Nathan Bickel
CSCE 350: Section H01
Professor: Dr. Narayanan
January 24, 2023

CSCE 350 Homework 1

Problem 1
(a) Assume you have an empty stack, show the stack after a sequence of operations: push(a), push(b), pop, push(c), push(d), pop, pop, push(e), pop.
(b) Assume you have an empty queue, show the queue after a sequence of operations: enqueue(a), enqueue(b), dequeue, enqueue(c), enqueue(d), dequeue, dequeue, enqueue(e), dequeue.

Solution.

(a) We show the stack at each step, with left to right going from bottom to top:

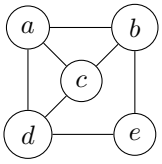
- (a)
- (a, b)
- (a)
- (a, c)
- (a, c, d)
- (a, c)
- (a)
- (a, e)
- (a)

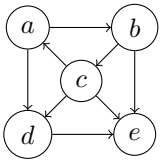
(b) We show the queue at each step, with left to right going from the front of the queue to the back:

- (a)
- (a, b)
- (b)
- (b, c)
- (b, c, d)
- (c, d)
- (d)
- (d, e)
- (e)

Problem 2 For the graphs in Figure 1,
(a) For both figures, write the adjacency matrix and the adjacency linked list.
(b) Is Figure (A) a complete graph? Why?

(c) For both figures, are there any loops in each graph? If so, write down the corresponding edges.
(d) For both figures, are there any cycles in each graph? If so, write down the corresponding paths.

(A) 

(B) 

Solution.

(a) Let $v_1 = a, \ldots, v_5 = e$. Then, for an adjacency matrix M, $(M)_{ij} = 1$ if $v_j v_i \in E(G)$ and 0 otherwise.

$$A = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix} \quad B = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

We can also write the adjacency linked list for A:

$$\{a : [b, c, d], b : [a, c, e], c : [a, b, d], d : [a, c, e], e : [b, d]\}$$

and for B:

$$\{a : [b, d], b : [c, e], c : [a, d, e], d : [e], e : []\}.$$

(b) No, because not every pair of vertices are neighbors, such as c and e.

(c) There are no loops, because no vertex connects to itself.

(d) Cycles in A:

- (a, b, c, a)
- (a, c, d, a)
- (a, b, c, d, a)
- (a, b, e, d, a)
- (b, c, d, e, b)
- (c, a, d, e, b, c)
- (c, d, e, b, a, c)
- (c, b, e, d, a, c)

The only cycle in B is (a, b, c, a).

Problem 3 How would you implement a dictionary of a reasonably small size n of you knew that all its elements are distinct (e.g., names of the countries in the world)? Specify an implementation of each dictionary operation.

Solution.

```
class Node
    // Contains value of the Node, whatever object it may be
    value
    // Points to the next node in the list, or Null if there isn't one
    next: Node
end class
// Input: Node head that is the first node in the list
// Input: Object value that is being searched for
// Output: Whether the list contains value
function SEARCH(head, value)
    current ← head
    while current.value ≠ value and current.next ≠ Null do
        current ← current.next
    end while
    return current.value = value
end function
// Input: Node head that is the first node in the list
// Input: Object value that is being added
// Output: Node with value added in
function ADD(head, value)
    newNode ← new Node(next = head, value = value)
    head ← newNode
    return head
end function
// Input: Node head that is the first node in the list
// Input: Object value that is being deleted
// Output: Node with value deleted
function DELETE(head, value)
    if head.value = value then
        return head.next
    else
        current ← head
        while current.next.value ≠ value do
            current ← current.next
            if current.next = Null then
                // We didn't find the value, so return the list
                return head
            end if
        end while
        // We found the value, so we need to skip over it
        current.next ← current.next.next
        return head
    end if
end function
```

Problem 4 Consider the following algorithm for finding the distance between the two closest elements in an array of numbers.
(a) What are the candidates of basic operations of this algorithm? For the worst case, how many times each of them are performed as a function of the array size n?
(b) Make as many improvements as you can in this algorithmic solution to the problem. You must write down the pseudocode for your new algorithm. What is the basic operation of your new algorithm? How many times is the basic operation performed as a function of the array size n? (If you need to, you may change the algorithm altogether.)

Algorithm 1 MinDistance($A[0 \dots n-1]$)

```
// Input: Array A[0...n-1] of numbers
// Output: Minimum distance between two of its elements
dmin ← ∞
for i ← 0 to n − 1 do
    for j ← 0 to n − 1 do
        if i ≠ j and √((A[i] − A[j])²) < dmin then
            dmin ← √((A[i] − A[j])²)
        end if
    end for
end for
return dmin
```

Solution.

(a) The candidates of basic operations are assignment, comparison, subtraction, and exponentiation. Because of the nested for loops, there are n^2 iterations. In the worst case, *dmin* is reassigned at each iteration, which will result in:

- 2 assignments per iteration (one for j and one for *dmin*), one assignment each time the outer loop changes, and one initial assignment for *dmin*, so $2n^2 + n + 1$ times.
- 2 comparisons per iteration (one for $i \neq j$ and one for $\sqrt{(A[i] - A[j])^2} < dmin$), so $2n^2$ times.
- 2 subtractions per iteration (one for checking if $\sqrt{(A[i] - A[j])^2} < dmin$ and one for $dmin \leftarrow \sqrt{(A[i] - A[j])^2}$), so $2n^2$ times.
- 4 exponentiations per iteration (two per time Euclidean distance is checked, for squaring the sum and then raising it to the $\frac{1}{2}$ power), so $4n^2$ times.

(b) We can improve the algorithm by taking the absolute value of the difference rather than squaring and then square rooting, which gets rid of all exponentiation. We can also store the value of the absolute value instead of calculating it twice. Finally, instead of looping through all possible i and j, we can have i run from 0 to $n-1$ and j run from $i+1$ to j. This way, $i \neq j$ always and this cuts the values being checked in half.

Using this method, assignment is the basic operation. In the worst case, it happens three times per iteration (once for j, once for d, and once for *dmin*). It also happens once each time the outer loop changes, and once at

the beginning for *dmin*. There will be $\binom{n}{2}$ iterations, so assignment will happen $\frac{3n(n-1)}{2}+n+1 = \frac{3}{2}n^2-\frac{1}{2}n+1$ times.

// Input: Array $A[0 \dots n-1]$ of numbers
// Output: Minimum distance between two of its elements
$dmin \leftarrow \infty$
for $i \leftarrow 0$ to $n-1$ **do**
    for $j \leftarrow i+1$ to $n-1$ **do**
        $d \leftarrow |A[i] - A[j]|$
        if $d < dmin$ **then**
            $dmin \leftarrow d$
        end if
    end for
end for
return *dmin*

Problem 5 We have introduced the Selection Sort as shown in the Algorithm 2 below. Is Selection Sort stable? Is it in place? Explain your assertion.

Algorithm 2 SelectionSort($A[0 \dots n-1]$)

// Sorts a given array by selection sort
// Input: Array $A[0 \dots n-1]$ of orderable elements
// Output: Array $A[0 \dots n-1]$ sorted in ascending order
for $i \leftarrow 0$ to $n-2$ **do**
    $min \leftarrow i$
    for $j \leftarrow i+1$ to $n-1$ **do**
        if $A[j] < A[min]$ **then**
            $min \leftarrow j$
            swap $A[i]$ and $A[min]$
        end if
    end for
end for

Solution.

Selection Sort is not stable. For example, take the list $(2_A, 3, 2_B, 1)$ (the subscripts are added for illustration). Then, the list will take the following states:

- $(1, 3, 2_B, 2_A)$
- $(1, 2_B, 3, 2_A)$
- $(1, 2_B, 2_A, 3)$.

As shown, the 2s do not stay in the same order. However, Selection Sort is in place, because no additional lists are created in memory to aid sorting.

Problem Bonus 1 Prove that a complete binary tree has the height of $h = \lfloor \log_2 n \rfloor$, where n is the total number of nodes in the tree.

Solution.

Let T be a complete binary tree on n vertices and height h. Then, define levels $L_0, L_1, \ldots, L_h$, where L_0 is the set containing the root and (for $0 < i \leq h$) L_i is the set containing the children of every vertex in L_{i-1}.

Since T is complete, we have $L_0 = 1$, $L_1 = 2$, $L_2 = 4$, and so on, except L_h does not need to be full as long as it is not empty. So for $0 \leq i < h$, $|L_i| = 2^i$, and $1 \leq |L_h| \leq 2^h$. Since every vertex will be in exactly one level, we have

$$\begin{aligned}
n &= 2^0 + 2^1 + 2^2 + \ldots + 2^{h-1} + |L_h| \\
&= \sum_{k=0}^{h-1} 2^k + |L_h| \\
&= 2^h - 1 + |L_h| && \text{(using finite geometric series formula)} \\
\implies |L_h| &= n - 2^h + 1 \\
\implies 1 &\leq n - 2^h + 1 \leq 2^h && \text{(since } 1 \leq |L_h| \leq 2^{h-1}\text{)} \\
\implies 2^h &\leq n \leq 2^{h+1} - 1 && \text{(algebra)} \\
\implies 2^h &\leq n < 2^{h+1} \\
\implies h &\leq \log_2 n < h + 1.
\end{aligned}$$

Since h is an integer, we must have $h = \lfloor \log_2 n \rfloor$. □