Worth: 8% **Due:** By 5:59pm on Tuesday 17 November

Remember to write the *full name* and *student number* of *every group member* prominently on your submission.

---

*Please read and understand the policy on Collaboration given on the Course Information Sheet. Then, to protect yourself, list on the front of your submission **every** source of information you used to complete this homework (other than your own lecture and tutorial notes). For example, indicate clearly the **name** of every student from another group with whom you had discussions, the **title and sections** of every textbook you consulted (including the course textbook), the **source** of every web document you used (including documents from the course webpage), etc.*

*For each question, please write up detailed answers carefully. Make sure that you use notation and terminology correctly, and that you explain and justify what you are doing. Marks **will** be deducted for incorrect or ambiguous use of notation and terminology, and for making incorrect, unjustified, ambiguous, or vague claims in your solutions.*

---

1. In this question, we will use a graph algorithm to solve a problem in philosophy.

   A *paradox* is a group of statements that lead to a contradiction. For example, the following two statements form a famous paradox:

   ```
   1. Statement 2 is FALSE.
   2. Statement 1 is TRUE.
   ```

   If we assume Statement 1 is TRUE, then Statement 2 is FALSE, which in turn means Statement 1 is FALSE. But if we assume Statement 1 is FALSE, then Statement 2 is TRUE, which in turn means Statement 1 is TRUE. So Statement 1 is TRUE iff Statement 1 is FALSE, a blatant contradiction.

   Now, suppose you are given a group of N statements, numbered from 1 to N. Each statement has the form: "`Statement X is TRUE/FALSE`," where `X` is a number between 1 and N. Your task is to figure out whether this group of statements forms a paradox. In particular, answer the following questions.

   (a) Describe how to construct a graph to solve this problem. Be precise: state clearly what vertices your graph contains (and what each vertex represents) and what edges your graph contains (and what each edge represents).

   (b) Give a **necessary and sufficient** condition for the N statements to form a paradox. Justify your answer.

   (c) How do you efficiently detect whether your graph from part (a) satisfies your condition described in part (b)? Describe your algorithm in concise (but precise) English.

   (d) Analyse the worst-case runtime of your algorithm.

   Answer

   (a) construct graph $G(V, E)$

   V: vertices set V contains all N statement. We order $V = \{v_1, v_2, v_3 ... v_N\}$ such that v_i is a i_{th} statement.

   E: There is edge $v_i\vec{v}_j$ with weight 1 iff i_{th} statement says j_{th} statement is true. There is edge $v_i\vec{v}_j$ with weight -1 iff i_{th} statement says j_{th} statement is false.

(b) Suppose $G(V,E)$ is the graph constructed from N statements. let's define the truth assignment τ for this $G(V,E)$.

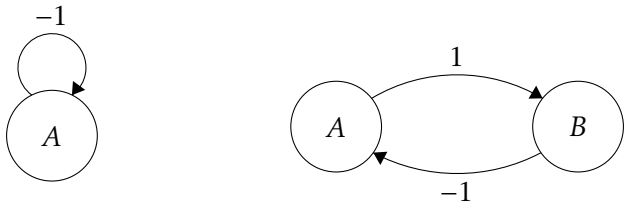
$$\tau : V \to \{1,-1\}, \quad where\ 1\ stands\ for\ true\ and\ -1\ for\ false$$

Now lets define STABLE for a truth assignment τ, a truth assignment τ is STABLE iff for any edge $v_i\vec{v}_j \in E$, $\tau(v_i) \times \tau(v_j) = Weight(v_i\vec{v}_j)$.

Then let's define random assignment Φ, random assignment Φ is a truth assignment which defined recursively as follows,

Pick any vertex a in graph $G(V,E)$ and assign 1 to a.
for any node $v \in V$, $\forall v' \in Neighbours(A)$,
if $Weight(v\vec{v'}) = 1$ or $Weight(v'\vec{v}) = 1$, then $\Phi(v') = \Phi(v)$;
if $Weight(v\vec{v'}) = -1$ or $Weight(v'\vec{v}) = -1$, then $\Phi(v') = -\Phi(v)$.

It's is obvious that random assignment Φ is stable, it can be derived from the definition of stable and random assignment.
Note: not all graph has a random assignment Φ (see following graphs).



For the left graph is we assign 1 to A, then based on the weight of edge A should be assigned -1 hence, contradiction then no random assignment. Same for the right hand side graph.

Claim : N statements do not form a paradox iff there exist a random assignment Φ for the corresponding graph $G(V,E)$.

Here we only consider the case when $G(V,E)$ is connected. Since if $G(V,E)$ is not connected we can just check random assignment Φ on each of its connected components. Therefore, it's sufficient to check just for connected graph. Proof

- ($\Leftarrow$) This direction is easy to prove, since if a graph has a stable random assignment Φ then just assign true and false to each statement corresponding with their truth value then clearly it's stable then no paradox.
- ($\Rightarrow$) Assume those N statements do not form a paradox, which means if we assign *true* to any $a \in V$. We can get the truth assignment for all $a's$ neighbours(A). Basically, if $v \in Neighbours(A)$ then v gets *true* iff a_{th} statement says v_{th} is true or v_{th} statement says a_{th} true and v gets value *false* iff a_{th} statement says v_{th} is false or v_{th} statement says a_{th} false. Recursively we can get value of all vertices in $G(V,E)$, clearly if we assign -1 to false and 1 to true this is exactly a random assignment Φ for $G(V,E)$.

(c) From (b) we can get N statements form a paradox iff there do not exist random assignment Φ for the corresponding graph $G(V,E)$, which means starts with any vertex a with value 1, we could not finish this assignment and we will always encounter contradiction no matter which initially node we choose.

Hence, our algorithm would simple be do a DFS on each of connected components of $G(V,E)$ and keep tracking truth value of every node. Precisely, every node has list, which initially is empty, then when we do DFS on a connected component, we starts with pick any nodes a add 1 to $a's$ list. And updating other nodes' list based on the recursively definition for random assignment and the last add value stands for the truth value for any node. When we finish the DFS we, do another DFS to check if any node has list with different value in it. If does, means there do not exist random assignment Φ for the corresponding graph $G(V,E)$; hence, we can get N statements form a paradox . otherwise we can get N statements do not form a paradox.

(d) Based on algorithm, the worst case runtime is same as worst time runtime for DFS which is $\mathcal{O}(|V|+|E|)$ we know that $|V| = N$ also every state only contribute 1 edge then $|E| = N$ as well. Therefore, worst time runtime is $\mathcal{O}(N)$

2. In this question, you will use a graph algorithm to solve the general form of a well-known brain-teaser.

You are given two **initially empty** buckets A and B, with capacities of m litres and n litres, respectively. Your goal is to measure exactly k litres of water using these two buckets. Assume that m, n and k are positive **integers** and that $k \leqslant \max(m,n)$. You want to achieve this goal by performing a sequence of **moves** until one of the buckets has exactly k litres of water. Each move can be one of the following:

- Fill a bucket until it's full.
- Empty a bucket.
- Use the water in one bucket to fill the other until one of the buckets is full or empty.

Now, you must devise an algorithm BUCKETMEASURE(m,n,k), which takes m, n and k as inputs and outputs a sequence of moves that results in one bucket (it does not matter which one) having exactly k litres of water. The number of moves in the returned sequence must be the **smallest possible** number of moves that are needed to achieve the goal. If it is impossible to measure k litres of water using the two buckets, the algorithm returns NIL.

Answer the following questions.

(a) How do you construct a graph for solving this problem? Describe the vertices and edges in your graph clearly and precisely.

(b) How does your algorithm work? Give a detailed description and justify its correctness.

(c) Analyse the worst-case runtime of your algorithm.

Answer

(a) First let's define a few operations

FA: fill bucket A until its full.

FB: fill bucket B until its full.

EA empty bucket A.

EB: empty bucket B.

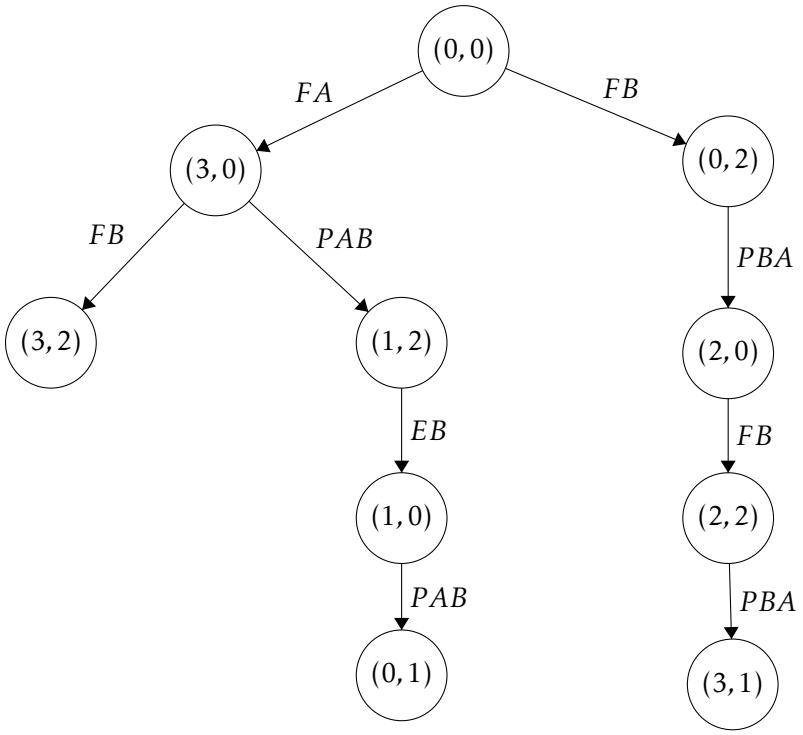
PAB: transfer water from A to B until one of the buckets is full or empty.

PBA: transfer water from B to A until one of the buckets is full or empty

Now let's define how to construct the tree $T(V,E)$. Each vertex in V is a state (x,y) with $(0 \le x \le m) \wedge (0 \le y \le n)$, meaning the amount of water in A and B. Every edge in E is one of the operations defined above. The initial state (root) is $(0,0)$.

We build the tree recursively by defining the children for each vertex. Children of a vertex (x,y) is all possible states $\{(x',y')\}$ which are different from all previous states and obtained by applying a single operation. The edge connect a child (x',y') and parent (x,y) is the operation which transfers $(x,y) \Rightarrow (x',y')$.

NOTE: every node also stores the address of all its children, its parent if exist. Every node except root also stores the edge information which connect itself with its parent in order to keep tracking of operations that have been made. Here we show an example where $m = 3, n = 2$,



(b) Suppose we already construct the graph $T(V,E)$, then BUCKETMEASURE(m,n,k) works as follow,

- First do a special case BFS on tree $T(V,E)$ starts at root $(0,0)$, search for k and return the first node $O(x,y)$ such that $x = k$ or $y = k$. Return NIL if no such node exist.
- If BFS returns a node O instead of NIL, then we do:

```
result = stack()
return EXACTINFO(O, result)

EXACTINFO(node, stack):
  if node.parent ≠ NIL:
      list ← append(node.edgeInfo)
      EXACTINFO(node.parent, stack)
  return list
```

Claim: BucketMeasure(m, n, k) returns a sequence of moves that results in one bucket (it does not matter which one) having exactly k litres of water. The number of moves in the returned sequence is smallest.

Proof of Termination

We know that BFS always terminates. It's clearly that exactInfo also always terminates if the input node is not nil, since the height of tree is finite. Therefore, procBucketMeasure(m, n, k) also terminates.

Proof of Partial Correcness

We know from the lecture that for BFS when search for k it always returns the shortest the path from root to any node (x, y) where $x = k$ or $y = k$. Hence, we just need to proof $result$ contains the correct sequence of operations.

This is also clear since we know stack always append to the top. Then when we trace back from the accepting state, the last operation is in the bottom the stack and the first operation is at top. Hence in correct sequence.

(c) Note: The number of nodes and edges of tree we constructed from (a) are both in $\mathcal{O}mn$.

First a new states is added to the tree only if it is different from all previous states; hence, every node in the tree is unique. Then there are at most $(m+1)(n+1)$ combinations for states, which means the number of nodes of tree is in $\mathcal{O}mn$.

Secondly, we know that for three $|E| = |V| - 1$; therefore, the number of edges of tree is also in $\mathcal{O}mn$.

The worst case occurs when such k does not exist, which means we have search every node in tree. Then the worst-case runtime is $\mathcal{O}(mn)$.