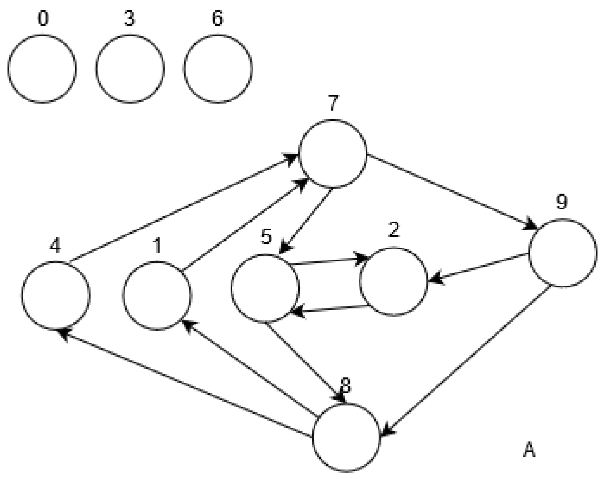
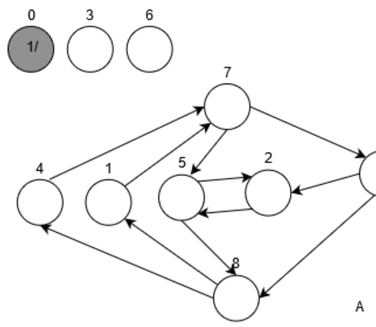
Part A

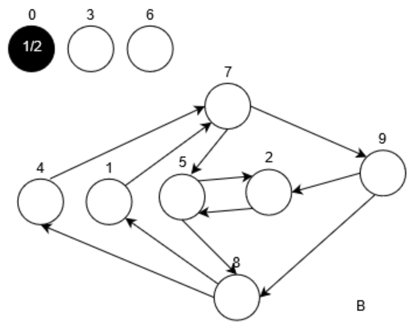
1. (a) Initial number: 44925817
   SNI: 984792581752

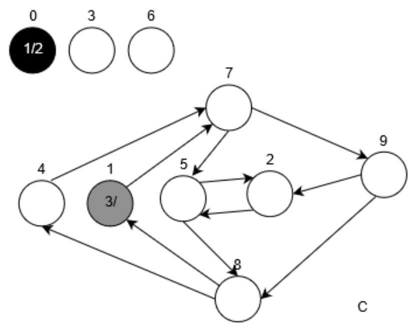
   (b)

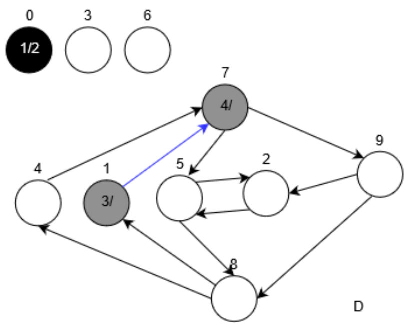


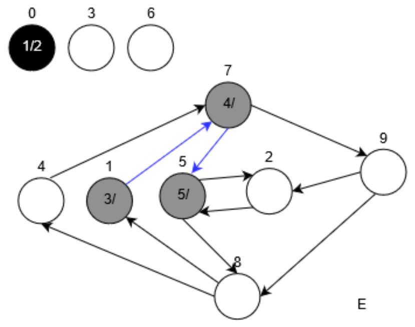
2. (a)

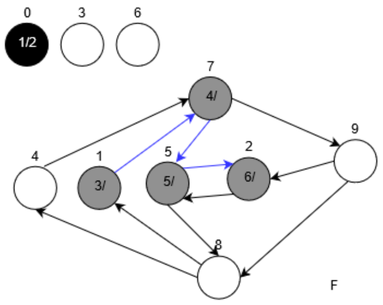




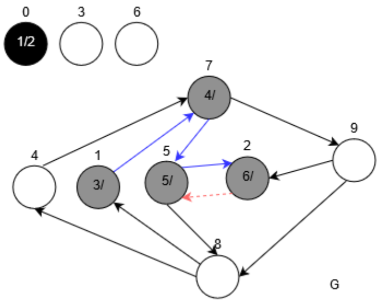




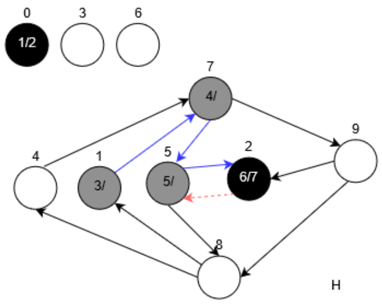



0
3
6
1/2
7
4/
9
4
1
5
2
3/
5/
6/
8

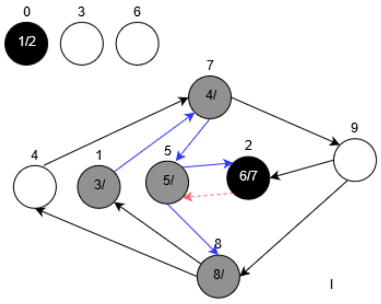
F


0
3
6
1/2
7
4/
9
4
1
5
2
3/
5/
6/
8

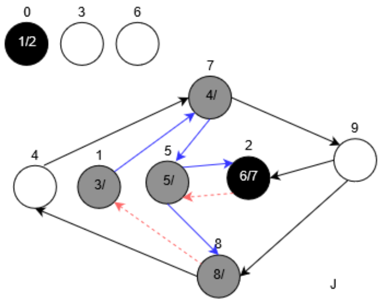
G


0
3
6
1/2
7
4/
9
4
1
5
2
3/
5/
6/7
8

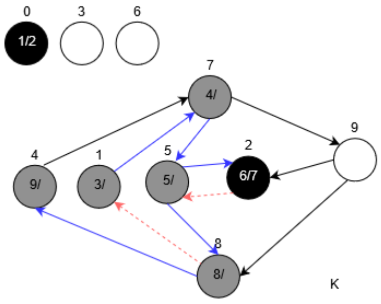
H


0
3
6
1/2
7
4/
9
4
1
5
2
3/
5/
6/7
8
8/

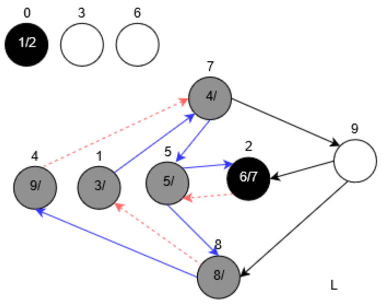
I


0
3
6
1/2
7
4/
9
4
1
5
2
3/
5/
6/7
8
8/

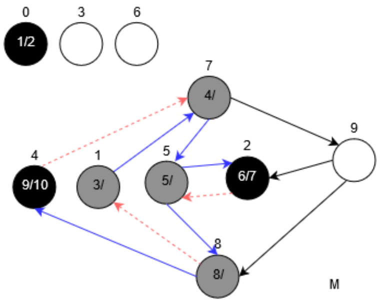
J


0
3
6
1/2
7
4/
9
4
1
5
2
9/
3/
5/
6/7
8
8/

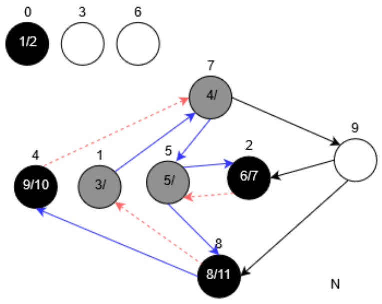
K


0
3
6
1/2
7
4/
9
4
1
5
2
9/
3/
5/
6/7
8
8/

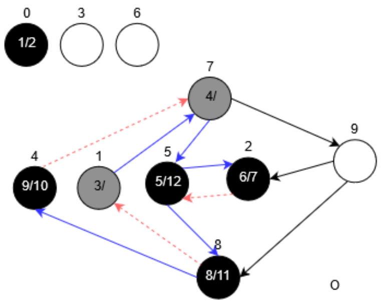
L


0
3
6
1/2
7
4/
9
4
1
5
2
9/10
3/
5/
6/7
8
8/

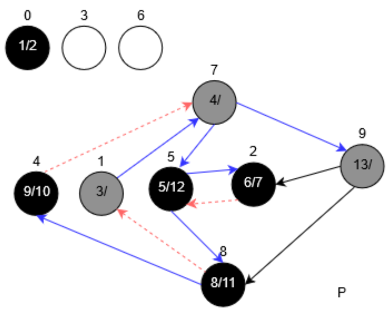
M


0
3
6
1/2
7
4/
9
4
1
5
2
9/10
3/
5/
6/7
8
8/11

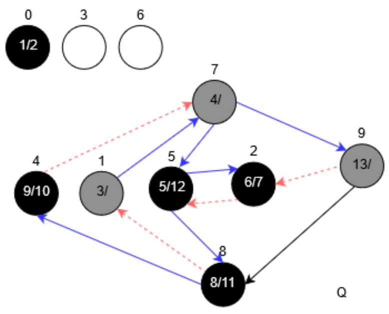
N


0
3
6
1/2
7
4/
9
4
1
5
2
9/10
3/
5/12
6/7
8
8/11

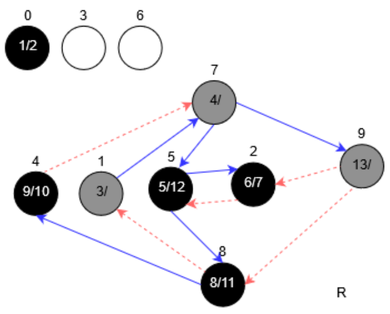
O


0
1/2
3
6
7
4/
4
9/10
1
3/
5
5/12
2
6/7
9
13/
8
8/11

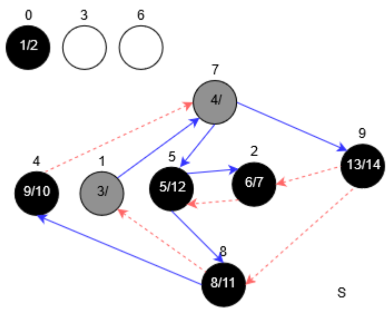
P


0
1/2
3
6
7
4/
4
9/10
1
3/
5
5/12
2
6/7
9
13/
8
8/11

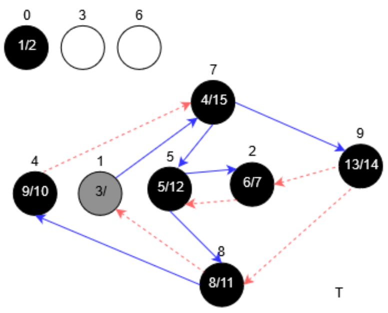
Q


0
1/2
3
6
7
4/
4
9/10
1
3/
5
5/12
2
6/7
9
13/
8
8/11

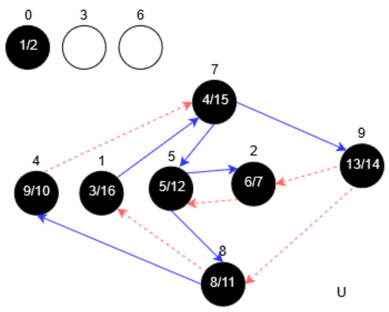
R


0
1/2
3
6
7
4/
4
9/10
1
3/
5
5/12
2
6/7
9
13/14
8
8/11

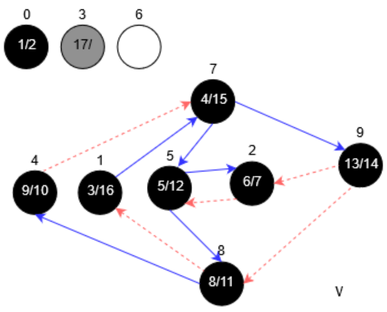
S


0
1/2
3
6
7
4/15
4
9/10
1
3/
5
5/12
2
6/7
9
13/14
8
8/11

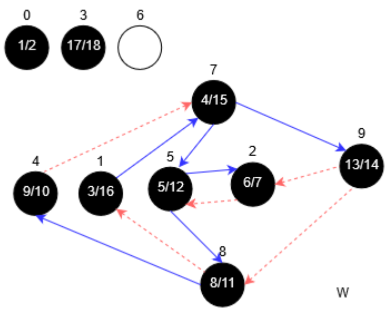
T


0
1/2
3
6
7
4/15
4
9/10
1
3/16
5
5/12
2
6/7
9
13/14
8
8/11

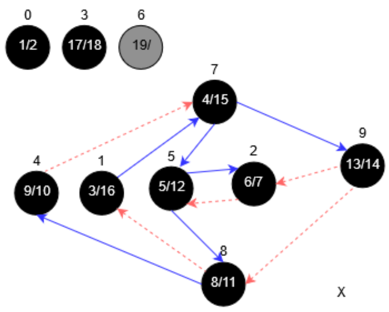
U


0
1/2
3
17/
6
7
4/15
4
9/10
1
3/16
5
5/12
2
6/7
9
13/14
8
8/11

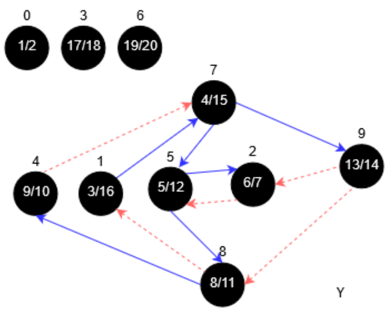
V


0
1/2
3
17/18
6
7
4/15
4
9/10
1
3/16
5
5/12
2
6/7
9
13/14
8
8/11

W

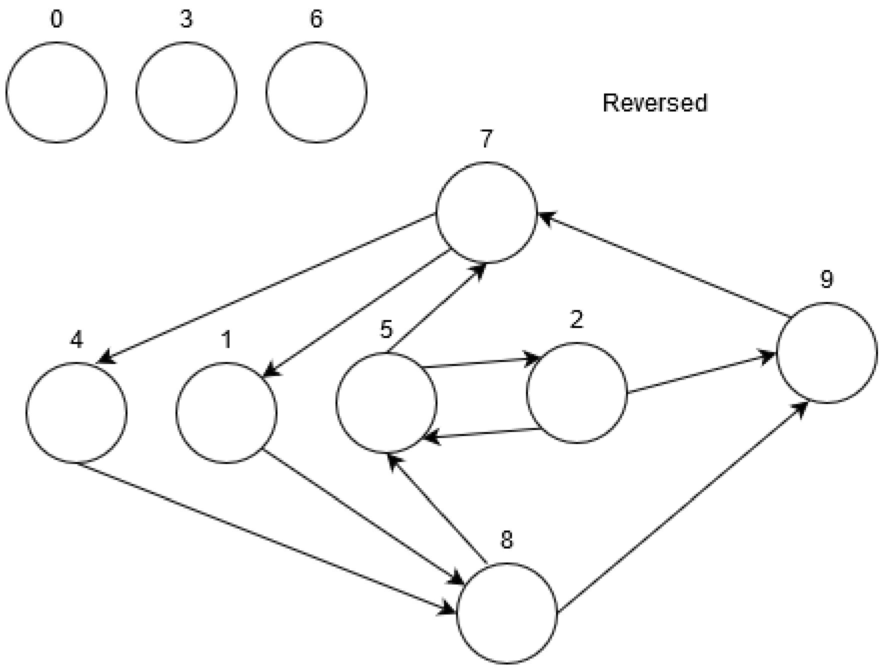

0
1/2
3
17/18
6
19/
7
4/15
4
9/10
1
3/16
5
5/12
2
6/7
9
13/14
8
8/11

X

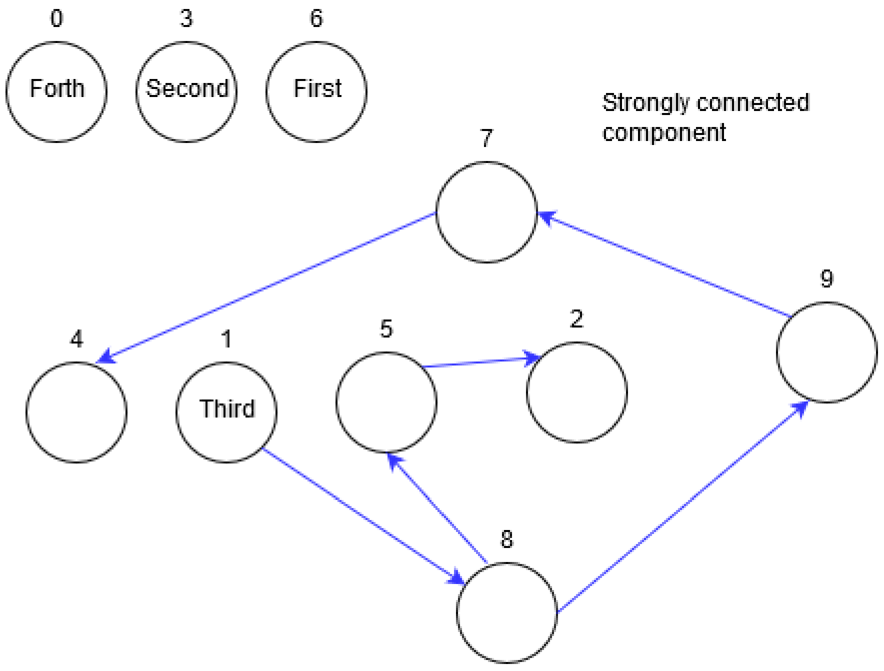

0
1/2
3
17/18
6
19/20
7
4/15
4
9/10
1
3/16
5
5/12
2
6/7
9
13/14
8
8/11

Y

(b)


0
3
6
Reversed
7
4
1
5
2
9
8

(c)


0
Forth
3
Second
6
First
Strongly connected component
7
4
1
Third
5
2
9
8

Part B

4. (a)

```
BFS(W, Source):
    Initialize()
    unvisitedWarehouse = a copy of W
    Q.enqueue(Source)
    S.parent = null
    Destination = null // have not found destination
    While(Destination is null && Q is not empty):
        C = Q.dequeue()
        unvisitedWarehouse.remove(C)
        // keep track on the warehouse that can be visited
        possibleTypes = C.availableItemTypes()
        possibleTypes = possibleTypes – visitedTypes
        // already visited types wouldn't be a Transferable Type
        Edges = GetNeighbours(possibleTypes, unvisitedWarehouse)
        For each e in Edges:
            // Edge is a data structure with warehouse and type
            if (e.Warehouse is not grey):
                // Keep track of grey warehouses so that you will not
                // enqueue the same warehouse twice
                Q.enqueue(e.Warehouse), mark e.Warehouse as grey
            visitedType.add(e.Type) // keep track of visited Types
            cameFrom.put(e.warehouse, Edge(C, e.type))
            if(e.warehouse.canAddItem(e.type)):
                Destination = e.warehouse
    ShowTransactions(Destination, cameFrom) // iterate cameFrom

GetNeighbours(possibleTypes, unvisitedWarehouse):
    // find the possible candidates in only possibleTypes
    // and unvisited warehouses
    Neighbours = an empty List
    For each t in possibleTypes:
        For each w in unvisitedWarehouse:
            If (w can store t):
                Neighbours.add(Edge(w, t))
    Return Neighbours
```

(b)
Line 1 initialization takes O(1) time, it’s just setting up some data structures so it knows what are the unvisited warehouse, already visited item types, where the transaction is coming from and to and via what kind of item type.
Line 2 making a copy of all warehouse takes O(w) time.
Line 3 – 5 take O(1).
Line 6 is a while loop that has the property at worst case loops w times since each warehouse will be visited at most one time and worst case you loop to the last one and find no valid transaction.
Line 7,8 take O(1).
Line 10, each iteration you find the current warehouse’s available item times, so it’s $\sum_{i=0}^{w} Warehouse_i . getAvailableItem() = \mathrm{O(n)}$. Justification: The total available items is bounded below total item.
Assuming the set remove operation is constant time for line 11 where the set operation happens.
Line 11, the visitedType will start from 0 to at most t within w loops, so it’s $\frac{(0+t)\,w}{2} \in O(tw)$.
Line 13, calls GetNeighbours so look at GetNeighbours script. Assume that the possible types will not overlap with visitedType so it’s worst case, and each time unvisitedWarehouse gets smaller.
$\sum_{i=0}^{w} Warehouse_i . getAvailableItem() * (w - i) \leq$
$\sum_{i=0}^{w} Warehouse_i . getAvailableItem() * \mathrm{w} = \mathrm{O(nw)}$
Line 14 is just like line 13, it takes the output from GetNeighbours and do some constant time work. So it’s $\in O(\mathrm{nw})$ as well.
Line 24 ShowTransactions to through cameFrom map which the algorithm uses it to keep track on the information of transactions. Transactions will have at most w transaction. O(w)
To sum up. The BFS algorithm takes $some\ O(1) + O(w) + O(n) + O(tw) + O(\mathrm{nw}) + O(\mathrm{nw}) + O(w) = O(\mathrm{tw} + \mathrm{nw})$.