Optional Problems 4

No Due Date

Handin

Optional problems will not count towards your grade in any way and are provided as an opportunity for you to get more practice.

1 Written Problems

Problem 4.1

Using Multiple Data Structures

Write pseudocode to remove duplicates from an unsorted linked list in $O(n)$ time.

Solution:

```
deleteDuplicates(node):
  """ Transforms: node, the head of an unsorted linked list -> none

      Purpose: This function works by going through the linked list
      and hashing each node, removing the node from the linked
      list if the hashtable already contains it
  """
  h = Hashtable()
  prev = null
  while (node != null):
       if (h.contains(node)):
            prev.next = node.next
       else:
            h.insert(node)
            prev = node
       node = node.next
```

Problem 4.2

Binary Search in a Matrix

So far you have been taught how to binary search through a one dimensional list of sorted numbers. In this problem you will binary search through a two dimensional matrix of numbers which are sorted in a particular form.

Problem

Let the matrix be of size $n \times n$ where

1. Each row is in non-decreasing (non-strictly increasing) order.
2. Each column is in non-decreasing (non-strictly increasing) order.

Determine, in $O(n)$ time, whether a particular element is in the matrix. Then write a brief explanation of why your solution is $O(n)$.

Example

Here is an example of a matrix that satisfies the above conditions.

$$\begin{array}{ccccc}
1 & 3 & 4 & 7 & 8 \\
4 & 5 & 6 & 8 & 8 \\
7 & 8 & 9 & 12 & 15 \\
8 & 10 & 11 & 15 & 16 \\
10 & 11 & 13 & 16 & 20
\end{array}$$

How you would search for 1 in the matrix above. What about 6? 19? 21?

Hints

It is a difficult jump to get to the answer for this problem so feel free to use the hints below, just make sure to think through the problem and come up with a couple ideas on your own first. They go in order of increasing helpfulness so go through them one at a time.

1. Ordinary binary search revolves around a pivot which is choosen as the $\frac{length}{2}$ element in the list. We can test whether the element we're searching for is smaller or larger than the pivot and eliminate part of the list.

   Which position in the matrix should be choosen as a pivot?

2. If you haven't determined the position of the pivot think about the layout of the matrix. In our matrix binary search, the smallest elment is in the upper left corner. Which position is the largest element in? Which position is near the middle (maybe not exactly the median) element in?

3. In ordinary binary search we test whether the element we're searching for is smaller or larger than the pivot. From this comparison we can eliminate half of the list.

   Using the pivot you determined above can you eliminate any elements in the matrix?

4. Since each row is increasing we know that the first element is the smallest and the last element is the largest. If the number we are searching for is smaller than the smallest element in the row, what can we say about the row? What if it is higher than the highest number in the row?

5. The above statement is also true for columns. How can you combine this information to form a solution?

Solution:

Answer

In binary search you choose your pivot to be the middle element ($\frac{length}{2}$) in the list which guarantees that you will be able to eliminate half of the list at each step.

In this matrix binary search we are going to use the upper right (lower left works as well) element as our pivot.

- If our element = pivot, we have found the element.
- If our element < pivot, remove rightmost column. Repeat algorithm on the submatrix.
- If our element > pivot, remove topmost row. Repeat algorithm on the submatrix.

Runtime Explanation

If we choose the upper right element as our pivot then worst case scenario is where the element is in the bottom left corner. In this case we would need to remove n rows and n columns, for a total of $2n$ operations. This means our solution is $O(n)$.
