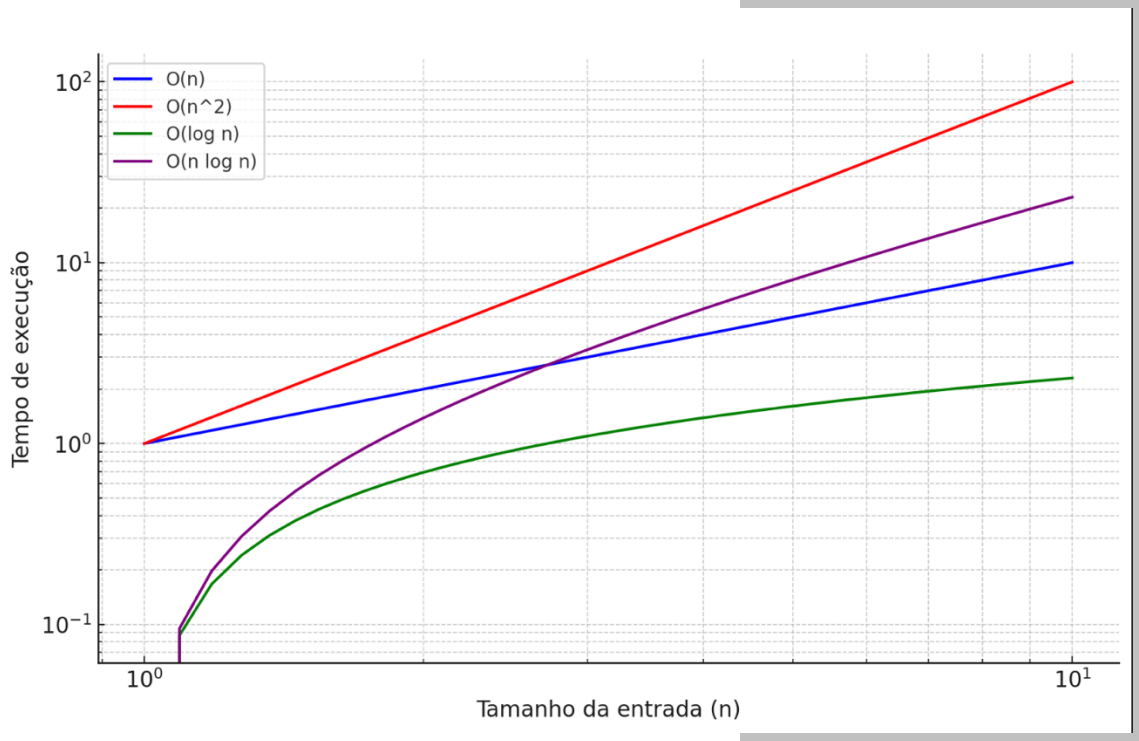
DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

# Análise Temporal de Algoritmos




**9 de junho de 2024**


Paulo Pereira 1221967
Manuel Marques 1201933
Rui Pinto 1230401
Paulo Silva 1120629
Alexandre Guerra 1220961


ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

# Índice

# Introdução

A complexidade de um algoritmo é a quantidade de trabalho necessária (temporal e espacial) para a sua execução, que depende do algoritmo e do tamanho do input.

Ao estudarmos a complexidade algorítmica pretendemos estimar quais os recursos necessários para um algoritmo ser implementado à medida que o tamanho do problema cresce. Além disso, dado um problema, permite-nos escolher, entre os vários algoritmos que resolvem esse problema, qual deles é o mais eficiente e ainda poder desenvolver novos algoritmos mais eficientes para problemas que já têm solução (Moura, 2023).

A análise temporal de algoritmos é, por isso, uma técnica essencial utilizada na teoria da computação para medir o tempo de execução de um algoritmo em função do tamanho da entrada de dados. Este tipo de análise é fundamental para avaliar a eficiência e a viabilidade prática de algoritmos, especialmente quando lidamos com grandes volumes de dados.

Para descrever o comportamento do tempo de execução à medida que o tamanho da entrada cresce, utilizamos a notação assintótica.

A notação e caso de análise utilizados na presente análise serão a “O-notation” ou (Big O) que representa o limite superior do tempo de execução, que terá por referência o cenário de“Pior caso”.

O método de análise consistirá na contagem do número de operações básicas realizadas pelo algoritmo.

O procedimento em estudo será a aplicação de um algoritmo de Dijkstra, onde se pretendem fazer estudos de cálculo de rotas de menor custo para a aplicação de sinalética de evacuação de emergência para parques urbanos.

Esta exigência é espoletada no decorrer das atividades letivas do Curso de Engenharia Informática do Instituto Superior de Engenharia do Porto, mais concretamente das unidades curriculares de Matemática Discreta e Laboratório/Projeto II.

# US13

| Linha | procedure: main | |
|---|---|---|
| 1 | *bubbleSort* (edges) | $O(n^2)$ |
| 2 | mst := *KruskalMST* (edges) | $1A + O(n^2)$ |
| 3 | *printMST* (mst, edges) | $O(n)$ |
| 4 | outputGraph_TXTFile := *outputTXTFileNameOf* (workFile, *GRAPH_STRING*) | $O(n)$ |
| 5 | outputMST_TXTFile := *outputTXTFileNameOf* (workFile, *MST_STRING*) | $O(n)$ |
| 6 | *printGraphToTXTFile* (edges, outputGraph_TXTFile) | $O(n)$ |
| 7 | *printGraphToTXTFile* (mst, outputMST_TXTFile) | $O(n)$ |
| 8 | outputGraph_SVGFile = *outputSVGFileNameOf* (outputGraph_TXTFile, *IMAGE_EXTENSION_STRING*) | $O(n)$ |
| 9 | outputMST_SVGFile = *outputSVGFileNameOf* (outputMST_TXTFile, *IMAGE_EXTENSION_STRING*) | $O(n)$ |
| 10 | *plotGraph* (outputGraph_TXTFile, outputGraph_SVGFile) | |
| 11 | *plotGraph* (outputMST_TXTFile, outputMST_SVGFile) | |
| total | | $2n^2 + 7n + 1$ |
| Estimativa O | | $O(n^2)$ |

| Linha | procedure: BubbleSort(graph[1], graph[2], ..., graph[n]) | |
|---|---|---|
| 1 | c:= n | 1A |
| 2 | aux | |
| 3 | for i := 0 to c-1 | nA + nC |
| 4 | for j:= 0 to c - i - 1 | n(nA-nC)=$n^2$A+$n^2$C |
| 5 | if graph[j].cost > graph[j + 1].cost then | n(n-1)C=($n^2$-n)C |
| 6 | aux := graph[j] | n(n-1)A=($n^2$-n)A |
| 7 | graph[j] := graph[j + 1] | n(n-1)A=($n^2$-n)A |
| 8 | graph[j + 1] := aux | n(n-1)A=($n^2$-n)A |
| total | | $6n^2 - 2n + 1$ |
| Estimativa O | | $O(n^2)$ |

| Linha | procedure: kruskalMST(graph[1], graph[2], ..., graph[n]) | |
|---|---|---|
| 1 | vertexList := *getVertexList* (graph) | $O(n^2)$ |
| 2 | result | |
| 3 | e := 0 | 1A |
| 4 | uf | |
| 5 | while e < graph.size and result.size < vertexList.size | n(3C) |
| 6 | next_edge := graph[e] | n * 1A |
| 7 | x := find(*getVertexIndex* (next_edge.src, vertexList)) | n * O(n) |
| 8 | int y = uf.find(*getVertexIndex* (next_edge.dest, vertexList)); | n(1A + O(n)) |
| 9 | if (x != y) { | n * 1C |
| 10 | result.add(next_edge); | n * 1A |
| 11 | uf.union(x, y); | n * O(n) |
| 12 | e++; | n * 1A |
| 13 | return result; | 1R |
| total | | $4n^2 + 6n + 2$ |
| Estimativa O | | $O(n^2)$ |

| Linha | procedure: printMST(mst[1 to n]:Edges, edges[1 to n]:edges) { | |
|---|---|---|
| 1 | cost := 0 | 1A |
| 2 | for each edge of mst | (n+1)C + (n+1)A |
| 3 | print(edge.source -- edge.destination = edge.cost) | nI |
| 4 | cost := cost + e.cost | nA + nS |
| 5 | print(Graph Dimension = edges.size() : Graph Order = mst.size() : Minimum cost = cost) | 1I |
| total | | $5n + 4$ |
| Estimativa O | | $O(n)$ |

| Linha | procedure: outputTXTFileNameOf(workFile: String; type:String) | |
|---|---|---|
| 1 | inputFile := workFile.split("/")[1] * | 1A + O(n) |
| 2 | outputFile := inputFile.split(".")[0] + type + *OUTPUT_TXT_EXTENSION* * | 1A + O(n) |
| 3 | return outputFile | 1R |
| total | | $2n + 3$ |
| Estimativa O | | $O(n)$ |

* foi usada uma função de bilbioteca Java, no entanto, sabe-se pela documentação que esta tem complexidade O(n)

| Linha | procedure: printGraphToTXTFile(edges:Edges; filename:String) | |
|---|---|---|
| 1 | outputFolderAndFile := OUTPUT_FOLDER + "/" + filename | 1A |
| 3 | printToFile := PrintWriter(outputFolderAndFile); | 1A |
| 4 | printToFile(graph filename { | 1I |
| 5 | for edge of edges | (n+1)A + (n+1)C |
| 6 | printToFile(edge.source -- edge.destination [weight=edge.cost]) | nI |
| 7 | printToFile(}); | 1I |
| total | | $3n + 6$ |
| Estimativa O | | $O(n)$ |

| Linha | procedure: outputSVGFileNameOf(outputGraphTxtFile:String; svgExtensionString:String) | |
|---|---|---|
| 1 | return outputGraphTxtFile.split(".")[0] + svgExtensionString * | O(n) + 1A + 1R |
| total | | $n + 2$ |
| Estimativa O | | $O(n)$ |

* foi usada uma função de bilbioteca Java, no entanto, sabe-se pela documentação que esta tem complexidade O(n)

| Linha | procedure: plotGraph(String inputFile, String outputFile) | |
|---|---|---|
| 1 | ProcessBuilder pb1 = new ProcessBuilder("dot", "-Tsvg", *OUTPUT_FOLDER + UNIX_DIR* | |
| 2 | pb1.inheritIO(); | |
| 3 | Process process1 = pb1.start(); | |
| 4 | process1.waitFor(); | |
| total | | |
| Estimativa O | | |

| Linha | procedure: getVertexList(edges[1], edges[2], ..., edges[n]) | |
|---|---|---|
| 1 | auxVertexList | |
| 2 | for i:=1 to n | (n+1)A + (n+1)C |
| 3 | auxVertexList:=auxVertexList + edges[i].souce | nA |
| 4 | auxVertexList:=auxVertexList + edges[i].destination | nA |
| 5 | vertexList := *removeDuplicates* (auxVertexList) | $O(n^2)$ |
| 6 | sort(vertexList) * | O(n) |
| 7 | return vertexList | 1R |
| total | | $n^2 + 5n + 3$ |
| Estimativa O | | $O(n^2)$ |

* foi usada uma função de bilbioteca Java, no entanto, sabe-se pela documentação que esta tem complexidade O(n)

| Linha | procedure: removeDuplicates(vertexList[1], vertexList[2],...,vertexList[n]) | |
|---|---|---|
| 1 | vertexListAux | |
| 2 | for i:=1 to n | (n+1)A + (n+1)C |
| 3 | if not (vertexList.contains(vertexList[i])) * | n * O(n) |
| 4 | vertexListaux:=vertexListAux+vertexList[i] | nA |
| 5 | return vertexListAux | 1R |
| total | | $n^2 + 3n + 3$ |
| Estimativa O | | $O(n^2)$ |

* foi usada uma função de bilbioteca Java, no entanto, sabe-se pela documentação que esta tem complexidade O(n)

| Linha | procedure: union(x, y:integer; parent[1], parent[2],..., parent[n]) | |
|---|---|---|
| 1 | rootX := find(x) | O(n) |
| 2 | rootY := find(y) | O(n) |
| 3 | parent[rootX] := rootY | 1A |
| total | | $2n + 1$ |
| Estimativa O | | $O(n)$ |

| Linha | procedure find(x:integer; parent[1], parent[2],..., parent[n]) | |
|---|---|---|
| 1 | if parent[x] <> x | 1C |
| 2 | parent[x] := find(parent[x])* | 1A + O(n)C |
| 3 | return parent[x] | 1R |
| total | | $n+3$ |
| Estimativa O | | $O(n)$ |

* foi usada um método recursivo, no entanto, assume-se que este tem complexidade O(n)

| Linha | procedure: getVertexIndex(vertex: string, vertexList[1], vertexList[2], ..., vertexList[n]) | |
|---|---|---|
| 1 | return vertexList.indexOf(vertex) | 1R + O(n) |
| total | | 1R + O(n) |
| Estimativa O | | O(n) |

* foi usada uma função de bilbioteca Java, no entanto, sabe-se pela documentação que esta tem complexidade O(n)

# US17

| Linha | Procedure: presentOptions() | |
|---|---|---|
| 1 | verticesNames := *readPointNamesData* (*"datasets/us17_points_names.csv"*) | $O(n)$ |
| 2 | matrix = *readEdgesData* (*"datasets/us17_matrix.csv"*) | $O(n^2)$ |
| 3 | *printPointAndMatrixData* (verticesNames, matrix) | $O(n^2)$ |
| 4 | *printInitalGraphToTXTFile* (matrix, verticesNames, *"us17_dot"* + ".txt") | $O(n^2)$ |
| 5 | *plotGraph* (*"us17_dot"* + *".txt"*, "us17_initial_graph" + *".svg"*) | |
| 6 | opt := "" | 1A |
| 7 | print("Choose one of the following options:") | 1I |
| 8 | print("1 - All routes") | 1I |
| 9 | print("2 - Choose a sign") | 1I |
| 10 | print("0 - Exit") | 1I |
| 11 | read := Scanner(System.*in* ) | 1A |
| 12 | do | 1C |
| 13 | opt = read.nextLine(); | 1A |
| 14 | if opt <> "1" and opt <> "2" and opt <> "0" | 3C |
| 15 | print("Insert the right option!"); | 1I |
| 16 | while opt <> "1" and opt <> "2" and opt <> "0" | |
| 17 | switch (opt) | 1C |
| 18 | case "1": | |
| 19 | *proceedToAllRoutes* (verticesNames, matrix) | $O(n^3)$ |
| 20 | case "2": | |
| 21 | *proceedToOneRoute* (verticesNames, matrix) | $O(n^2)$ |
| total | | $n^3 + 4n^2 + n + 13$ |
| Estimativa O | | $O(n^3)$ |

| Linha | procedure: readPointNamesData(points: String) | |
|---|---|---|
| 1 | pointNames := null | 1A |
| 2 | readFile = Scanner(File(points)) | 1A |
| 3 | print("File found!") | 1I |
| 4 | if (readFile.hasNext()) | 1C |
| 5 | pointNames = readFile.nextLine().split(";") | 1A |
| 6 | if (pointNames != null) | 1C |
| 7 | for i := 0 to pointNames.length | (n+1)A + (n+1)C |
| 8 | pointNames[i] := pointNames[i].replace("\uFEFF", "") | nA |
| 9 | return pointNames | 1R |
| total | | $3n + 9$ |
| Estimativa O | | $O(n)$ |

| Linha | procedure: readEdgesData(matrix: String) | |
|---|---|---|
| 1 | edgesDataAux | |
| 2 | auxStrVector | |
| 3 | auxFltVector | |
| 4 | i := 0 | 1A |
| 5 | readFile = Scanner(File(matrix)) | 1A |
| 6 | print("File found!") | 1I |
| 7 | while (readFile.hasNext()) do | (n+1)C |
| 8 | auxStrVector := readFile.nextLine().split(";") | nA |
| 9 | auxFltVector := new double[auxStrVector.length] | nA |
| 10 | for j := 0 to auxStrVector.length | n((n+1)A +(n+1)C) |
| 11 | auxFltVector[j] := Double.*parseDouble* (auxStrVector[j].replace("\uFEFF", "")) | n(nA) |
| 12 | edgesDataAux := edgesDataAux + auxFltVector | nA |
| 13 | i := i +1 | nA |
| 17 | return edgesDataAux | 1R |
| total | | $3n^2 + 5n + 5$ |
| Estimativa O | | $O(n^2)$ |

| Linha | procedure: printPointAndMatrixData(pointNamesData[1 to n]:String; matrix[1 to n][1 to n]: double | |
|---|---|---|
| 1 | print("/") | 1I |
| 2 | for i := to pointNamesData.length | (n+1)A + (n+1)C |
| 3 | print(pointNamesData[i]) | nI |
| 4 | print("\n") | 1I |
| 5 | for i := 0 to matrix.length | (n+1)A + (n+1)C |
| 6 | print(pointNamesData[i]) | nI |
| 7 | for j := 0 to matrix[i].length | n((n+1)A + (n+1)C) |
| 8 | print(matrix[i][j]) | n(nI) |
| 9 | print("\n") | nI |
| total | | $3n^2 + 7n + 7$ |
| Estimativa O | | $O(n^2)$ |

| Linha | procedure: printInitalGraphToTXTFile(graphWeights[1 to n][1 to n]:double; vertices[1 to n]:String; filename: String) | |
|---|---|---|
| 1 | outputFolderAndFile = *"output-files"* + *"/"* + filename | 1A |
| 2 | printToFile = PrintWriter(File(outputFolderAndFile)); | 1A |
| 3 | printToFile("graph filename {\n") | 1I |
| 4 | for i := 0 to graphWeights.length | (n+1)A + (n+1)C |
| 5 | for j := 0 to i | n((n+1)A + (n+1)C) |
| 6 | if graphWeights[i][j] > 0 | n(nC) |
| 7 | line = "vertices[i] -- vertices[j] [label=graphWeights[i][j]]" | n(nA) |
| 8 | printToFile(line) | n(nI) |
| 9 | printToFile("\n}") | 1I |
| total | | $5n^2 + 4n + 6$ |
| Estimativa O | | $O(n^2)$ |

| Linha | procedure: plotGraph(inputFile:String; outputFile:String) | |
|---|---|---|
| 1 | ProcessBuilder pb1 = new ProcessBuilder("dot", "-Tsvg", *OUTPUT_FOLDER + UNIX_DI* | |
| 2 | pb1.inheritIO(); | |
| 3 | Process process1 = pb1.start(); | |
| 4 | process1.waitFor(); | |
| total | | |
| Estimativa O | | |

| Linha | procedure: proceedToOneRoute(verticesNames[1 to n]:String;matrix[1 to n][1 to n]:double) | |
|---|---|---|
| 1 | *printPointAndMatrixData* (verticesNames, matrix) | $O(n^2)$ |
| 2 | read = Scanner(System.*in* ) | 1A |
| 3 | vertices = List.*of*(verticesNames) * | 1A + O(n) |
| 4 | print("\nChoose the initial sign?") | 1I |
| 5 | vertex := read.nextLine() | 1A |
| 6 | if vertices.contains(vertex) | 1C |
| 7 | *us17Routine* (matrix, verticesNames, vertex); | $O(n^2)$ |
| 8 | else | |
| 9 | print("Vertex not found!") | 1I |
| total | | $2n^2 + n + 6$ |
| Estimativa O | | $O(n^2)$ |

* foi usada uma função de bilbioteca Java, no entanto, sabe-se pela documentação que esta tem complexidade O(n)

| Linha | procedure: us17Routine(matrix[1 to n][1 to n]; verticesNames[1 to n]; vertex:String) | |
|---|---|---|
| 1 | copy = *deepCopyMatrixDouble* (matrix) | $O(n^2)$ |
| 2 | marcas[1 to n] | |
| 3 | antecessor[1 to n] | |
| 4 | *applyDijkstraAlg* (copy, verticesNames, *"AP"*, marcas, antecessor) | $O(n^2)$ |
| 5 | print(*printShortestRoutesToCSV* (verticesNames, antecessor, marcas, vertex, "dijkstra_us17") | $O(n^2)$I |
| 6 | *printGraphToTXTFile* (matrix, verticesNames, antecessor, *"us17_dot"*+ *".txt"*, vertex) | $O(n^2)$ |
| 7 | *plotGraph* (*"us17_dot"* + *".txt"*, "point_" + vertex + *".svg"*); | |
| total | | $4n^2$ |
| Estimativa O | | $O(n^2)$ |

| Linha | procedure: deepCopyMatrixDouble(matrix[1 to n][1 to n]) | |
|---|---|---|
| 1 | aux[1 to n][1 to n] | |
| 2 | for i := 0 to matrix.length | (n+1)A + (n+1)C |
| 3 | for j := 0 to matrix[i].length | n((n+1)A + (n+1)C) |
| 4 | aux[i][j] := matrix[i][j] | n(nA) |
| 5 | return aux | 1R |
| total | | $3n^2 + 4n + 3$ |
| Estimativa O | | $O(n^2)$ |

| Linha | procedure: applyDijkstraAlg(matriz[1 to n][1 to n]:double; vertices[1 to n]:String; pInicial:String; marcas[1 to n]:double; antecessor[1 to n]:String) | |
|---|---|---|
| 1 | visitados[1 to n]:boolean | |
| 2 | for i := 0 to vertices.length | (n+1)A + (n+1)C |
| 3 | marcas[i] := Double.*MAX_VALUE* | nA |
| 4 | visitados[i] := false | nA |
| 5 | antecessor[i] := "-" | nA |
| 6 | indice1 := *posicaoVertice* (vertices, pInicial) | 1A + O(n) |
| 7 | indice2 := *caminhoMaisCurto* (indice1, matriz) | 1A + O(n) |
| 8 | marcas[indice1] := 0 | nA |
| 9 | while indice1 >= 0 do | (n+1)C |
| 10 | if indice2 >= 0 | nC |
| 11 | if marcas[indice2] > marcas[indice1] + matriz[indice1][indice2] and NOT(visitados[indice1] | nC + nC + nC |
| 12 | marcas[indice2] := marcas[indice1] + matriz[indice1][indice2] | nA + nS |
| 13 | matriz[indice1][indice2] := 0 | nA |
| 14 | matriz[indice2][indice1] := 0 | nA |
| 15 | antecessor[indice2] := vertices[indice1] | nA |
| 16 | indice1 := *marcaMinima* (marcas, visitados) | nO(n) |
| 17 | indice2 := *caminhoMaisCurto* (indice1, matriz) | nO(n) |
| 18 | else | |
| 19 | matriz[indice1][indice2] := 0 | nA |
| 20 | matriz[indice2][indice1] := 0 | nA |
| 21 | indice1 := *marcaMinima* (marcas, visitados) | nO(n) |
| 22 | indice2 := *caminhoMaisCurto* (indice1, matriz) | nO(n) |
| 23 | else | |
| 24 | visitados[indice1] := true; | nA |
| 25 | indice1 := *marcaMinima* (marcas, visitados); | nO(n) |
| 26 | if indice1 >= 0 | nC |
| 27 | indice2 := *caminhoMaisCurto* (indice1, matriz) | nO(n) |
| total | | $6n^2 + 22n + 5$ |
| Estimativa O | | $O(n^2)$ |

| Linha | procedure: printShortestRoutesToCSV(vertices[1 to n]:String; previousVertices[1 to n]:String; weights[1 to n]:double; vertex:String; fileName:String) | |
|---|---|---|
| 1 | csvLine | |
| 2 | requested := "" | 1A |
| 3 | csvLines[]:String | |
| 4 | for each v in vertices | (n+1)A + (n+1)C |
| 5 | csvLine := *getCsvLine* (v, vertices, previousVertices, weights) | $1A + O(n^2)$ |
| 6 | if v = vertex | nC |
| 7 | requested := csvLine | nA |
| 8 | csvLines := csvLines + csvLine | nA |
| 9 | *printDataToFile* (csvLines, fileName) | nO(n) |
| 10 | return requested | 1R |
| total | | $2n^2 + 5n + 5$ |
| Estimativa O | | $O(n^2)$ |

| Linha | procedure: printGraphToTXTFile(graphWeights[1 to n][1 to n]:double; vertices[1 to n]:String; antecessores[1 to n]:String; filename:String; vertice:String) | |
|---|---|---|
| 1 | verticesF[1 to vertices.length]:String | |
| 2 | antecessoresF[1 to antecessores.length]:String | |
| 3 | *filtraCaminhoMaisCurtoDoVertice* (verticesF, antecessoresF, vertices, antecessores, vertice) | $O(n^2)$ |
| 4 | line:String | |
| 5 | outputFolderAndFile := "output-files" + "/" + filename) | 1A |
| 7 | printToFile = PrintWriter(File(outputFolderAndFile)) | |
| 8 | printToFile("graph filename {\n") | 1I |
| 9 | for i := 0 to graphWeights.length | (n+1)A + (n+1)C |
| 10 | for j := 0 to i | n(n+1) |
| 11 | if graphWeights[i][j] > 0 | n(nC) |
| 12 | if *isPresent* (i, j, verticesF, antecessoresF) | n(nC) |
| 13 | line = "vertices[i] -- vertices[j] [label=graphWeights[i][j]][color=red]" | n(nA) |
| 14 | else | |
| 15 | line = "vertices[i] -- vertices[j] [label=graphWeights[i][j]]" | |
| 16 | printToFile(line) | n(nI) |
| 17 | printToFile("}") | 1I |
| total | | $7n^2 + 3n + 5$ |
| Estimativa O | | $O(n^2)$ |

| Linha | procedure: proceedToAllRoutes(String[] verticesNames, double[][] matrix) { | |
|---|---|---|
| 1 | for each v in verticesNames | (n+1)A + (n+1)C |
| 2 | *us17Routine* (matrix, verticesNames, v) | $nO(n^2)$ |
| total | | $n^3$ |
| Estimativa O | | $O(n^3)$ |

| Linha | procedeure: posicaoVertice(vertices[1 to n]:String; pInicial:String) | |
|---|---|---|
| 1 | for i := 0 to vertices.length | (n+1)A + (n+1)C |
| 2 | if vertices[i] = pInicial | nA |
| 3 | return i; | 1R |
| 4 | return -1; | |
| total | | $3n + 3$ |
| Estimativa O | | $O(n)$ |

| Linha | procedure: caminhoMaisCurto(indice1:integer; matriz[1 to n][1 to n]) | |
|---|---|---|
| 1 | valAux := Double.*MAX_VALUE* | 1A |
| 2 | index := -1 | 1A |
| 3 | for i := 0 to matriz[indice1].length | (n+1)A + (n+1)C |
| 4 | if matriz[indice1][i] < valAux and matriz[indice1][i] > 0 | nC + nC |
| 5 | valAux := matriz[indice1][i] | nA |
| 6 | index := i | nA |
| 7 | return index | 1R |
| total | | $6n + 5$ |
| Estimativa O | | $O(n)$ |

| Linha | procedure: marcaMinima(marcas[1 to n]:double; visitados[1 to n]:boolean) | |
|---|---|---|
| 1 | valAux := Double.*MAX_VALUE* | 1A |
| 2 | index := -1 | 1A |
| 3 | for i := to marcas.length | (n+1)A + (n+1)C |
| 4 | if marcas[i] <= valAux and NOT(visitados[i]) | nC + nC |
| 5 | valAux := marcas[i] | nA |
| 6 | index := i | nA |
| 7 | return index | 1R |
| total | | $6n + 5$ |
| Estimativa O | | $O(n)$ |

| Linha | procedure: getCsvLine(v:String; vertices[1 to n]:String; antecessores[1 to n]:String; marcas[1 to n]:Double) | |
|---|---|---|
| 1 | csvLine := *"("* | 1A |
| 2 | index := *posicaoVertice* (vertices, v) | 1A + O(n) |
| 3 | custo := marcas[index] | 1A |
| 4 | antecessor:String | |
| 5 | do | nC |
| 6 | antecessor := antecessores[index] | nA |
| 7 | csvLine := csvLine + vertices[index] | nA |
| 8 | if antecessor <> "-" | nC |
| 9 | csvLine := csvLine + "," + " " | nA |
| 10 | index := *posicaoVertice* (vertices, antecessor) | nO(n) |
| 11 | while antecessor <> "-" | |
| 12 | return csvLine + ")" + ";" + custo | 1R + 1A |
| total | | $n^2 + 6n + 5$ |
| Estimativa O | | $O(n^2)$ |

| Linha | procedure: printDataToFile(csvLines[1 to n]:String; name:String) | |
|---|---|---|
| 1 | prtToFile = PrintWriter(File(*"output-files"* + *"/"* + name + *".csv"*)) | 1A |
| 2 | for each line in csvLines | (n+1)A + (n+1)C |
| 3 | prtToFile(line) | nI |
| total | | $3n + 3$ |
| Estimativa O | | $O(n)$ |

| Linha | private static boolean isPresent(int i, int j, String[] vertices, String[] antecessores) { | |
|---|---|---|
| 1 | return vertices[i] = antecessores[j] ou vertices[j] = antecessores[i] | 1R + 2C |
| total | | 3 |
| Estimativa O | | O(1) |

| Linha | procedure: filtraCaminhoMaisCurtoDoVertice(verticesF[1 to n]String; antecessorF[1 to n]:String; vertices[1 to n]:String; antecessores[1 to n]:String; vertice:String) | |
|---|---|---|
| 1 | index := *posicaoVertice* (vertices, vertice) | 1A + O(n) |
| 2 | String antecessor | |
| 3 | do | nC |
| 4 | antecessor = antecessores[index] | nA |
| 5 | verticesF[index] = vertices[index] | nA |
| 6 | antecessorF[index] = antecessores[index] | nA |
| 7 | index = *posicaoVertice* (vertices, antecessor) | nA + nO(n) |
| 8 | while antecessor <> "-" | |
| total | | $n^2 + 6n + 1$ |
| Estimativa O | | $O(n^2)$ |

# US18

| Linha | Procedure: presentOptions() | |
|---|---|---|
| 1 | verticesNames := *readPointNamesData* (*"datasets/us18_points_names.csv"*) | $O(n)$ |
| 2 | matrix = *readEdgesData* (*"datasets/us18_matrix.csv"*) | $O(n^2)$ |
| 3 | *printPointAndMatrixData* (verticesNames, matrix) | $O(n^2)$ |
| 4 | *printInitalGraphToTXTFile* (matrix, verticesNames, *"us17_dot"* + ".txt") | $O(n^2)$ |
| 5 | *plotGraph* (*"us17_dot"* + *".txt"*, "us17_initial_graph" + *".svg"*) | |
| 6 | opt := "" | 1A |
| 7 | print("Choose one of the following options:") | 1I |
| 8 | print("1 - All routes") | 1I |
| 9 | print("2 - Choose a sign") | 1I |
| 10 | print("0 - Exit") | 1I |
| 11 | read := Scanner(System.*in* ) | 1A |
| 12 | do | 1C |
| 13 | opt = read.nextLine(); | 1A |
| 14 | if opt <> "1" and opt <> "2" and opt <> "0" | 3C |
| 15 | print("Insert the right option!"); | 1I |
| 16 | while opt <> "1" and opt <> "2" and opt <> "0" | |
| 17 | switch (opt) | 1C |
| 18 | case "1": | |
| 19 | *proceedToAllVertices* (verticesNames, matrix) | $O(n^4)$ |
| 20 | case "2": | |
| 21 | *proceedToOneVertex* (verticesNames, matrix) | $O(n^3)$ |
| total | | $n^4 + n^3 + 3n^2 + n + 13$ |
| Estimativa O | | $O(n^4)$ |

| Linha | procedure: us18Routine(verticesNames[1 to n]:String; matrix[1 to n][1 to n]:double; vertice:String) | |
|---|---|---|
| 1 | pathAndCost:String | |
| 2 | smalestPaths[]:String | |
| 3 | smalestCosts[]:double | |
| 4 | workedMatrixes[][][]:double | |
| 5 | workedVnames[][]:String | |
| 6 | workedPrevVertixes[][]:String | |
| 7 | apIndexes[]:integer = *identifyAP_indexes* (verticesNames) | 1A + O(n) |
| 8 | for each i in apIndexes | (n+1)A + (n+1)C |
| 9 | apIndexesAux := List.*copyOf* (apIndexes) * | n(1A + O(n)) |
| 10 | apIndexesAux.remove(i) * | nO(n) |
| 11 | workingMatrix[][]:double = *removeUnwantedMatrixIndexes* (apIndexesAux, matrix) | $n(1A + O(n^2))$ |
| 12 | workingVNames[]:String=*removeUnwantedVerticesIndexes* (apIndexesAux, verticesNames) | n(1A + O(n)) |
| 13 | copy[][]:double := *deepCopyMatrixDouble* (workingMatrix) | $n(1A + O(n^2))$ |
| 14 | weights[workingVNames.length]:double | |
| 15 | prevVertices[workingVNames.length]:String | |
| 16 | *applyDijkstraAlg* (copy, workingVNames, verticesNames[i], weights, prevVertices) | $nO(n^2)$ |
| 17 | pathAndCost := *printShortestRoutesToCSV* (workingVNames, prevVertices, weights, vertice, *"dijkstra_us18"* + "_" + verticesNames[i]); | $nO(n^2)$ |
| 18 | cost:double = Double.*parseDouble* (pathAndCost.split(";")[1].trim().replace(",", ".")) * | n(1A + O(n)) |
| 19 | smalestPaths := smalestPaths + pathAndCost | n(1A) |
| 20 | smalestCosts := smalestCosts + cost | n(1A) |
| 21 | workedMatrixes := workedMatrixes + workingMatrix | n(1A) |
| 22 | workedVnames := workedVnames + workingVNames | n(1A) |
| 23 | workedPrevVertixes := workedPrevVertixes + prevVertices | n(1A) |
| 24 | smalestIndex:integer = *getSmallestIndex* (smalestCosts) | 1A + O(n) |
| 25 | print(smalestPaths[smalestIndex]) | |
| total | | $4n^3 + 4n^2 + 14n + 4$ |
| Estimativa O | | $O(n^3)$ |

* foi usada uma função de bilbioteca Java, no entanto, sabe-se pela documentação que esta tem complexidade O(n)

| Linha | procedure: removeUnwantedMatrixIndexes(apIndexes[1 to n]:integer; matrix[1 to n][1 to n]) | |
|---|---|---|
| 1 | newSize := matrix.length - apIndexes.size() | 1A + 1S |
| 2 | aux[newSize][newSize]:double | |
| 3 | newRow:integer := 0 | 1A |
| 4 | newCol:integer | |
| 5 | for i := 0 to matrix.length | (n+1)A + (n+1)C |
| 6 | if NOT(apIndexes.contains(i)) | nC |
| 7 | newCol := 0 | nA |
| 8 | for j := 0 to matrix[0].length | n((n+1)A + (n+1)C) |
| 9 | if NOT(apIndexes.contains(j)) | n(nC) |
| 10 | aux[newRow][newCol] := matrix[i][j] | n(nA) |
| 11 | newCol := newCol + 1 | n(nA) |
| 12 | newRow := newRow + 1 | nA |
| 13 | return aux | 1R |
| total | | $5n^2 + 5n + 6$ |
| Estimativa O | | $O(n^2)$ |

| Linha | procedure: removeUnwantedVerticesIndexes(apIndexes[1 to n]:integers; verticesNames[1 to n]:String | |
|---|---|---|
| 1 | aux[verticesNames.length - apIndexes.size()]:String | |
| 2 | deleted:integer := 0 | 1A |
| 3 | for i := 0 to verticesNames.length | (n+1)A + (n+1)C |
| 4 | if NOT(apIndexes.contains(i)) | nC |
| 5 | aux[i - deleted] := verticesNames[i] | nA |
| 6 | else | |
| 7 | deleted := deleted +1 | |
| 8 | return aux | |
| total | | $4n + 3$ |
| Estimativa O | | $O(n)$ |

| Linha | procedure: getSmallestIndex(smallestCosts[1 to n]:double) | |
|---|---|---|
| 1 | smallest:double := smallestCosts[0] | 1A |
| 2 | index := 0 | 1A |
| 3 | for i := 1 to smallestCosts.size() | (n+1)A + (n+1)C |
| 4 | if smallestCosts[i] < smallest | nC |
| 5 | smallest := smallestCosts[i] | nA |
| 6 | index := i | nA |
| 7 | return index | 1R |
| total | | $5n + 5$ |
| Estimativa O | | $O(n)$ |

| Linha | procedure: proceedToOneVertex(verticesNames[1 to n]:String; matrix[1 to n][1 to n]:double) | |
|---|---|---|
| 1 | read = Scanner(System.*in*) | |
| 2 | vertices[]:String := List.*of*(verticesNames) * | 1A + O(n) |
| 3 | print("Choose the initial sign?") | 1I |
| 4 | vertex:String = read.nextLine() | 1A |
| 5 | if vertices.contains(vertex) * | 1C + O(n) |
| 6 | *us18Routine* (verticesNames, matrix, vertex); | $O(n^3)$ |
| 7 | else | |
| 8 | print("Vertex not found!") | |
| total | | $n^3 + 2n + 4$ |
| Estimativa O | | $O(n^3)$ |

* foi usada uma função de bilbioteca Java, no entanto, sabe-se pela documentação que esta tem complexidade O(n)

| Linha | procedure: proceedToAllVertices(verticesNames[1 to n]:String, matrix[1 to n][1 to n]) | |
|---|---|---|
| 1 | read := Scanner(System.*in* ) | 1A |
| 2 | opt:String := "" | 1A |
| 3 | print("Proceed to all vertices? (y/n)") | 1I |
| 4 | do | 1C |
| 5 | opt := read.nextLine() | 1A |
| 6 | if opt <> "y" and opt <> "n" | |
| 7 | print("Insert the correct option!") | |
| 8 | while opt <> "y" and opt <> "n" | |
| 9 | if opt = "y" | 1C |
| 10 | apIndexes[]:integer := *identifyAP_indexes* (verticesNames) | 1A + O(n) |
| 11 | verticesNamesCopy[]:String | |
| 12 | for i = 0 to verticesNames.length | (n+1)A + (n+1)C |
| 13 | if NOT(apIndexes.contains(i)) * | nC |
| 14 | verticesNamesCopy := verticesNamesCopy + verticesNames[i] | nA |
| 15 | print("Smallest routes") | 1I |
| 16 | for each v in verticesNamesCopy | (n+1)A + (n+1)C |
| 17 | *us18Routine* (verticesNames, matrix, v) | nO($n^3$) |
| total | | $n^4 + 7n + 12$ |
| Estimativa O | | O($n^4$) |

* foi usada uma função de bilbioteca Java, no entanto, sabe-se pela documentação que esta tem complexidade O(n)

| Linha | procedure: identifyAP_indexes(verticesNames[1 to n]:String) | |
|---|---|---|
| 1 | aux[]:integer | |
| 2 | for i := 0 to verticesNames.length | (n+1)A + (n+1)C |
| 3 | if verticesNames[i].startsWith("AP")) | nC |
| 4 | aux := aux + i | nA |
| 5 | return aux | 1R |
| total | | $4n + 3$ |
| Estimativa O | | O($n$) |

# Conclusão

A análise temporal do algoritmo em questão revelou uma complexidade assintótica de para o “pior caso” de análise de $O(n^4)$. Isso significa que o tempo de execução do algoritmo tem um crescimento proporcional à quarta potência do tamanho da entrada. Em termos práticos, quando aplicado a dados de entrada de grandes dimensões, o crescimento do tempo de execução do algoritmo analisado é especialmente elevado quando comparado com outras complexidades comuns.

Seria recomendável explorar as otimizações possíveis para o presente algoritmo, entre as quais a utilização de funções de bibliotecas Java, já implementadas e com complexidade temporal devidamente documentada.

Por outro lado, face ao objetivo de aplicabilidade do presente algoritmo, isto é, no âmbito proposto inicialmente, será muito pouco expectável que o volume de dados de entrada possa vir a crescer para ordens de grandeza que possam torná-lo inviável.

# Bibliogafia

Moura, A. (2023). *Matemática Discreta*. http://www.educ.fc.ul.pt/docentes/opombo/seminario/