**1Resolução do Exame Normal de ESINF (30 de Janeiro 2017)**

Nelson Tavares Rafael (1101369)

Tudo corrigido pela regente, exceto o 3 (Márcia Pinto).

1.

public static Map <Integer, LinkedList <Integer>> KsubLists (LinkedList <Integer> list, ArrayList <Integer> centers) {

Map <Integer, LinkedList<Integer>> maplists=new LinkedHashMap<>();

Integer center=0, nearCenter=0;

ListIterator <Integer> it = list.ListIterator(); // coloca 2 no primeiro elemento

While (it.hasNext()) {

Integer elem = it.next();

// agora calcula-se a distancia do ponto em L até ao centro em C

Integer absDist=Integer.MAX\_VALUE;

for (int i=0; i <centers.size();i++){

Integer center=centers.get(i);

If (Math.abs(elem-center)<absDist) {

absDist=Math.abs(elem-center);

nearCenter=center;c

}

}

LinkedList<Integer> subList= maplists.get(nearCenter);

if (subList==null) {

maplists.put(nearCenter, new LinkedList <Integer>());

subList= maplists.get(nearCenter);

}

subList.add(elem);

}

Return maplists;

}

2.

O método faz um clone da heap.

É determinístico. Porque vamos sempre iterar todos os elementos, vai ter sempre a mesma complexidade.

Complexidade: n log n

(Nota: se fosse não determinístico tinha que se dizer no exame o melhor e pior caso)

3.

A regente propôs adaptar o find. Comparar a string. Se for 1 vai à direita. Se for 0 vai à esquerda.

Public static integer elem (BST tree, String s) {

if (tree==null && s==null)

return null;

int i;

Node n = tree.root();

for (i=0; i<s.length();i++){

char c=s.charAt();

if (c==0)

n=n.getLeft();

if (c==1)

n=n.getRight();

if (n==null)

return null;

}

return (integer)n.getElement();

}

4.

private String nextPromotion(Graph<String, Integer> g) {

for (String vert : g.vertices()) // para percorrer os vértices do grafo

if (g.inDegree(vert)==0)

return vert;

Return null;

} // Isto devolverá o João que está no grau 0

//vamos precisar de retirar nós do grafo, para isso faz-se um clone

public List<String> getPromotions(Graph <String, Integer> g, Integer n) {

List <String> l = new ArrayList<>();

Graph <String, Integer> gc = g.clone();

for (int i=0;i<n;i++){

String nextProm=nextPromotion(gc);

if (nextProm==null)

return null;

gc.removeVertex(nextProm);

l.add(nextProm);

}

return l;

}

5.

- Visitar em largura (visita get Left e get Right)

- Iterativo

- Não existe travessias

- Uma heap é um vetor

public int getSubHeap (int idx, v[] vet) {

if (idx>heap.size())

Return -1;

int i=0;

LinkedList <Integer> queue = new LinkedList<>();

queue.add(idx);

vet[i]=heap.get(idx).getValue();

while (!queue.isEmpty()) {

idx=queue.remove();

if (hasLeft(idx)) {

i=i+1;

vet[i]=heap.get(left(idx)).geValue();

queue.add(left(idx));

}

if (hasRight(idx)) {

i=i+1;

vet[i]=heap.get(right(idx)).geValue();

queue.add(right(idx));

}

}

return i+1;

}