HQ 15

// given 3 beakers b_1, b_2, b_3 with integral capacity of A, B, C respectively.

We will construct the graph $G(V, E)$ as follows →

Defining vertices, V

each vertex have 3 values representing the capacity of each beaker denoted by $b_1.c$, $b_2.c$, $b_3.c$.

Number of vertices

The capacity of one beaker can be minimum 0 to maximum its given max-capacity, thus in the worst case we must consider all such little values from 0 to max-capacity. So number of vertices = $A \times B \times C$.

Defining edges, E

Here edges will represent one of the three actions: fill, empty and pour-to. The edges will be unweighted and directed.

The following if conditions will represent when to add an edge from u vertex to v vertex in graph G for the three ~~of~~ actions: fill, empty, pour-to

From each vertex u we will have an edge to a new vertex v if the conditions below are satisfied.

Every if condition satisfied will result in a new vertex v from u.

conditions for fill action :

// b1 : if $u.b1.c < A$

$u \xrightarrow{G} v$ [denoting an edge in graph G from vertex u to v]

such that,

if $u.b1.c = x$

then $v.b1.c = x + (A - u.b1.c)$

// filling remaining capacity of beaker b_1 to its maximum capacity.

// rest of the values in v remain same as in u

// b2 : if $u.b2.c < B$

$u \xrightarrow{G} v$

such that,

if $u.b2.c = x$

then $v.b2.c = x + (B - u.b2.c)$

// filling remaining capacity of beaker b2 to its maximum capacity.

// rest of the values in v are same as in u

// b3 : if $u.b3.c < C$

$u \xrightarrow{G} v$

such that,

if $u.b3.c = x$

then $v.b3.c = x + (C - u.b3.c)$

// filling remaining capacity of beaker b_3 with its maximum capacity.

// rest of the values in v are same as in u

conditions for empty action :

// for b1 : if (v.b1.c != 0)

$$U \xrightarrow[q]{} V$$

such that v.b1.c = 0

// emptying contents of beaker b1

// rest values remain same as in U

// for b2: if (v.b2.c != 0)

$$U \xrightarrow[q]{} V$$

such that v.b2.c = 0

// emptying contents of beaker b2

// rest values remain same as in U

// for b3: if (v.b3.c != 0)

$$U \xrightarrow[q]{} V$$

such that v.b3.c = 0

// emptying contents of beaker b3

// rest values remain same as in U

conditions for pour-to action :

// b1 poured-to b2

if (v.b1.c != 0 && v.b2.c < 8)

$$U \xrightarrow[q]{} V$$

// b1 can pour to b2 if b1 is not empty and b2 has capacity to be filled

~~such that,~~ ~~v.b1.c = 0 &&~~

~~v.b2.c = v.b2.c +~~ (

such that, l = 3 - u.b2.c // remaining capacity of b2

cap = min (l, u.b1.c) // quantity to be filled in b2 from b1

// either entire b1 is poured to b2, handled by if condition or ~~part of~~ b2 is completely filled, handled by else condition. In any of the above cases we stop pouring.

if (u.b1.c < l)

v. b1.c = 0

v. b2.c = cap.

else

v. b1.c = u.b1.c - l.

v. b2.c = cap.

// b1 poured to b3.

if (u.b1.c != 0 && u.b3.c < c) // pouring contents of b1 to b3 if ~~either~~ b1 is ~~one~~ not empty and b3 has capacity to be filled

u ⟶ v

such that,

l = c - u.b3.c // remaing contents of b3

cap = min (l, u.b1.c)

if (u.b1.c < l) // entire quantity of b1 filled to b3

v.b1.c = 0

v.b2.c = cap

else // here b3 is full so filling ~~to~~ upto till b3's capacity only

v.b1.c = u.b1.c - l

v.b3.c = cap.

now similarly, b_2 pour-to b_1, b_2-pour-to b_3, b_3-pour-to-b_1 and b_3-pour-to-b_2 can be filled.

Number of edges

On the worst case each vertex can have 12 outgoing edges from it. For each beaker we have one fill state. For each beaker we have one empty state. For each beaker we have 2 pour-to states. Thus 12 outgoing edges; and we have $A \times B \times C$ vertices, so in worst case we will have $12 \times A \times B \times C$ edges.

Graph Problem to solve on $G(V, E)$

We need to solve shortest path from source vertex $(0,0,0)$ to ~~either~~ & any one of the 3 target vertices: $(-, t, -)$ or $(-, -, t)$ or $(t, -, -)$ where $(0,0,0)$ represents all beakers in the initial empty state & the target source represents any one beaker ~~for~~ reaching a capacity of t. $-$, value can be anything.

Algorithm

We can use BFS to solve the problem mentioned above. Also BFS will ensure that one vertex / state of the contents of the beakers visited once will not be visited again. This way we can avoid duplicate / multiple same computation

We can construct the graph ~~on~~ at runtime only. Because from each state, the possible ~~number of~~ actions can be determined using the conditions.

Time complexity

BFS for an undirected graph from source to target takes $O(V+E)$.

mapping the above complexity with our solution —

$|V| = A \times B \times C$

$|E| = 12 \times A \times B \times C$

so time complexity of above problem $= O(A \times B \times C)$