Theoretical Assignment 2

Kaushik Raj V Nadar
Roll No.: 200499
ESO207A: Data Structures and Algorithms
March 21, 2022

Question 1

Statement: It is asymptotically faster to square an n-bit integer than to multiply two n-bit integers. State whether the above statement is True or False. Prove your answer.

Solution

The Statement is **False**.
Multiplication of two n-bit integers can be written as follows:

$$a \times b = \frac{((a+b)^2 - a^2 - b^2)}{2}$$

Here, we are using addition, subtraction and division by 2, and squaring $(a+b)$, a, and b.
Addition and subtraction take $O(n)$ time, division by 2 can be done using the right shift operator in $O(1)$ time.
So, if square-computing algorithm has time complexity O(f(n)), our algorithm to multiply different numbers has complexity O(3*f(n) + 3*n+1), equal to O(f(n)).
Therefore, squaring an n-bit integer is asymptotically as hard as multiplying two n-bit integers.

Theoretical Assignment 2

Kaushik Raj V Nadar
Roll No.: 200499
ESO207A: Data Structures and Algorithms
March 21, 2022

Question 2

Given the array of integers A[1 . . . n] , where

$$M = \max A[i] - \min A[i]$$

(a) (8 points) Design an algorithm that can sort the array in $O(n + M)$ time
(b) (5 points) When M = O(n), sorting can be done in linear time: why doesn't the $\Omega(nlogn)$ lower bound apply in this case?

Solution

(a) Psuedocode :

```
void sort( int A[] ) :
    let max = -infinity, min = infinity

    let n = length of the array A

    for i from 0 to n-1:
        if(A[i]>max):
            max=A[i]
        if(A[i]<min):
            min=A[i]

    let range = max - min + 1

    int count[range]
    int result[n]

    for i from 0 to n-1:
        count[A[i] - min] = count[A[i] - min]+1

    for i from 1 to range-1:
        count[i] = count[i] + count[i - 1]

    for i from n-1 to 0 :
        final[count[A[i] - min] - 1] = A[i]
        count[A[i] - min] = count[A[i] - min] - 1

    for i from 0 to n-1 :
        A[i] = final[i]

sort(A)
```

(b) A sort has to distinguish among the $n!$ possible input permutations. Each comparison only yields 1 bit of information, but $n!$ has $log_2(n!) = \Omega(nlogn)$ bits of information, so you need $\Omega(nlogn)$ comparisons. Since, Counting sort is not a comparison sort; it uses key values as indexes into an array and the $\Omega(nlogn)$ lower bound for the sorting will not apply.

Theoretical Assignment 2

Kaushik Raj V Nadar
Roll No.: 200499
ESO207A: Data Structures and Algorithms
March 21, 2022

Question 3

Design a linear-time algorithm which, given an undirected graph G and a particular edge e in it, determines whether G has a cycle containing e. You also need to prove the correctness of your algorithm.

Solution

Let e be the edge connecting two vertices, say, u and v.
Take out that edge e(u,v) from G.
1. Run BFS to see if vertex v is still reachable from vertex u.
2. if yes, then the original graph G has a cycle containing e. otherwise there is not.

Proof :
If the edge e(u,v) is in a cycle, then there will be 2 possible ways to reach v from u. When the edge is removed, there will be one possible way remaining to reach v from u.

In case the edge is not in a cycle, then there will be only one possible way to reach vertex v from u. In this case, when we remove the edge e, there will be no possible way to reach vertex v from u.
We can verify whether there is a possible path to vertex v by running BFS(G,u).

All possible ways to traverse in graph G from vertex u is considered in BFS traversal.

Theoretical Assignment 2

Kaushik Raj V Nadar
Roll No.: 200499
ESO207A: Data Structures and Algorithms
March 21, 2022

Question 4

We have three containers whose capacities are m litres, n litres, and p litres respectively (assume $m > n > p$). The n-litre and p-litre containers start out full of water, but the m-litre container is initially empty. We are allowed one type of operation: pouring the contents of one container into another, stopping only when the source container is empty or the destination container is full. We want to find the smallest sequence of pourings (if it exists) that leaves exactly x litres in the n-litre or p-litre container where ($x \leq p$). If there are multiple smallest sequences, output any one of them.

(a) (10 points) Model this as a graph problem: give a precise definition of the graph involved (vertex set and edge set) and state the specific question about this graph that needs to be answered.
(b) (5 points) Design an $O(V + E)$ algorithm to solve this problem.
(c) (3 points) Let m = 10, n = 7, p = 4, x = 2, Find the smallest sequence using the above algorithm if it exists.

Solution

(a)

The problem can be modelled into a graph $G = (V, E)$ with the following definitions:
Vertex Set: V = Set of all vertices storing the state of the containers. The current volume of each container at every step is stored in the vertices.
Then the starting node is a list of three such objects:
$[\{volume : 0, max : 10\}, \{volume : 7, max : 7\}, \{volume : 4, max : 4\}]$
Edge Set: E = Set of all edges containing the action of pouring water from one container into another. Since the action of pouring is irreversible, G will be a directed graph.

To find the smallest sequence of pourings that leaves exactly x litres in the n-litre or p-litre container where ($x \leq p$), we need to find the shortest path in the graph G from the initial state of the containers to the state where there are exactly x litres in the n-litre or p-litre container.

(b)

By performing the Breadth First Search over the graph we can obtain the shortest distance of required state from initial state. Hence, the problem can be solved in O(V+E) time using BFS search.

```
BFS(graph, start_node, end_node):
    frontier = new Queue()
    frontier.enqueue(start_node)
    explored = new Set()

    while frontier is not empty:
        current_node = frontier.dequeue()
        if current_node in explored: continue
        if current_node == end_node: return success

        for neighbor in graph.get_neigbhors(current_node):
            frontier.enqueue(neighbor)
        explored.add(current_node)
```

(c)

Following is the shortest sequence for m = 10, n = 7, p = 4, x = 2 :

```
volumes: 0/10, 7/7, 4/4
pour from glass 3 to glass 1

volumes: 4/10, 7/7, 0/4
pour from glass 2 to glass 1

volumes: 10/10, 1/7, 0/4
pour from glass 1 to glass 3

volumes: 6/10, 1/7, 4/4
pour from glass 3 to glass 2

volumes: 6/10, 5/7, 0/4
pour from glass 1 to glass 3

volumes: 2/10, 5/7, 4/4
pour from glass 3 to glass 2

volumes: 2/10, 7/7, 2/4
```

Theoretical Assignment 2

Kaushik Raj V Nadar
Roll No.: 200499
ESO207A: Data Structures and Algorithms
March 21, 2022

Question 5

An $m \times n$ Young tableau is an $m \times n$ matrix such that the entries of each row are in sorted order from left to right and the entries of each column are in sorted order from top to bottom. Some of the entries may be ∞ that are treated as non-existent items. Thus a Young tableau can be used to hold $r \leq mn$ finite numbers. Note that an mn tableau Y is empty if $Y[1,1] = \infty$ and is full if $Y[m,n] < \infty$.
An example Young tableau is

$$\begin{bmatrix} 2 & 4 & 8 & 12 \\ 3 & 5 & 9 & \infty \\ 12 & 14 & 16 & \infty \end{bmatrix}$$

(a) (5 points) Give an algorithm to extract minimum element from a non-empty $m \times n$ Young tableau that runs in time $O(m+n)$ such that after removal of that element, the resulting matrix still remains a Young tableau.
(b) (5 points) Show how to insert a new element into a non-full $m \times n$ Young tableau in $O(m+n)$ time.
(c) (5 points) Using no other sorting method as a subroutine, show how to use an n×n Young tableau to sort given n^2 numbers in $O(n^3)$ time.
(d) (7 points) Given an $O(m+n)$ time algorithm to determine whether a given number is present in a given $m \times n$ tableau.

Solution

(a) Pseudocode:

```
func fixTableau(tableau, i=0, j=0):

    # get the values present at the bottom and right cell of the current cell
    bottom = tableau[i + 1][j] if (i + 1 < M) else infinity
    right = tableau[i][j + 1] if (j + 1 < N) else infinity

    if bottom = infinity  and right = infinity :
        return

    if bottom < right:      # go down

        # swap tableau[i][j] with tableau[i + 1][j]
        temp = tableau[i][j]
        tableau[i][j] = tableau[i + 1][j]
        tableau[i + 1][j] = temp

        fixTableau(tableau, i + 1, j)

    else:                   # go right

        # swap tableau[i][j] with tableau[i][j + 1]
        temp = tableau[i][j]
        tableau[i][j] = tableau[i][j + 1]
        tableau[i][j + 1] = temp

        fixTableau(tableau, i, j + 1)


# Function to extract the next minimum element from the Young tableau
func extractMin(tableau):

    # base case
    if not tableau:
        exit

    # the first cell of the tableau stores the minimum element
    min = tableau[0][0]

    # make the first element as infinity
    tableau[0][0] = infinity

    # fix the Young tableau property
    fixTableau(tableau)

    return min
```

The fixTableau() routine takes an M × N matrix and breaks it into either an (M-1) × N or an M × (N-1) matrix. So, the recurrence relation is:

$$T(M, N) = T(M - 1, N) + T(M, N - 1)$$

So, the overall time complexity of the Extract-Min operation on a Young tableau is O(M + N).

(b)
To insert an element into a non-full Young tableau, the idea is to place the element in the bottom-right corner of the tableau and then move it upwards and leftwards to its correct position within the tableau.
Pseudocode:

```
func insert(tableau, i, j):

    # base case
    if i == 0 and j == 0:
        return

    # handle separately for the first row
    if i == 0:
        if tableau[i][j] < tableau[i][j - 1]:
            # swap tableau[i][j] and tableau[i][j-1]
            temp = tableau[i][j]
            tableau[i][j] = tableau[i][j - 1]
            tableau[i][j - 1] = temp

            insert(tableau, i, j - 1)
        return

    # handle separately for the first column
    if j == 0:
        if tableau[i][j] < tableau[i - 1][j]:
            # swap tableau[i][j] and tableau[i-1][j]

            temp = tableau[i][j]
            tableau[i][j] = tableau[i - 1][j]
            tableau[i - 1][j] = temp

            insert(tableau, i - 1, j)
        return

    if tableau[i][j] < tableau[i - 1][j]:        # go up

        # swap tableau[i][j] and tableau[i-1][j]
        temp = tableau[i][j]
        tableau[i][j] = tableau[i - 1][j]
        tableau[i - 1][j] = temp

        insert(tableau, i - 1, j)

    if tableau[i][j] < tableau[i][j - 1]:        # go left

        # swap tableau[i][j] and tableau[i][j-1]
        temp = tableau[i][j]
        tableau[i][j] = tableau[i][j - 1]
        tableau[i][j - 1] = temp

        insert(tableau, i, j - 1)
```

```
n = element to be inserted
# place the element in the bottom-right corner of the tableau
tableau[M - 1][N - 1] = n
# move the element to its correct position in the tableau
insert(tableau, M - 1, N - 1)
```

(c)

To sort an array using Young tableau, insert each of its values into an empty Young tableau, one at a time. Afterward, repeatedly call the Extract-Min routine on the Young tableau until the tableau is empty and put the returned values back to the original array.

Pseudocode:

```
// Function construct an ‘N × N‘ Young tableau from the given keys
func construct(int keys[])
{
    // initialize the Young tableau by infinity
    int N = (int) ceil(sqrt(keys.size()))
    tableau(N, (N, INT_MAX))

    // do for each key
    for each key in keys:
    {
        // check for overflow
        if (tableau[N - 1][N - 1] != INT_MAX):
            break

        // place the key at the bottom-right corner of the tableau
        tableau[N - 1][N - 1] = key

        // move the key to its correct position in the tableau
        insert(tableau, N - 1, N - 1)
    }
    return tableau
}

// Function to extract the next minimum element from the Young tableau
int extractMin(tableau[][])
{
    // the first cell of the tableau stores the minimum element
    int min = tableau[0][0]

    // make the first element as infinity
    tableau[0][0] = infinity

    // fix the Young tableau property
    fixTableau(tableau, 0, 0)

    return min
}

void sort(int keys[])
{
    if (keys.size() = 0):
```

```
            return

        // construct a Young tableau from the above keys
        YoungTableau s
        let tableau[][] = s.construct(keys)

        // repeatedly call ‘extractMin()‘ and fill ‘keys[]‘ with the returned values
        for i from 0 to keys.size()-1 :
            keys[i] = s.extractMin(tableau)
    }

let arr = input array
sort(arr)
```

(d)

We start from the tableau's top-rightmost corner and compare the current element with the one we are looking for and accordingly move left or down:
1. If the current element is less than the key, increment the row index (move to the next row)
2. If the current element is more than the key, decrement the column index (move to the previous column)
3. If the current element is equal to the key, return true as we have found the element. If we end up in a row or column beyond the tableau boundary, terminate the search and return false.

Pseudocode:

```
func search(tableau, key):

    # base case
    if not tableau:
        return False

    # start from the top-rightmost cell of the matrix, i.e., (0, N-1) cell
    i = 0
    j = len(tableau[0]) - 1

    # run till matrix boundary is reached
    while i < len(tableau) and j >= 0:

        # if the current element is less than the key, increment the row index
        if tableau[i][j] < key:
            i = i + 1

        # if the current element is more than the key, decrement the column index
        else:
            if tableau[i][j] > key:
                j = j - 1

            # if the current element is equal to the key
            else:
                return True

    return False
```

Theoretical Assignment 2

Kaushik Raj V Nadar
Roll No.: 200499
ESO207A: Data Structures and Algorithms
March 21, 2022

Question 6

Given an undirected tree of n vertices, where each vertices are colored white or black or grey. The tree contains at least one white vertex and at least one black vertex. We pick an edge e and removing edge e from the tree will break the tree into two connected components. An edge is called 'pure' if neither of the resulting components contain vertices of both black and white color.

(a) (20 points) Design an $O(n)$ time algorithm to output all 'pure' edges in the tree. (Giving $O(n^2)$ time algorithm will only fetch a maximum of 5 marks.)
(b) (5 points) Analyze the time complexity of your algorithm and give the proof of correctness

Solution

(a) Algorithm :
(b)
Time Complexity: $O(n)$