# US303 – Análise de Complexidade

public ArrayList<Firm> setHUBs(int n) throws InvalidParameterException {  
 if (n < **0**) throw new InvalidParameterException("Number of hubs should be higher than 0")**;** ArrayList<User> vertices = graph.vertices()**;** ArrayList<Firm> firms = new ArrayList<>()**;** for (User v : vertices)  
 if (v instanceof Firm)  
 firms.add((Firm) v)**;**

if (firms.size() < n)   
 throw new InvalidParameterException("Number of HUBs should be lower or equal to the number of Firms")**;**  
  
 ArrayList<LinkedList<User>> paths = new ArrayList<>()**;** ArrayList<Integer> dists = new ArrayList<>()**;** ArrayList<Firm> hubs = new ArrayList<>()**;** ArrayList<Double> averages = new ArrayList<>()**;** int sum**,** index**;** double average**;** for (Firm v : firms) {  
Algorithms.*shortestPaths*(graph**,** v**,** Integer::*compare***,** Integer::*sum***, 0,** paths**,** dists)**;** sum = **0;** for (Integer i : dists) sum += i**;** average = **1.0d** \* sum / dists.size() - **1;** if (hubs.size() < n) {  
 hubs.add(v)**;** averages.add(average)**;** } else {  
 index = (averages.indexOf(Collections.*max*(averages)))**;** if (averages.get(index) > average) {  
 hubs.set(index**,** v)**;** averages.set(index**,** average)**;** }  
 }  
 }  
 for (Firm hub : hubs) hub.setHUB(true)**;** return hubs**;**}

|  |  |  |
| --- | --- | --- |
| Linha | Código | Complexidade |
| 1-3 | if (n < **0**) throw new InvalidParameterException("Number of hubs should be higher than 0")**;** ArrayList<User> vertices = graph.vertices()**;** ArrayList<Firm> firms = new ArrayList<>()**;** | 1 |
| 4-6 | for (User v : vertices)  if (v instanceof Firm)  firms.add((Firm) v)**;** | V |
| 7-13 | if (firms.size() < n) {  throw new InvalidParameterException("Number of HUBs should be lower or equal to the number of Firms")**;** } ArrayList<LinkedList<User>> paths = new ArrayList<>()**;** ArrayList<Integer> dists = new ArrayList<>()**;** ArrayList<Firm> hubs = new ArrayList<>()**;** ArrayList<Double> averages = new ArrayList<>()**;** int sum**,** index**;** double average**;** | 1 |
| 14 | for (Firm v : firms) { | F |
| 15 | Algorithms.*shortestPaths*(graph**,** v**,** Integer::*compare***,** Integer::*sum***, 0,** paths**,** dists)**;** | F\*(V log V + E) |
| 16 | sum = **0;** | F |
| 17 | for (Integer i : dists) sum += i**;** | F\*V |
| 18-27 | if (hubs.size() < n) {  hubs.add(v)**;** averages.add(average)**;** } else {  index = (averages.indexOf(Collections.*max*(averages)))**;** if (averages.get(index) > average) {  hubs.set(index**,** v)**;** averages.set(index**,** average)**;** } } | F |
| 28 | } |  |
| 29 | for (Firm hub : hubs) hub.setHUB(true)**;** | n |
| 30 | return hubs**;** | 1 |

Este algoritmo execute a *shortestPaths* F vezes, onde F é o número de empresas, para calcular todos as distâncias mínimas de F para todos os clientes/produtores, com o objetivo de obter os HUBs que lhes são mais próximos em média. O algoritmo *shortestPaths* tem complexidade O(VlogV+E), onde V e o número de vértices do grafo e E o número de arestas.

Logo tem complexidade **O(F\*(V log V + E))**