DAA ASSIGNMENT 6
Group No 19

AYUSH KHANDELWAL (IIT2019240)

AYUSH BHAGTA (IIT2019501)

TAUHID ALAM (BIM2015003)

Contents -

- Problem Statement
- Introduction
- Proposed Algorithm
- Illustration
- Time and space complexity analysis
- Conclusion
- References

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 liter jug and a n liter 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 liters
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.

Algorithm 2.2.

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

Output: An integer sequence corresponding to a feasible

solution (called M2) of the two water jugs problem, by filling the n-litre jug first.

Procedure:

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

Step 2. If $k = d$, then add n to k and assign the result to k.

Step 3. If $k > d$, then repeat subtracting m from k and assign the result

the result to k until $k = d$ or $k < m$.

Step 4. If $k = d$, then stop. Otherwise, repeat the steps from Step 2 to Step 4. The number of subtractions (say x2)

and additions (say y2) involved provides a solution to the Diophantine equation mx+ny=d, namely x = -x2, y = y2. The actual pouring sequence can be determined by referring to the integer sequence obtained.to k until k = d or k < m.

Step 4. If k = d, then stop. Otherwise, repeat the steps from Step 2 to Step 4. The number of subtractions (say x2) and additions (say y2) involved provides a solution to the Diophantine equation mx+ny=d, namely x = -x2, y = y2. The actual pouring sequence can be determined by referring to the integer sequence obtained

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]

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= $\Omega(1)$

Worst Case-If the given condition is true and the required amount of waterCase Time Complexity= $\Omega(1)$ 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
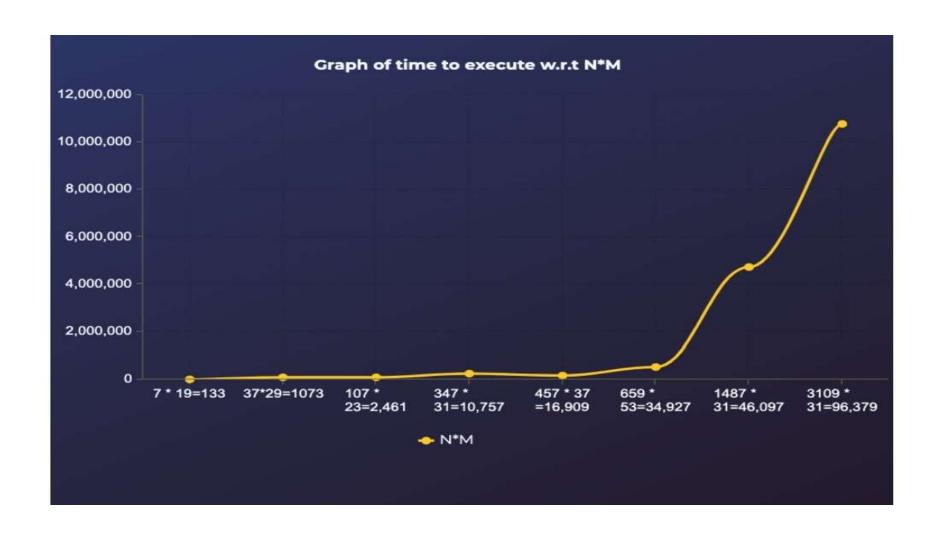and E is the number of Edges. Maximum value of V in
this case could be N*M where N and M are the maximum
capacities of mug.
So Worst Case Time Complexity=O(V)=O(N * M)

Time 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

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)

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.