US 13

```
procedure Kruskal(edges: Array of Edge, V: Integer, mode: Integer);
var
graph: Graph;
    currentTime: LongInt;
result: Array of Edge;
1.  subsets: Array of Subset;
2.  v, e, i, x, y, mstCost: Integer;
3.  execTime: Float;
4.  next_edge: Edge;
begin
5.  graph := DrawGraph(edges, 'MST');
6.  currentTime := GetCurrentTime();
7.  result := EmptyArray();

8.  edges := SortArrayList(edges);

9.  SetLength(subsets, V);
10. for v := 0 to V-1 do
11. subsets[v] := Subset(v, 0);
   end;

12. e := 0;
13. i := 0;

14. while (e < V - 1) and (i < Length(edges)) do
   begin
15. next_edge := edges[i];
16. i := i + 1;

17.  x := Find(subsets, next_edge.from.index);
18. y := Find(subsets, next_edge.to.index);

19. if x <> y then
     begin
20.  AddToArray(result, next_edge);
21.  Union(subsets, x, y);
22.  e := e + 1;
     end;
   end;

23. execTime := (GetCurrentTime() - currentTime) / 1000000.0;
24. mstCost := 0;

25. for edge in result do
   begin
```

26. Print(edge.from.name + ' - ' + edge.to.name + ': ' + edge.weight);
27. mstCost := mstCost + edge.weight;
    end;

28. Print('MST cost: ' + mstCost);
29. Print('Time: ' + execTime + 'us');

30. GenerateGraphVizSVG(edges, result, 'Minimum Spanning Tree', mstCost);

31. graph.SetAttribute('ui.title', 'Minimum Spanning Tree');
32. graph.SetAttribute('ui.quality');
33. graph.SetAttribute('ui.antialias');

34. OutputGraphCSV(result, mstCost);

35. highlightedGraph := HighlightGraph(graph, result);
36. AddTextToGraph(highlightedGraph, 'Cost: ' + mstCost + ' Time: ' + execTime + 'ms');
end;

Worst-case time complexity

Linha Operações Primitivas

1 | 1A
2 | 1A
3 | 1A
4 | 1A
5 | 1A
6 | 1A
7 | 1A
8 | -
9 | 1A
10-11 | 1A + 1I + 1C + 1A
12 | 1A
13 | 1A
14 | 2C
15 | 1A
16 | 1I
17 | 1A
18 | 1A
19 | 1C
20 | 1A

21 | 1A
22 | 1I
23 | 1A + 1Op
24 | 1A
25-27 | 1C + 1A
28 | 1A
29 | 1A
30 | 1A
31 | 1A
32 | 1A
33 | 1A
34 | 1A
35 | 1A
36 | 1A

Estimativa de O: O(ElogE), onde E é o número de arestas e V é o número de vértices.

US17

Pseudo code:

```
procedure dijkstra(pointNames: array of String; distancesMatrix: array of array of Integer;
source: String);
var
  1. n, sourceIndex, i, u, v: Integer;
  2. distance: array of Integer;
  3. visited: array of Boolean;
  4. predecessor: array of Integer;
begin
  5. n := Length(pointNames);
  6. SetLength(distance, n);
  7. SetLength(visited, n);
  8. SetLength(predecessor, n);

  9. for i := 0 to n - 1 do
  begin
    10. distance[i] := MAXINT;
```

```
  11. visited[i] := False;
  12. predecessor[i] := -1;
 end;

 13. sourceIndex := IndexOf(pointNames, source);
 14. distance[sourceIndex] := 0;

 15. for i := 0 to n - 2 do
 begin
  16. u := minDistance(distance, visited);
  17. visited[u] := True;
  18. for v := 0 to n - 1 do
  begin
   if (not visited[v]) and (distancesMatrix[u][v] <> 0) and (distance[u] <> MAXINT) and
(distance[u] + distancesMatrix[u][v] < distance[v]) then
    begin
     19. distance[v] := distance[u] + distancesMatrix[u][v];
     20. predecessor[v] := u;
    end;
  end;
 end;

 21. printSolution(distance, n, pointNames, predecessor);
end;

function minDistance(distance: array of Integer; visited: array of Boolean): Integer;
var
 22. min, minIndex, i: Integer;
begin
 23. min := MAXINT;
 24. minIndex := -1;
 25. for i := 0 to Length(distance) - 1 do
 begin
  if (not visited[i]) and (distance[i] <= min) then
  begin
   26. min := distance[i];
   27. minIndex := i;
  end;
 end;
 28. minDistance := minIndex;
end;
```

Worst-case time complexity

1-	1A
2-	1A
3-	1A
4-	1A

5-   1A
6-   1A
7-   1A
8-   1A
9-   (n+1)C + nI
10-   nA
11-   nA
12-   nA
13-   nC
14-   1A
15-   (n-1)C + (n-1)I
16-   (n-1)(nC + nC) = n(n-1)C
17-   (n-1)A
18-   (n-1)(nC + nI)
19-   (n-1)nA
20-   (n-1)nA
21-   1R
22-   1A
23-   1A
24-   1A
25-   (n+1)C + nI
26-   nC + nA
27-   nA
28-   1R

Estimativa total da complexidade: $O(n^2)$.

US 18

Pseudocode:
1. function Dijkstra(pointNames: array of String; distancesMatrix: array of array of Integer; source, point: String; finalPrint: Boolean): List;
2. var
3. n, i, u, v, sourceIndex: Integer;
4. distance, predecessor: array of Integer;
5. visited: array of Boolean;
6. output: List;
7. begin
8. n := Length(pointNames);
9. SetLength(distance, n);
10. SetLength(visited, n);
11. SetLength(predecessor, n);
12. output := CreateList();
13. for i := 0 to n - 1 do
14. begin
15. distance[i] := MaxInt;
16. visited[i] := False;
17. predecessor[i] := -1;
18. end;

```
19.  sourceIndex := IndexOf(pointNames, source);
20.  distance[sourceIndex] := 0;
21.  for i := 0 to n - 2 do
22.  begin
23.  u := MinDistance(distance, visited);
24.  visited[u] := True;
25.  if pointNames[u] = point then
26.  Break;
27.  for v := 0 to n - 1 do
28.  begin
29.  if (not visited[v]) and (distancesMatrix[u][v] <> 0) and (distance[u] <> MaxInt) and
(distance[u] + distancesMatrix[u][v] < distance[v]) then
30.  begin
31.  distance[v] := distance[u] + distancesMatrix[u][v];
32.  predecessor[v] := u;
33.  end;
34.  end;
35.  end;
36.  output.Append(PrintSolution(distance, n, pointNames, predecessor, point));
37.  output.Append(distance);
38.  Result := output;
39.  end;
40.  function GetClosestAP(assemblyPoints, pointNames: array of String;
distancesMatrix: array of array of Integer; point: String): String;
41.  var
42.  minDistance, distanceToPoint: Integer;
43.  closestAP: String;
44.  distancesFromAP: array of Integer;
45.  ap: String;
46.  begin
47.  minDistance := MaxInt;
48.  closestAP := nil;
49.  for ap in assemblyPoints do
50.  begin
51.  distancesFromAP := Dijkstra(pointNames, distancesMatrix, ap, point, False)[1];
52.  distanceToPoint := distancesFromAP[IndexOf(pointNames, point)];
53.  if distanceToPoint < minDistance then
54.  begin
55.  minDistance := distanceToPoint;
56.  closestAP := ap;
57.  end;
58.  end;
59.  Result := closestAP;
60.  end;
```

Complexity analysis:

1   -
2   -
3   5A
4   2A
5   1A
6   1A
7   -
8   1A
9   1A + nA
10   1A + nA
11   1A + nA
12   1A
13   (n+1)A + (n+1)C
14   -
15   nA
16   nA
17   nA
18   -
19   nA + nC
20   1A
21   nA + nC
22   -
23   $n^2(A + C)$
24   nA
25   nC
26   nC
27   n(nA + nC)
28   -
29   $n^2(4C)$
30   -
31   $n^2(Op + A)$
32   $n^2(A)$
33   -
34   -
35   -
36   nA + nOp
37   1A
38   1R
39   -
Estimativa de O: $O(n^2)$

Percentages:
Diogo Garrett nº1230556: 20%
Rodrigo Araújo nº1230560: 20%
Salvador Machado nº1230561: 20%
Bruno Teixeira nº1231091: 20%
Pedro Lemos nº1221009: 20%