hoki  mink  mink  mink  mink  mink
oryx  moth  mule  sole  mule  hawk  lynx  moth  moth
lion  myna  myna  wolf  myna  swan  lion  myna  mule
sole  mule  moth  moth  moth  lion  crow  mule  myna
wasp  oryx  wasp  crab  wasp  wasp  hare  oryx  oryx
lynx  swan  sole  crow  sole  bear  wasp  swan  slug
hoki  slug  oryx  oryx  oryx  deer  moth  slug  sole
crab  sole  slug  mule  slug  crow  slug  sole  swan
deer  toad  wolf  swan  wolf  ibex  kiwi  toad  toad
lamb  wolf  toad  myna  toad  oryx  hawk  wolf  wasp
toad  wasp  swan  lynx  swan  lynx  oryx  wasp  wolf
----  ----  ----  ----  ----  ----  ----  ----  ----
 0                                                4
```

(A) Original input

(B) LSD radix sort

(C) MSD radix sort

(D) 3-way radix quicksort (no shuffle)

(E) Sorted

9. (6 points) Why is the *Knuth-Morris-Pratt* substring search algorithm more efficient than the *naive* method (which checks, for each possible position in the target text at which the pattern could match, whether or not it does match)?

10. (10 points) Create the *Knuth-Morris-Pratt DFA* for the string `aacaaab` over the alphabet `{a,b,c}`. As usual, state 0 is the start state and state 7 is the accept state. Draw the DFA both graphically and as a table.