DAA ASSIGNMENT 6
Group No 19

AYUSH KHANDELWAL (IIT2019240)

AYUSH BHAGTA (IIT2019501)

TAUHID ALAM (BIM2015003)

Contents -

Problem Statement

You are given a m liter jug and a n liter jug. Both the jugs are initially empty. The jugs don't have markings to allow measuring smaller quantities. You have to use the jugs to measure d liters of water where d is less than n. (X, Y) corresponds to a state where X refers to amount of water in Jug1 and Y refers to amount of water in Jug2

Determine the path from initial state (xi, yi) to final state (xf, yf), where (xi, yi) is (0, 0) which indicates both Jugs are initially empty and (xf, yf) indicates a state which could be

(0, d) or (d, 0).

The operations you can perform are:

1. Empty a Jug, (X, Y)->(0, Y) Empty Jug 1

2. Fill a Jug, (0, 0)->(X, 0) Fill Jug 1

3. Pour water from one jug to the other until one of the jugs is either empty or

full, (X, Y) -> (X-d, Y+d)

Introduction

Given a m litre jug and a n litre jug which are initially empty. The jugs don't have markings to allow measuring smaller quantities. We have to use the jugs to measure d litres of water where d < n and d < m.

(X, Y) corresponds to a state where X refers to amount of water in Jug1 and Y refers to amount of water in Jug2. Determine the path from initial state (xi, yi) to final state (xf, yf), where (xi, yi) is (0, 0) which indicates both Jugs are initially empty and (xf, yf) indicates a state which could be (0, d) or (d, 0).

The operations you can perform are:

- Empty a Jug, (X, Y)− >(0, Y) Empty Jug 1
- Fill a Jug, (0, 0)->(X, 0) Fill Jug 1
- Pour water from one jug to the other until one of the jugs

is either empty or full, (X, Y) -> (X-d, Y+d)

Breadth-first search is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root, and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level. We run breadth first search on the states and these states will be created after applying allowed operations and we also use visited map of pair to keep track of states that should be visited only once in the search. This solution can also be achieved using depth first search.

ALGORITHMIC DESIGN

Brute Force :

The associated Diophantine equation of the problem is given by $mx + ny = d$, whose solution is described by the theorem below.

Theorem. The Diophantine equation $mx + ny = d$ is solvable if and only if $\gcd(m, n)$ divides d. For convenience, let us assume $mx+ny = d$ is solvable in the discussions below. Depending

on which jug is chosen to be filled first, there are two possible solutions for solving the two water jugs problems. They are labelled by M1 and M2 in the following algorithms:

Algorithm.

Input: The integers m, n and d, where $0<m<n$ and $d<n$.

Output: An integer sequence corresponding to a feasible

solution (called M1) of the two water jugs problem, by filling the m-litre jug first.

Procedure: Step 1. Initialize a dummy variable $k = 0$.

Step 2. If $k < d$, then repeat adding m to k and assign the result to k until $k = d$ or $k > n$.

Step 3. If $k > n$, then subtract n from k and assign the result to k.

Step 4. If $k = d$, then stop. Otherwise, repeat the steps from Step 2 to Step 4. The number of additions (say x1) and subtractions (say y1) involved provides a solution to the Diophantine equation $mx+ny=d$, namely $x = x1$, $y = -y1$. The actual pouring sequence can be determined by referring to the integer sequence obtained.

Using Graph

- Step 1. Initialize an empty pair of string containing [0,0] that is the initial state of the jugs. This string contain path for a particular state to be achieved.
- Step 2. We Initialize an empty deque(Doubly Ended Queue) and push the first path that is [[0,0]] to it.
- Step 3. We check if last state of the the left most path of the deque is the required path or not and exit the loop and save that path in the final path variable and move to Step6 if the condition satisfies. Otherwise continue to Step 4.
- Step 4. We look for all the possible cases from the last state of the first path that is in the queue and remove that path and further add all the possible paths to the queue to left for the DFS(Depth First Search) approach or to the right for BFS(Breadth First Search) approach. :
- Step 5.We go back to step 3 and continue the iteration until no further transition is possible that is the given condition could not be satisfied.
- Step 6. We print the final path variable in the order of transitions made to reach the condition.

Illustration

Brute-Force

Volume of first jug: 3

Volume of second jug: 4

Desired volume: 2

k=0

Repeat adding 3 to k until k=2 or k¿4

k=3+3=6

Now subtract 4 from k

k=6-4=2

As k= desired volume so we stop here.

The actual pouring sequence can be determined by referring to the integer sequence obtained. [0,0][0,3][3,0][3,3][2,4]

Using Graph:

BFS:

Volume of first jug: 5 Volume of second jug: 4 Desired volume: 2

- First we append 0,0 to our path path=[0,0]
- Now next possible transitions from [0,0] are [5,0] [0,4] next=[0,4],[5,0]
- Now next possible transitions from [5,0] are [5,4] [1,4] next=[0,4],[5,4],[1,4]
- Now next possible transitions from [0,4] are [5,4] [4,0] next=[5,4],[1,4],[5,4],[4,0]
- Now next possible transitions from [5,4] are [no more transitions]: next=[1,4],[5,4],[4,0]
- Now next possible transitions from [1,4] are [1,0]: next=[5,4],[4,0],[1,0]
- Now next possible transitions from [5,4] are [no more transitions]: next=[4,0],[1,0]
- Now next possible transitions from [4,0] are [4,4]: next=[1,0],[4,4]

- Now next possible transitions from [1,0] are [0,1]: next=[4,4],[0,1]
- Now next possible transitions from [4,4] are [5,3]: next=[0,1],[5,3]
- Now next possible transitions from [0,1] are [5,1]: next=[5,3],[5,1]
- Now next possible transitions from [5,3] are [0,3]: next=[5,1],[0,3]
- Now next possible transitions from [5,1] are [2,4]: next=[0,3],[2,4]
- Now next possible transitions from [0,3] are [3,0]: next=[2,4][3,0]
- Now next possible transitions from [2,4] are [2,0]: Goal is achieved as we have [2,0] as a state Now path=[0,0] [5,0] [1,4] [1,0] [0,1] [5,1] [2,4]

DFS:

Volume of first jug: 4 Volume of second jug: 3 Desired volume: 2

- First we append 0,0 to our path path=[0,0]
- Now next possible transitions from [0,0] are [4,0] [0,3] next=[0,3],[4,0]
- Now next possible transitions from [0,3] are [4,3] [3,0] next=[3,0],[4,3],[4,0]
- Now next possible transitions from [3,0] are [4,0] [3,3] next=[3,3],[4,0],[4,3][4,0]
- Now next possible transitions from [3,3] are [4,3] [4,2] next=[4,2],[4,3],[4,0],[4,3][4,0]
- Now next possible transitions from [4,2] are [0,2] Goal achieved as we have 2 litres in one jug Now Path=[0,0],[0,3],[3,0],[3,3][4,2]

Time complexity

Using Graph:

In this approach we iterate over whole map generated by the nodes containing all the possible amount of water in both the jugs until we find the any of teh one required condition. So to iterate over a graph by BFS or by DFS time complexity will be as below:

Best Case-If the given condition is not true or the required amount of water is equal to capacity of one of the jug.This will result in the time complexity of constant order.So Best

Case Time Complexity= Ω(1)

Worst Case-If the given condition is true and the required amount of waterCase Time Complexity= Ω(1) Worst Case-If the given condition is true and the required amount of water is what we get at the farthest end.This will result in the time complexity of order O(V + E) where E

ranges from 1 to V 2, but in our case number of edges is equal to six hence number of edges will be of order 3V , so,

the time complexity will be of order 4V, which is treated as

linear in time complexity, where V is the number of nodes and E is the number of Edges. Maximum value of V in

this case could be N*M where N and M are the maximum

capacities of mug.So Worst Case Time Complexity=O(V)=

O(N ∗ M) is what we get at the farthest end.This will

result in the time complexity of order O(V + E) where E

ranges from 1 to V^2

, but in our case number of edges is equal to six hence number of edges will be of order 3V

, so, the time complexity will be of order 4V, which is treated as linear in time complexity,

where V is the number of nodes and E is the number of Edges. Maximum value of V in this

case could be N*M where N and M are the maximum capacities of mug.So Worst Case

Time Complexity=O(V)= O(N $*$ M)

Space complexity

Brute-Force:

In Brute force we will be directly adding and subtracting in an integer so no extra space is required .

This will result in the space complexity of O(1).

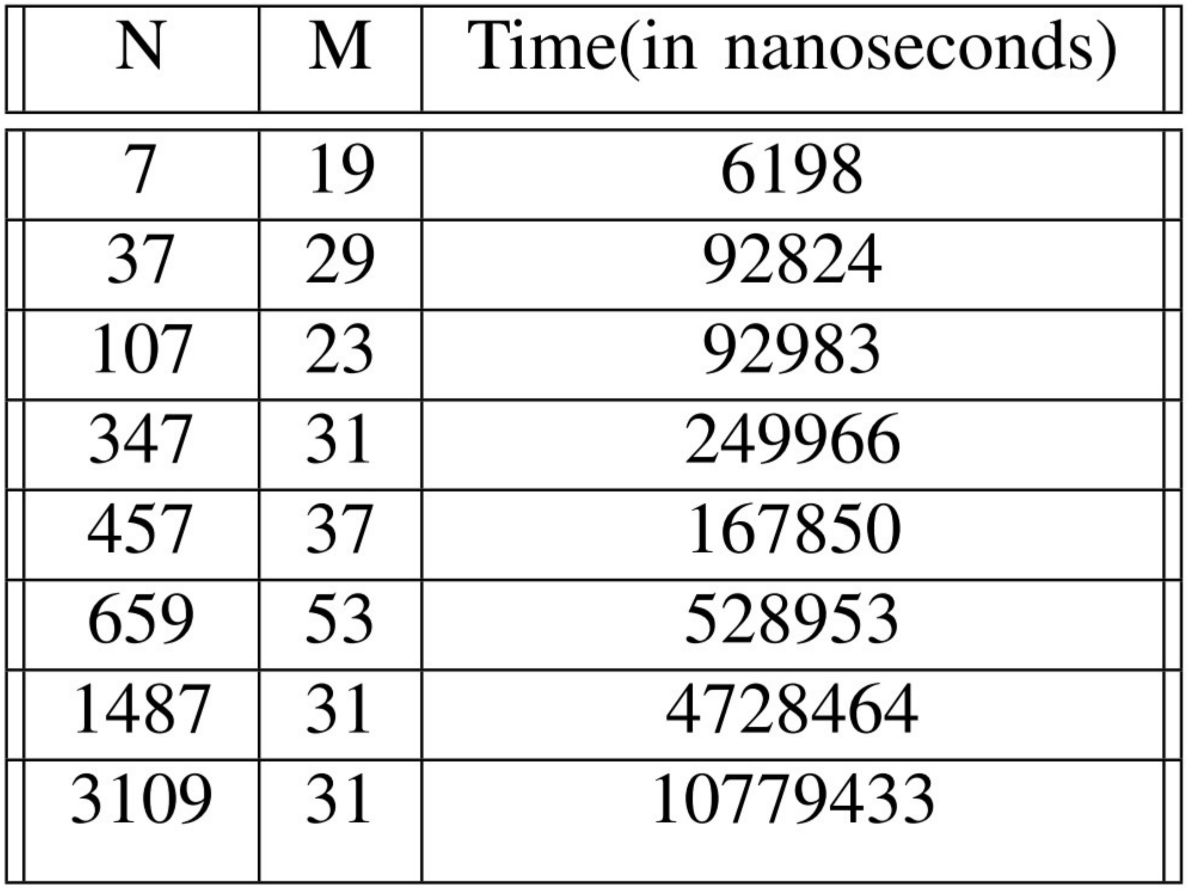
Using Graph:

In this approach we don't have need to store any graph as all the path are logically decided and checked at the time, so no more Extra space is required to store the map.But we store the path that is used to reach that point which whose length vary from 1 to V and we need to store path for all V vertices and hence the required space is of order V 2.

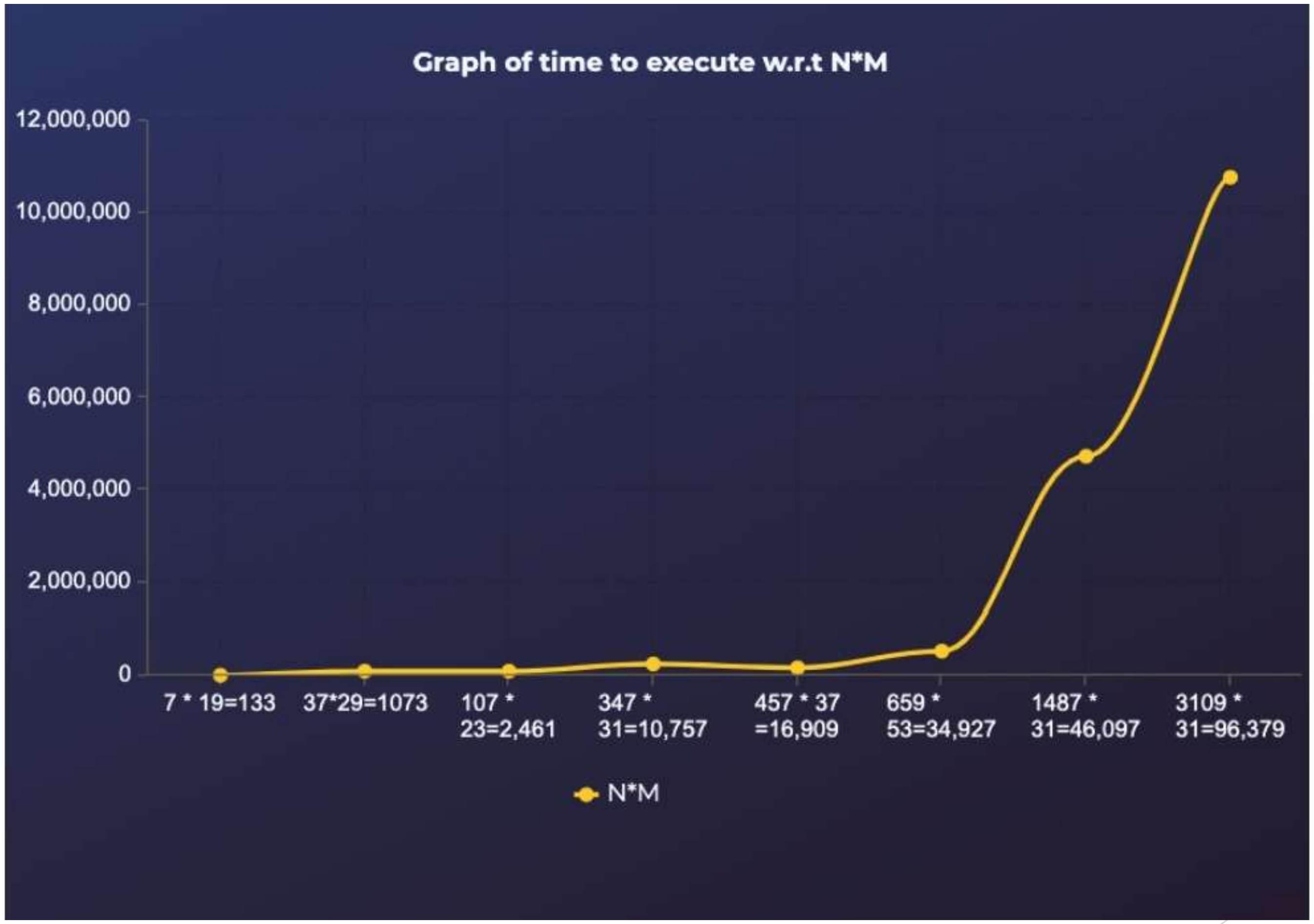
Maximum value of V in this case could be N*M where N and M are the maximum capacities of mug.So Space Complexity= O((N ∗ M)2)

This will result in the space complexity of O(n ∗ m).

Space complexity:



N	M	Time(in nanoseconds)
7	19	6198
37	29	92824
107	23	92983
347	31	249966
457	37	167850
659	53	528953
1487	31	4728464
3109	31	10779433


Graph of time to execute w.r.t N*M
12,000,000
10,000,000
8,000,000
6,000,000
4,000,000
2,000,000
0
7 * 19=133
37*29=1073
107 * 23=2,461
347 * 31=10,757
457 * 37 =16,909
659 * 53=34,927
1487 * 31=46,097
3109 * 31=96,379
N*M

Conclusion

We can observe that in graph the space required is more but it is much more efficient in terms of time and will be favourable for the values having bigger differences in the capacity of jugs.

References

1) https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/

2) https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/

3) https://www.eecis.udel.edu/~mccoy/courses/cisc4-681.10f/lecmaterials/handouts/search-water-jug-handout.pdf

4) https://en.wikipedia.org/wiki/Breadth-first search

5) https://en.wikipedia.org/wiki/Depth-first search

6) R. S. Mary, “An alternative arithmetic approach to the

water jugs problem,” Proceedings on National Con- ference on Computational Intelligence for Engineering

Quality Software, 1, 2014, pp. 10-13.

7) S. Abu Naser, “Developing visualization tool for the

teaching AI searching algorithms,” Information Technol-

ogy Journal, 7(2), 2008, pp. 350–355.