



```
1 public interface IceCave {
2     /** Return the terrain of the level, a matrix of positions. */
3     Position[][] getTerrain();
4
5     /** Return whether the position P is in bounds. */
6     boolean inBounds(Position p);
7
8     /** Slide in the given direction until an obstacle is hit. */
9     Position move(Position p, Direction d);
10 }
11
12 public interface Position {
13     boolean isIce();
14     boolean isRock();
15     boolean isGoal();
16 }
17
18 public enum Direction {
19     LEFT, UP, DOWN, RIGHT
20 }
21
22 public LinkedList<Position> solve(IceCave cave, Position start) {
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54 }
```

# STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

## 2 Data Structures & Algorithms Potpourri

For each of the following, give a data structure or algorithm that you could use to solve the problem or write **impossible** if it is impossible to meet the running time given in the question. For each question, we have one or more right answers in mind, all of which are among the data structures and algorithms listed below. For each answer, provide a brief description of how the algorithm or data structure can be used to solve the problem.

Possible answers:

DFS	Heap	Quick sort
BFS	Trie	Merge sort
Dijkstra's	Hash table	Insertion sort
Topological Sort	Balanced BST	Radix sort
Kruskal's	Linked list	Selection Sort

- (a) Given a list of  $N$  words with  $k$  characters each, find for each word its longest prefix that is the prefix of some other word in the list. Worst-case running time:  $O(Nk)$  character comparisons.
- (b) Given an undirected, weighted and connected graph with  $|E|$  edges, find the heaviest edge that can be removed without disconnecting the graph. Worst-case running time:  $O(|E| \log |E|)$ .
- (c) We would like to build a data structure that supports the following operations: add, remove and find the  $k^{\text{th}}$  largest element for any  $k$ . Worst-case running time:  $O(\log N)$  comparisons for each operation (where  $N$  is the number of elements currently in the data structure).
- (d) Given an unordered list of  $N$  Comparable objects, construct a binary search tree containing all of them. Running time:  $O(N)$  comparisons.

# STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

### 3 Hashing

The (unmemoized) `hashCode` method for the `String` class is as follows:

```
1 public int hashCode() {  
2     int h = 0;  
3     for (int i = 0; i < length() ; i++) {  
4         h = 31 * h + charAt(i);  
5     }  
6     return h;  
7 }
```

For parts (a) and (b), assume that a `HashSet` uses the value of this function by taking it modulo the number of buckets, after first masking off the sign bit to make the number non-negative (The actual `HashSet` and `HashMap` implementations do something more sophisticated.). The other parts of the problem are disjoint from (a) and (b).

- (a) Given a `String` of length  $L$  and a `HashSet` that contains  $N$  `String`s, give the worst- and best-case running times of inserting the `String` into the `HashSet`.
- (b) In Java, `HashSets` always use arrays whose size is a power of two. If this were not the case, the `hashCode` method shown above could be a very poor hash function. Give an example of an integer  $N$  such that `hashCode` would be a very poor choice of hash function for a `HashSet` whose array had size  $N$ . Give a brief explanation of your answer. Assume that the Java `HashSet` uses external chaining to resolve collisions.
- (c) What is a collision in a hash table? Select the best definition below.
- Two key-value pairs that have equal keys but different values.
  - Two key-value pairs that have different keys and hash to different indices.
  - Two key-value pairs that have different keys but hash to the same index.
  - Two key-value pairs that have equal keys but hash to different indices.
- (d) Suppose that your hash function does not satisfy the uniform hashing assumption. Which of the following can result? Select all that apply. For each one selected, give a brief explanation of why it would result.
- Poor performance for insert.
  - Poor performance for search hit.
  - Poor performance for search miss.
- (e) Suppose that instead of using a linked list for our external chaining, we instead use another `HashMap`. List some disadvantages of this approach.

**STOP!**

Don't proceed until everyone in your group has finished and understands all exercises in this section!

## 4 Synthetic Trees

Complete the following program by filling in the blank line on this page and adding any necessary code to the next page.

```

1  public interface Tree {
2      /** Return true iff this Tree contains X. */
3      boolean contains(int x);
4
5      /** Insert X, if not already present, and return the resulting tree. */
6      Tree insert(int x);
7
8      /** The empty tree, containing nothing. */
9      static Tree emptyTree() {
10
11         return -----;
12     }
13 }
14
15 public final class RegularTree implements Tree {
16     /** NOTE: Final classes cannot be extended. */
17     private int label;
18     private Tree left, right;
19
20     /** A Tree containing label LAB and the contents of trees L and R,
21      * where all elements of L are < LAB and those of R are > LAB. */
22     RegularTree(int lab, Tree L, Tree R) {
23         this.label = lab;
24         this.left = L;
25         this.right = R;
26     }
27
28     @Override
29     public boolean contains(int x) {
30         if (x == label) {
31             return true;
32         } else if (x < label) {
33             return left.contains(x);
34         } else {
35             return right.contains(x);
36         }
37     }
38
39     @Override
40     public Tree insert(int x) {
41         if (x < label) {
42             left = left.insert(x);
43         } else if (x > label) {
44             right = right.insert(x);
45         }
46         return this;
47     }
48 }

```

Add any additional code on the next page. It **may not** contain any if statements, while statements, switch statements, conditional expressions, or try statements.

## 5 Disjoint Trivia

- (a) Explain how the disjoint sets data structure could be used in an implementation of Kruskal's MST algorithm. Consider efficiency. Would you actually want to use this data structure in this algorithm? Explain.
  
- (b) True or False: To improve the efficiency of joins on two trees, we keep track of the height of the two trees and always link the root of the shorter tree to the root of the taller tree.
  
- (c) Extra: Assume we are using disjoint sets with path compression. How many calls to `find()` need to be made in order for each node to be directly connected to the root node?

**STOP!**

Don't proceed until everyone in your group has finished and understands all exercises in this section!

## 6 Sorting Mechanics

Below, the **leftmost column** is an unsorted list of strings. The **rightmost column** gives the same strings in sorted order. Each of the remaining columns gives the contents of the list during some intermediate step of one of the algorithms listed below. Match each column with Merge sort, Quicksort, Heap sort, LSD radix sort, or MSD radix sort. For quicksort, choose the topmost element as the pivot. Use the recursive (or top-down) implementation for merge sort.

	A	B	C	D	E	F	G
1	4873	1876	1874	1626	9573	2212	1626
2	1874	1874	1626	1874	7121	8917	1874
3	8917	2212	1876	1876	9132	7121	1876
4	1626	1626	1897	4873	6973	1626	1897
5	4982	3492	2212	4982	4982	9132	2212
6	9132	1897	3492	8917	8917	6152	3492
7	9573	4873	4873	9132	6152	4873	4873
8	1876	9573	4982	9573	1876	9573	4982
9	6973	6973	6973	1897	1626	6973	6152
10	1897	9132	6152	3492	1897	1874	6973
11	9587	9587	7121	6973	1874	1876	7121
12	3492	4982	8917	9587	3492	9877	8917
13	9877	9877	9132	2212	4873	4982	9132
14	2212	8917	9573	6152	2212	9587	9573
15	6152	6152	9587	7121	9587	3492	9587
16	7121	7121	9877	9877	9877	1897	9877

## 7 Trying to Find Partial Matches

Given a list of  $N$  input words all of length at most  $k$  and  $M$  query words, we would like to find, for each query word, the number of input words that match the first  $k/2$  letters of the query word. Describe an algorithm that accomplishes this and give its running time as a function of  $N$ ,  $M$ , and  $k$ .

**STOP!**

Don't proceed until everyone in your group has finished and understands all exercises in this section!

## 8 T9

You're living in the future in the late 1990's. Everyone owns a cell phone — the handheld kind — and SMS is the bee's knees. However, cell phones only have 9 keys while there are 26 letters in the alphabet. Your company is on the verge of developing a new algorithm for faster texting called "Text on 9 keys", or "T9". Each keypress maps to one of three or four letters in the alphabet. Given a trie containing the dictionary of words, define a procedure, `getWords`, that returns the set of matching words for a given key-press sequence.

```

1  public interface TrieNode {
2      public char getCharacter();
3      public boolean isWord();
4      public String getWord();
5      public Map<Character, TrieNode> getChildren();
6  }
7  public class T9 {
8      private static char[][] KEY_MAPPINGS = {
9          {}, {}, // keys 0 and 1 don't map to any characters
10         {'a', 'b', 'c'},
11         {'d', 'e', 'f'},
12         {'g', 'h', 'i'},
13         {'j', 'k', 'l'},
14         {'m', 'n', 'o'},
15         {'p', 'q', 'r', 's'},
16         {'t', 'u', 'v'},
17         {'w', 'x', 'y', 'z'}
18     };
19     public static Set<String> getMatches(int[] keyPresses, TrieNode words) {
20         return getMatches(keyPresses, words, 0);
21     }
22     private static Set<String> getMatches(int[] keyPresses, TrieNode wordNode, int
23         index) {
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45     }
46 }

```

# STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!