SPRINT 3 – US19

MATEMÁTICA DISCRETA

Instituto Superior de Engenharia do Porto

1DCDD – GRUPO 034 - CODEFLOW

PEDRO COSTA (1221790)
RUI SANTIAGO (1221402)
FRANCISCO TROCADO (1230608)

Table of Contents

Algoritmo de Kruskal – US13

Neste método, o arrayDeSacos representa um conjunto de “sacos” ao qual cada contém vértices dentro.

Size representa o tamanho da lista.

Source é o vértice de origem da aresta

Destination é o vértice de destino da aresta

Pseudocódigo:

LINHA	PSEUDOCÓDIGO
-	procedure getMinimalSpanningTree()
1	minimalSpanningTree := empty list
2	bubbleSortByDistance(edges: list of Edge)
3	numVertices := size(vertexes)
4	criterioParagem := numVertices - 1
5	addedEdges := 0
6	arrayDeSacos := nova lista de sacos de tamanho numVertices
7	for i := 0 to numVertices – 1
8	saco := nova lista
9	saco.add(vertexes[i])
10	arrayDeSacos.add(saco)
11	for j := 0 to size(edges) - 1 and addedEdges <= criterioParagem
12	edge := edges[j]
13	v1 := edge.source
14	v2 := edge.destination
15	indexSacoV1 := -1
16	indexSacoV2 := -1
17	for i := 0 to size(arrayDeSacos) – 1
18	if isVertexInSet(arrayDeSacos[i], v1) then
19	indexSacoV1 := i
20	if isVertexInSet(arrayDeSacos[i], v2) then
21	indexSacoV2 := i
22	if indexSacoV1 != indexSacoV2 then
23	arrayDeSacos[indexSacoV1].addAll(arrayDeSacos[indexSacoV2])
24	arrayDeSacos[indexSacoV2] := empty list
25	addedEdges++
26	minimalSpanningTree.add(edge)
27	return minimalSpanningTree

Análise da Complexidade:

LINHA	ALGORITMO GETMINIMALSPANNINGTREE()
1	1A
2	$O(n^2)$ → Algoritmo bubbleSortByDistance
3	1A
4	1A+ 1Op
5	1A
6	1A
7	(n+1)I + (n+1)C
8	nA
9	nA
10	nA
11	(n+1)I+(n+1)C
12	nA
13	nA
14	nA
15	nA
16	nA
17	n((n+1)I + (n+1)C)
18	n(nC) →Usa o Algoritmo isVertexInSet
19	n(nA)
20	n(nC)
21	n(nA)
22	nC
23	nA
24	nA
25	nA
26	nA
27	1R
TOTAL	6A + O(n^2) + 1Op + (n+1)I + (n+1)C + 8nA + 6nI + 6nC + 1R
ESTIMATIVA (O)	$O(n^2)$

Método BubbleSortByDistance

Este método ordena uma lista de arestas de um grafo por ordem crescente de distância (custo).

edges: list of Edge representa uma lista de arestas de um grafo

distance é a distância (double) que representa a distância/custo dessa aresta.

Pseudocódigo:

LINHA	PSEUDOCÓDIGO
-	procedure bubbleSortByDistance(edges: list of Edge)
1	n := size(edges)
2	for i := 0 to n – 2
3	for j := 0 to n - i – 2
4	if edges[j].distance > edges[j + 1].distance then
5	temp := edges[j]
6	edges[j] := edges[j + 1]
7	edges[j + 1] := temp

Análise da Complexidade:

LINHA	ALGORITMO BUBBLESORTBYDISTANCE
1	1A
2	(n)I + (n)C
3	$(\frac{n(n-1)}{2})(I+C)+(\mathrm{I}+\mathrm{C})$
4	$(\frac{n(n-1)}{2})C$
5	$(\frac{n(n-1)}{2})A$
6	$(\frac{n(n-1)}{2})A$
7	$(\frac{n(n-1)}{2})A$
TOTAL	$1A+\mathrm{n}(\mathrm{I}+\mathrm{C})+\left(\frac{n^2-n}{2}\right)(I+C)+(I+C)+\left(\frac{n^2-n}{2}\right)(3A+C)$
ESTIMATIVA (O)	$O(n^2)$

Método isVertexInSet

Este método verifica se um determinado vértice está presente dentro de um Set (dentro de um “saco” de vértices)

Set: list of String representa um “saco” que contem o nome dos vertices.

Vertex: String é o nome do vértice que queremos verificar se está presente no set.

Pseudocódigo:

LINHA	PSEUDOCÓDIGO
-	procedure isVertexInSet(set: list of String, vertex: String)
1	for i := 0 to size(set) - 1
2	if set[i] = vertex
3	return true
4	return false

Análise da Complexidade:

LINHA	ALGORITMO ISVERTEXINSET
1	(n+1)I + (n+1)C
2	(n)C
3	1R (No pior caso, a última ser verdadeira)
4	1R
TOTAL	(n+1)I+(2n+1)C+2R
ESTIMATIVA (O)	O(n)

Algoritmo de Dijkstra – US17:

Este algoritmo calcula o caminho de custo mínimo de um vértice inicial para todos os outros vértices de um grafo. Inicializamos uma lista de distâncias com infinito, exceto a distância do vértice inicial, que é zero, e um conjunto de vértices visitados, inicialmente vazio. Iteramos pelos vértices, escolhendo o vértice com a menor distância conhecida, colocando-o nos visitados e examinamos as arestas adjacentes.

Para cada aresta, se o destino ainda não foi visitado, calculamos a distância potencial até o vértice de destino através do vértice atual e, se for menor que a distância registrada, atualizamos a distância e o predecessor.

Repetimos até que todos os vértices sejam visitados, o resultado é uma lista de distâncias mínimas do vértice inicial para os outros vértices todos.

Pseudocódigo:

G é de um tipo de dados abstrato que contêm uma lista de vértices e arestas;

d[] é um conjunto de números inteiros e representa a distancia de um Vértice até ao AP.

p[] é um conjunto de Vértices que representa o vértice anterior a outro certo vértice;

s é considerado o Vértice inicial, neste caso é o nosso AP

o método “size()” é nativo ao Java e obtem o tamanho de uma lista, ou seja, itera por uma lista e retorna o count de elementos dessa lista, tendo complexidade o número de elementos da lista (no pior caso) ou seja O(n)

LINHA	PSEUDOCÓDIGO
-	Procedure dijkstra(G; d[1],d[2],..., d[n] : integer; p[1],p[2],..., p[n] : V; s : V)
1	for i:= 0 to size(G.vertexes):
2	d[i] := infinite
3	p[i] := null
4	d[s] := 0
5	p[s] := s
6	visited := []
7	visiting := s
8	while size(visited) ≠ to size(G.vertexes):
9	adjacentEdges[] := getEdgesFromVertex(G, visiting)
10	for i := 0 to size(adjacentEdges):
11	destination := adjacentEdges[i].destinationVertex
12	if destination not in visited:
13	if d[destination] > d[visiting] + adjacentEdges[i].weight:

14	d[destination] := d[visiting] + adjacentEdges[i].weight
15	p[destination] := visiting
16	visited.add(visiting)
17	visiting := getMinDistanceVertex(G, d, visited)

Análise da Complexidade

LINHA	ALGORITMO DIJKSTRA
1	(n + 1)A
2	nA
3	nA
4	1A
5	1A
6	1A
7	1A
8	(n + 1)C
9	$O(n) * n = O(n^2)$ → Algoritmo getEdgesFromVertex
10	(n + 1)C + (n + 1)R
11	nA
12	nC
13	nC + nOp
14	nC + nOp
15	nA
16	1A
17	$O(n) * n = O(n^2)$ → Algoritmo getMinDistanceVertex
TOTAL	$O(2n^2 + n)$

Método getMinDistanceVertex

Este método retorna um vértice não visitado do grafo onde a distancia calculada até ao source é a menor registada.

Pseudocódigo:

LINHA	PSEUDOCÓDIGO
-	Procedure getMinDistanceVertex(G, distance, visited)
1	nextVertex := null
2	for vertex in G.vertexes:
3	if vertex not in visited:
4	if nextVertex is null:
5	nextVertex := vertex
6	if distance[vertex] < distance[nextVertex] then
7	nextVertex := vertex
8	return nextVertex

Análise da Complexidade:

LINHA	ALGORITMO GETMINDISTANCEVERTEX
1	1A
2	(n + 1)C + (n + 1)A
3	nC
4	nC
5	nA
6	nC
7	nA
8	1R
TOTAL	O(n)

Método getEdgesFromVertex

Este método, retorna uma lista de todas as arestas adjacentes a um determinado vertice do grafo.

Pseudocódigo

LINHA	PSEUDOCÓDIGO
-	Procedure getEdgesFromVertex(edges, vertex)
1	result := []
2	for edge in edges:
3	if edge.vertexFrom equals vertex or edge.vertexTo equals vertex:
4	result.add(edge)
5	return result

Análise da Complexidade

LINHA	ALGORITMO GETEDGESFROMVERTEX
1	1A
2	(n + 1)C + (n + 1)A
3	2nC
4	nA
5	1R
TOTAL	O(n)

Algoritmo de Dijkstra – US18:

Pseudocódigo

G é de um tipo de dados abstrato que contêm uma lista de vértices e arestas;

d[] é um conjunto de números inteiros e representa a distancia de um Vértice até um AP.

p[] é um conjunto de Vértices que representa o vértice anterior a outro certo vértice;

dd[] é o conjunto de números inteiros que representam a distancia definitiva de um vértice a um certo AP.

pd[] é o conjunto de Vértices que representa o vértice anterior definitivo a outro certo vértice para obter o caminho para um certo AP.

s[] é o conjunto de vértices considerados APs

LINHA	PSEUDOCÓDIGO
-	Procedure getDistFromMultipleAP(G; d[1], d[2],..., d[n] : integer; p[1], p[2],..., p[n] : V ; dd[1],dd[2],..., dd[n] : integer; pd[1],pd[2],..., pd[n] : V; s[1], s[2],...,s[n] : V)
1	for i := 0 to n do
2	dijkstra(G, d[], p[], s[i])
3	for j:= 0 to size(G.vertexes) do
4	if dd[j] = null or dd[j] > d[j] :
5	dd[j] := d[j]
6	pd[j] := p[j]

Análise da Complexidade

LINHA	ALGORITMO GETDISTFROMMULTIPLEAP
1	(n + 1)C
2	$O(n^2 + (n/2)) * n = O(n^3 + (n^2/2))$ → Algoritmo dijkstra
3	(n + 1)C
4	nC
5	nA
6	nA
TOTAL	$O(n^3 + n^2 + 5n)$