1.

a.

Explain the difference between
closed hashing = open addressing
and open hashing = closed addressing.

There is just one point for being able to say which is which, so don't be too worried about mixing them up.

What are their relative advantages and disadvantages?

b.

Give a C++ implementation of a hash table of dictionary entries.
A dictionary entry consists of a word and its definition, both strings.
You can use either open or closed hashing.

Include a constructor,
a method for adding a new entry,
a method for looking up a word and returning its definition,
and a destructor.

# 2.

An non-text “binary” file called `StolenNumbers.dat` contains a lot of personal details.

The file consists of a large number records for people, each record consists of 64 bytes, exactly as follows.

- 1 thirty-two bit int for birth date
- 20 characters for First Name (padded with spaces if necessary)
- 20 characters for Last Name (padded with spaces if necessary)
- 1 thirty-two bit int for social security number
- 16 characters for Primary Credit-card number

So, if the file has the records for 1,000 people, it will be exactly 64,000 bytes long.

Write a C++ program that uses the standard <iostream>, <fstream> and <string> functionality to sort this file so that the names appear in alphabetical order, primarily by last name, but when last names are the same, use first names.

Use any sorting algorithm you like.

This sort should be performed on disc. DO NOT read the file into an array and sort it there.

Imagine you are programming a very small microprocessor that has very limited memory capacity, but is connected to a normal capacity disc drive. The processor can only store a very few records in memory at once.

The sorting is not expected to be very fast.

3.

a.

What is a priority queue?
What are the operations that a fully functional priority queue must be able to perform?
Briefly describe (just a couple of sentences) a possible implementation of a priority queue.

b.

One simple representation for a graph is to use an *adjacency matrix*: each of the N nodes is given a unique number in the range 0 to N-1 to identify it. A large two dimensional array A with N rows and N columns is created so that

A[x][y] stores the cost of travelling directly from node x to node y.
if A[x][y] is zero, then there is no direct connection from x to y.
A[x][y] does not need to equal A[y][x] - there can be one-way links.

Assume that a priority queue exactly as you described for part A has been implemented, and you can use it. Also assume that an adjacency matrix as just described has been created for an N node graph, and you can use it too.

Write a function that finds the length of the shortest path (or the cost of the cheapest path) between nodes S and D, which are provided as parameters.

4.

A.

Give the algorithm for adding a new value to a (min) heap.

B.

Give the algorithm for removing the smallest item from a (min) heap.

C.

Code part B in C++.

5.

The following numbers are to be added to an initially empty 2-3-tree.
Draw the 2-3-tree clearly, as it would be after each individual insertion.
(you may occasionally skip a diagram if it shows no significant change)

i. 30

ii. 50

iii. 80

iv. 40

v. 70

vi. 20

vii. 90

viii. 84

ix. 82

x. 88

xi. 86

xii. 97

xiii. 35

xiv. 45

xv. 10

xvi. 15

xvii. 99

6.

A file contains a list of roads, one per line, in this format:

A R C

where

A is the name of a town
R is the name of a road
C is the name of another town, not the same as A.
A, R, and C are simple strings and they are separated by a space.
No two towns or roads will have the same name.

The only way you know that a town exists is that it appears in one of more of these lines. The same applies to roads.

Design and implement software that will read a file in this format, then find the optimum route between two towns whose names are provided by the user.

The optimum route is the one that has the smallest number of different roads: the vehicle has trouble with corners.

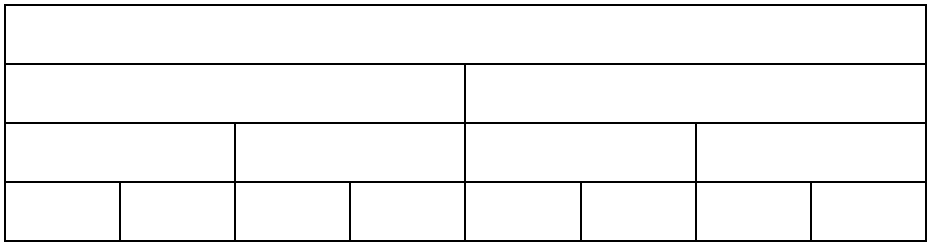
The output should be the list of roads to follow (in order), or “NO WAY” if no route can be found.

7.

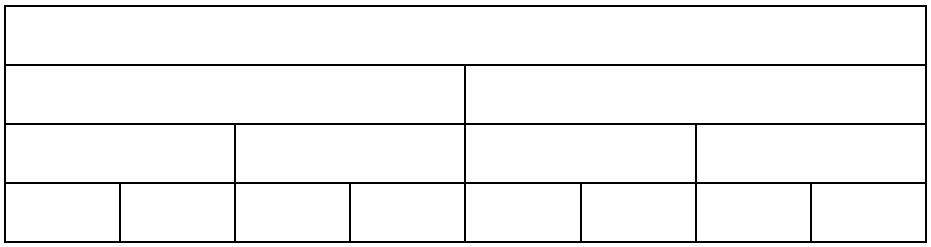
A min-heap is to be constructed, containing only strings, and using the usual ordering for strings (that is, the C++ operators <, >, ==, etc. will do the job).

Show the contents of the heap, all in its correct place, after each of these operations. The heap is empty at the start.

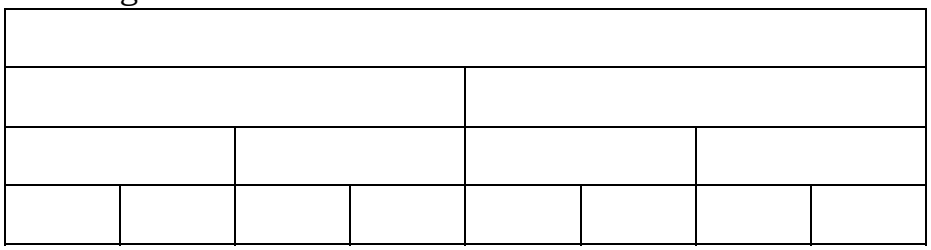
A. insert “Hat”



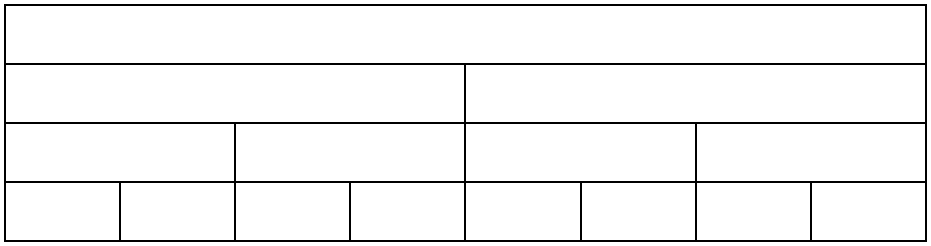
B. insert “Lion”



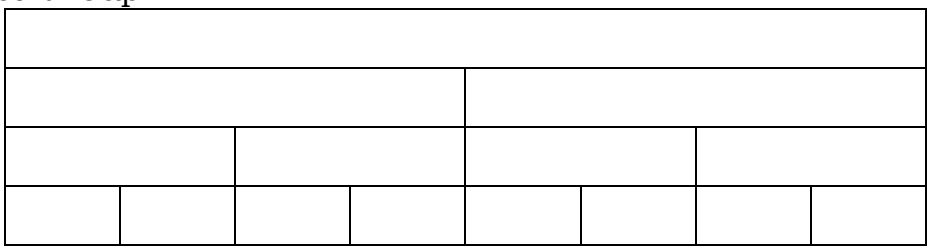
C. insert “Dog”



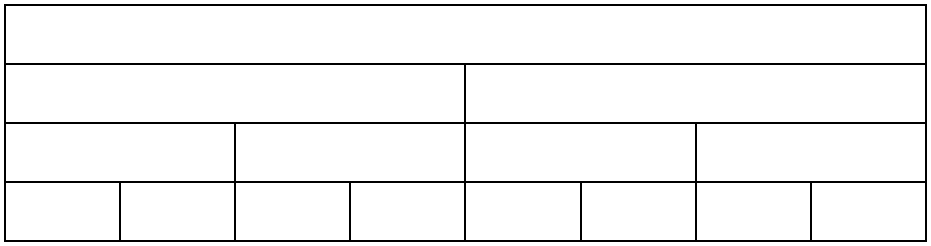
D. insert “Owl”



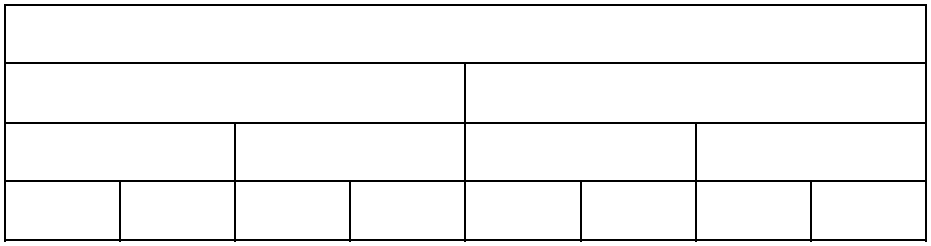
E. insert “Gap”



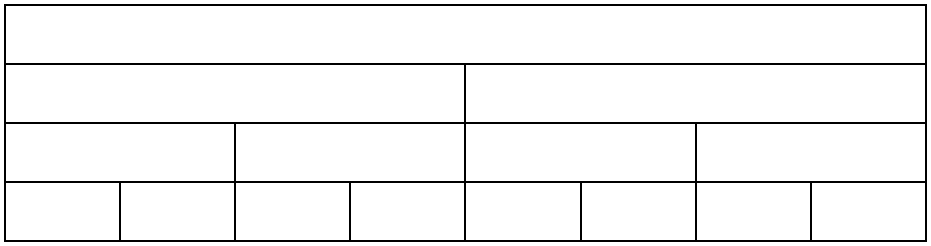
F. insert “Ink”



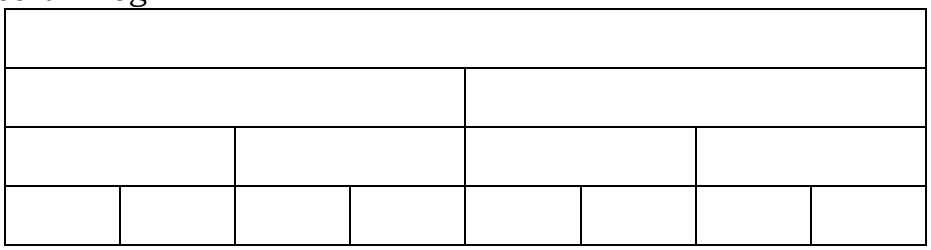
G. insert “Cat”



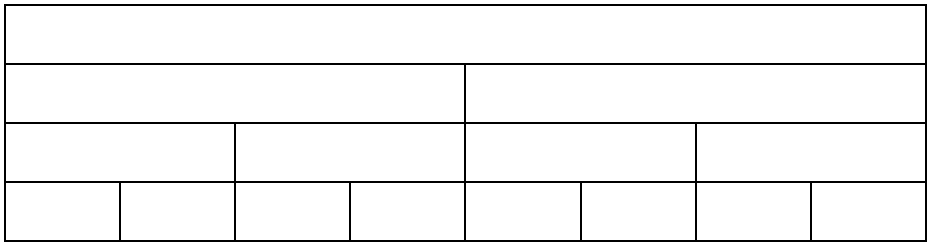
H. insert “Yeti”



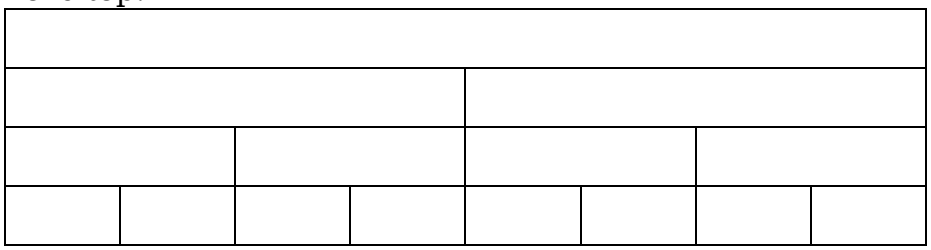
I. insert “Frog”



J. insert “Eel”



K. remove top.



8.

a.

Show how a 2-3-tree can contain 10 data items and still be perfectly balanced.
A simple diagram can be enough.

b.

Design a C++ struct suitable for representing a node in a 2-3-tree.

c.

Write a function that inserts a new data item into a 2-3-tree.
Or get as close as you can. A good clear design is most important.

9.

The ultimate aim of this question is to solve the "longest monotonic subsequence" problem with dynamic programming. It is an easy problem to understand once you are told that here "monotonic" just means "non-decreasing". These two sequences are monotonic:

4, 6, 6, 7, 8, 12, 34, 34, 34, 34, 41
5, 6, 7, 8, 9, 10, 11

These two are not monotonic:

4, 6, 6, 7, 8, 12, 34, 30, 34, 34, 41
5, 6, 7, 8, 9, 10, 9

This sequence:

54, 43, 34, 43, 56, 65, 65, 64, 65, 66, 70, 40, 42, 30, 9, 10

is not monotonic but has 7 (non-overlapping) monotonic subsequences:

54 and 43 and 34, 43, 56, 65, 65 and
64, 65, 66, 70 and 40, 42 and 30 and 9, 10

There is nothing to say that the subsequences can not overlap, so 56, 65 and many other are also monotonic subsequences, but overlapping is not relevant to this question.

The solution could be to produce the subsequence itself, or just to give its beginning and ending positions, you can choose.

You may assume that the input (the full sequence) will be provided in an array or a vector or read from a file or anything reasonable.

Your answers do not need to be coded in any particular programming language, but must still be complete descriptions that leave nothing for the reader to work out.

a.

Give a brute-force algorithm. That means it can be as inefficient as you like so long as it would eventually find the answer if given a totally unlimited amount of time.

b.

Give a Dynamic Programming solution.