8.0 a)
// the Floyd-Warshall algorithm that stores the answer

global Ecost(1...n, 1...n) initialized to store the cost of pairs (i, j) and infinity if no edge is present;
global Pcost(1...n, 1...n) stores the shortest path, initialized to equal Ecost;
global kIntermediate(1...n, 1...n) initialized to be filled with 0 if there is an edge, and -1 otherwise;

```
global G(V, E);
procedure F-W(n);
   for k <- 1 to n do
      for i <- 1 to n do
         for j <- 1 to n do
            if Pcost(i,j) > Pcost(i,k) + Pcost(k,j) then
               Pcost(i,j) <- Pcost(i,k) + Pcost(k,j);
               kIntermediate(i,j) <- k; // store the answer
            end if;
         end for;
      end for;
   end for;
end_F-W;
```

Correct

b)

```
// path recovery algorithm
// drive
procedure PathDrive(i,j);
   if kIntermediate(i,j) == -1 then
      // i and j are not connected
      print('No path');
   else
      PrintPath(i,j);
      print(j);
   end if;
end_PathDrive;

// actual algorithm
procedure PrintPath(i,j);
   k <- kIntermediate(i,j);
   if k == 0 then
      // shortest path is the edge between i and j
      print(i);
   else
      PrintPath(i,k);
      PrintPath(k,j); // does not print j
   end if;
```

end_PrintPath;

Correct

8.05

This question requires a slight change to the standard F-W algorithm to calculate 2 edges (i,k) and (k,j). As such, we can put the k-loop inside the i-j-loops and initialize a Best(i,j) as infinity to be updated and store the best k.

```
global Best(1...n, 1...n) stores the best path, initialized to be filled with infinity;
global Ecost(1...n, 1...n) initialized to store the edge costs. For pairs with no edges or edges
from and to the same vertex, the cost is infinity;
procedure TwoEdges(n);
   for i <- 1 to n do
      for j <- 1 to n do
         for k <- 1 to n do
            if Best(i,j) > Ecost(i,k) + Ecost(k,j) then
               Best(i,j) <- Ecost(i,k) + Ecost(k,j);
            end if;
         end for;
      end for;
   end for;
end_TwoEdges;
```

Correct

8.1 Floyd-Warshall

// the Floyd-Warshall algorithm that stores the first vertex after i

```
global Ecost(1...n, 1...n) initialized to store the cost of pairs (i, j) and infinity if no edge is
present;
global Pcost(1...n, 1...n) stores the shortest path, initialized to equal Ecost;
global kIntermediate(1...n, 1...n) initialized to be filled with 0 if there is an edge, and -1
otherwise;
global G(V, E);
procedure F-W(n);
   for k <- 1 to n do
      for i <- 1 to n do
         for j <- 1 to n do
            if Pcost(i,j) < Pcost(i,k) + Pcost(k,j) then
```

```
            kIntermediate(i,j) <- j;
          else if kIntermediate(i,j) == 0 or kIntermediate(i,j) == -1 then
            kIntermediate(i,j) <- k;
          end if;
        end for;
      end for;
   end for;
end_F-W;
```

Correct

8.rwb

Copy G three times, and connect G1 to G2 with only redges; G2 to G3 with only wedges, and G3 to G1 with only bledges. We can then apply the FW algorithm on this super graph to find all pairs that start and end at G1.

Correct

8.cookie

a)

```
global Ecost(1...n, 1...n) initialized to store the cost of pairs (i, j) and infinity if no edge is
present;
global Pcost(1...n, 1...n) stores the shortest path, initialized to equal Ecost;
global kIntermediate(1...n, 1...n) initialized to be filled with 0 if there is an edge, and -1
otherwise;
global Cookie[1..n];
global G(V, E);
procedure F-W(n);
   for k <- 1 to n do
      for i <- 1 to n do
        for j <- 1 to n do
          if Pcost(i,j) > Pcost(i,k) + Pcost(k,j) then
            Pcost(i,j) <- Pcost(i,k) + Pcost(k,j);
            kIntermediate(i,j) <- k; // store the answer
          end if;
        end for;
      end for;
   end for;
end_F-W;

// path recovery algorithm
```

```
// drive
global cookies initialized at 0;
procedure PathDrive(i,j);
   if kIntermediate(i,j) ≠ -1 then
      PrintPath(i,j);
      cookies <- cookies + Cookie[j];
   end if;
end_PathDrive;

// actual algorithm
procedure PrintPath(i,j);
   k <- kIntermediate(i,j);
   if k == 0 then
      // shortest path is the edge between i and j
      cookies <- cookies + Cookie[i];
   else
      PrintPath(i,k);
      PrintPath(k,j); // does not print j
   end if;
end_PrintPath;
```

My counting part happens in the path recovery part.

b)

To store equal paths, we change kIntermediate(i,j) to store an array of k which offers the shortest path from i to j.

```
global Ecost(1...n, 1...n) initialized to store the cost of pairs (i, j) and infinity if no edge is present;
global Pcost(1...n, 1...n) stores the shortest path, initialized to equal Ecost;
global kIntermediate(1...n, 1...n) initialized to be filled with 0 if there is an edge, and -1 otherwise;
global Cookie[1..n];
global G(V, E);
procedure F-W(n);
   for k <- 1 to n do
      for i <- 1 to n do
         for j <- 1 to n do
            if Pcost(i,j) > Pcost(i,k) + Pcost(k,j) then
               Pcost(i,j) <- Pcost(i,k) + Pcost(k,j);
               kIntermediate(i,j).append(k); // store the answer
            end if;
         end for;
      end for;
   end for;
```

```
end_F-W;

// path recovery algorithm
// drive
global cookies initialized at 0;
procedure PathDrive(i,j);
   if kIntermediate(i,j) ≠ -1 then
      cookies <- PrintPath(i,j) + Cookie[j];
   end if;
end_PathDrive;

// actual algorithm
global max initialized to be -infinity;
function PrintPath(i,j);
   for x <- 0 to kIntermediate(i,j).size() do
      cookies <- 0;
      k <- kIntermediate(i,j)[x];
      if k == 0 then
         // shortest path is the edge between i and j
         cookies <- cookies + Cookie[i];
      else
         PrintPath(i,k);
         PrintPath(k,j); // does not print j
      end if;
      if cookies > max then
         max <- cookies;
         return max;
      end if;
   end for;
end_PrintPath;
```

For the FW part, I should have included a condition where the new path is equal to the best path.

8.vb

a)

Copy Manhattan, connect the two VBs in the two graphs. The edge that connects VBs has a 0 cost. The complete path would start from G1, and end in G2.

Correct

b)

The constant c should be 8, since the graph is copied into 2, and 2^3 = 8.

Correct

8.cookied

a)

```
global Ecost(1...n, 1...n) initialized to store the cost of pairs (i, j) and infinity if no edge is
present;
global Pcost(1...n, 1...n) stores the shortest path, initialized to equal Ecost;
global kIntermediate(1...n, 1...n) initialized to be filled with 0 if there is an edge, and -1
otherwise;
global Cookie[1..n];
global G(V, E);
procedure F-W(n);
   for k <- 1 to n do
      for i <- 1 to n do
         for j <- 1 to n do
            if Pcost(i,j) > Pcost(i,k) + Pcost(k,j) then
               if Cookie[k] > 0 then
                  Pcost(i,j) <- Pcost(i,k) + Pcost(k,j);
                  kIntermediate(i,j).append(k); // store the answer
               end if;
            end if;
         end for;
      end for;
   end for;
end_F-W;
```

I used a nested if-then to test if a path contains cookies.

b)

```
global Cookie[1..n];
global Best(1...n, 1...n) stores the best path, initialized to be filled with infinity;
global Ecost(1...n, 1...n) initialized to store the edge costs. For pairs with no edges or edges
from and to the same vertex, the cost is infinity;
procedure TwoEdges(n);
   for i <- 1 to n do
      for j <- 1 to n do
         for k <- 1 to n do
            if Best(i,j) > Ecost(i,k) + Ecost(k,j) then
               if Cookie[k] + Cookie[j] > 0 then
                  Best(i,j) <- Ecost(i,k) + Ecost(k,j);
               end if;
            end if;
         end for;
      end for;
   end for;
```

end_TwoEdges;

Here I used the same logic as part a) which I added an additional if test.

8.2718

global Ecost(1...n, 1...n) initialized to store the cost of pairs (i, j) and infinity if no edge is present;
global Pcost(1...n, 1...n) stores the shortest path, initialized to equal Ecost;
global kIntermediate(1...n, 1...n) initialized to be filled with 0 if there is an edge, and -1 otherwise;
global G(V, E);
global Edges(i,j) stores the number of edges in the shortest path, initialized to be 1 if there is an edge, and 0 otherwise;

```
procedure F-W(n);
   for k <- 1 to n do
      for i <- 1 to n do
         for j <- 1 to n do
            if Pcost(i,j) > Pcost(i,k) + Pcost(k,j) then
               Pcost(i,j) <- Pcost(i,k) + Pcost(k,j);
               kIntermediate(i,j) <- k; // store the answer
               Edges(i,j) <- Edges(i,k) + 1;
            end if;
         end for;
      end for;
   end for;
end_F-W;
```

The line highlighted should be Edges(i,j) <- Edges(i,k) + Edges(k,j);

8.3 Floyd-Warshall

The wrong placement of the k-loop would result in the algorithm calculating the edges from i to k, and k to j. It will not calculate the complete paths.

Correct

8.21

a)

```
global Ecost(1...n, 1...n) initialized to store the cost of pairs (i, j) and infinity if no edge is
present;
global Pcost(1...n, 1...n) stores the shortest path, initialized to equal Ecost;
global kIntermediate(1...n, 1...n) initialized to be filled with 0 if there is an edge, and -1
otherwise;
global G(V, E);
procedure F-W(n);
  for k <- 1 to n do
    for i <- 1 to n do
      for j <- 1 to n do
        if Pcost(i,j) > max{longest(i,k), longest(k,j)} then
          Pcost(i,j) <- max{longest(i,k), longest(k,j)};
          kIntermediate(i,j) <- k; // store the answer
        end if;
      end for;
    end for;
  end for;
end_F-W;

// algorithm to determine the lonest edge in a path
function longest(i,j);
  for all edges x on path (i,j) return max{x};
end_longest;
```

Correct

b)

```
global Ecost(1...n, 1...n) initialized to store the cost of pairs (i, j) and infinity if no edge is
present;
global Pcost(1...n, 1...n) stores the shortest path, initialized to equal Ecost;
global kIntermediate(1...n, 1...n) initialized to be filled with 0 if there is an edge, and -1
otherwise;
global G(V, E);
procedure F-W(n);
  for k <- 1 to n do
    for i <- 1 to n do
      for j <- 1 to n do
        if Pcost(i,j) > product(i,k) * product(k,j) then
          Pcost(i,j) <- product(i,k) * product(k,j);
          kIntermediate(i,j) <- k; // store the answer
        end if;
      end for;
    end for;
  end for;
end_F-W;


// algorithm to determine the product of all edges in a path
```

```
function longest(i,j);
   product <- 1;
   for all edges x on path (i,j) do
      product <- product * x;
   end for;
   return product;
end_longest;
```

Correct

c)
That cycles should only be travelled once; If there is an edge with cost between 0 and 1, cycling more times will give smaller paths.

Correct

8.2 Floyd-Warshall

```
global Ecost(1...n, 1...n) initialized to store the cost of pairs (i, j) and infinity if no edge is
present;
global Pcost(1...n, 1...n) stores the shortest path, initialized to equal Ecost;
global kIntermediate(1...n, 1...n) initialized to be filled with 0 if there is an edge, and -1
otherwise;
global G(V, E);
procedure F-W(n);
   for k <- 1 to n do
      for i <- 1 to n do
         for j <- 1 to n do
            if Pcost(i,j) < Pcost(i,k) + Pcost(k,j) then
               kIntermediate(i,j) <- i;
            else
               kIntermediate(i,j) <- k;
            end if;
         end for;
      end for;
   end for;
end_F-W;
```

I did not initialize kIntermediate as in the solution. It should store the value of kIntermediate(k,j) when updated.

8.33 Shortest path

Copy the graph into 2 pieces. G1 has all edges from s to banks, and G2 has all edges from shops to s. Connect G1 and G2 with edges from banks to shops. Perform a FW algorithm on the overall graph, find the shortest path from s1 to s2.

I think mine would potentially work. Instead of having 3 floors, I put the top floor back to the original floor.

8.39 Floyd-Warshall

a)
Copy G into 2 copies. Connect G1 and G2 with all edges. We start at G1 (or G2), and finish at G1 (or G2).

Mine could work since each move comes in pairs.

b)
Copy G into 2 copies. In each subgraph, connect vertices with edges calculated using the 2-step algorithm. Connect G1 and G2 using the original edges. We start at G1, and finish at G2.

We can use the 2-step algorithm and have an alternative EEcost array.

c)

```
global Ecost(1...n, 1...n) initialized to store the cost of pairs (i, j) and infinity if no edge is
present;
global Pcost(1...n, 1...n) stores the shortest path, initialized to equal Ecost;
global kIntermediate(1...n, 1...n) initialized to be filled with 0 if there is an edge, and -1
otherwise;
global Cookie[1..n];
global G(V, E);
procedure F-W(n);
  for k <- 1 to n do
    for i <- 1 to n do
      for j <- 1 to n do
        if Pcost(i,j) > Pcost(i,k) + Pcost(k,j) and (Edges(i,k) + 1) mod 2 == 0 then
          Pcost(i,j) <- Pcost(i,k) + Pcost(k,j);
          kIntermediate(i,j) <- k; // store the answer
        end if;
      end for;
    end for;
  end for;
end_F-W;
```

I should recognize the need to calculate odd paths as well.

8.41 Floyd-Warshall

global Ecost(1...n, 1...n) initialized to store the cost of pairs (i, j) and infinity if no edge is present;
global Pcost(1...n, 1...n) stores the shortest path, initialized to equal Ecost;
global kIntermediate(1...n, 1...n) initialized to be filled with 0 if there is an edge, and -1 otherwise;
global kSecond(1...n, 1...n) same as kIntermediate, but stores the 2nd shortest path;
global Cookie[1..n];
global G(V, E);

```
procedure F-W(n);
  for k <- 1 to n do
    for i <- 1 to n do
      for j <- 1 to n do
        if Pcost(i,j) > Pcost(i,k) + Pcost(k,j) then
          Pcost(i,j) <- Pcost(i,k) + Pcost(k,j);
          kSecond(i,j) <- kIntermediate(i,j);
          kIntermediate(i,j) <- k; // store the answer
        else if Pcost(i,j) = Pcost(i,k) + Pcost(k,j) then
          kSecond(i,j) <- kIntermediate(i,j);
        end if;
      end for;
    end for;
  end for;
end_F-W;
```

I should use another array PPcost to store the second path, which make it easier to distinguish between different situations.

8.42 Floyd-Warshall

global Ecost(1...n, 1...n) initialized to store the cost of pairs (i, j) and infinity if no edge is present;
global Pcost(1...n, 1...n) stores the shortest path, initialized to equal Ecost;
global kIntermediate(1...n, 1...n) initialized to be filled with 0 if there is an edge, and -1 otherwise;
global shortestCount(1...n, 1...n) initialized to be filled with 1 if there is an edge, and 0 otherwise;
global G(V, E);

```
procedure F-W(n);
  for k <- 1 to n do
    for i <- 1 to n do
      for j <- 1 to n do
```

```
            if Pcost(i,j) > Pcost(i,k) + Pcost(k,j) then
               shortestCount(i,j) <- shortestCount(i,k) + shortestCount(k,j);
               Pcost(i,j) <- Pcost(i,k) + Pcost(k,j);
               kIntermediate(i,j) <- k; // store the answer
            else if Pcost(i,j) = Pcost(i,k) + Pcost(k,j) then
               shortestCount(i,j) <- shortestCount(i,j) + 1;
            end if;
         end for;
      end for;
   end for;
end_F-W;
```

For the smaller case , I should time the two parts together; for the equal case, I should add two parts instead.