Ordkedja

Carl Johan Helgstrand - ak0064

Gustav Florén - aj8349

Robert Lupeskovic - ak3463

Kod

Graf-klassen som följer med boken kunde inte hantera strängar så istället för att använda den skrev vi vår egna graf och nod klass som var inspirerade av graf-klassen som finns på kursens hemsida[1], där vår klass kan hantera strängar och varje nod är en sträng.

Graph

Vår Graph-klass fungerar ungefär som bokens Graph-klass fast med några justeringar. Bokens Graph-klass har int-värden som values och en "Bag" med alla bågar associerat till varje nod. I vår Graph-klass är istället varje nod en klass som heter Node som har en sträng-variabel som heter node, som är ett ord på fem bokstäver, en arraylist av bågar som innehåller nodens bågar, en boolean som heter visited som används vid breadth-first search för att bestämma om en nod har varit besökt eller ej, och en int som heter distance som också används vid breadth-first search för att bestämma avståndet mellan en nod och en annan.

Graph-klassen har även två nya metoder som används vid instansiering av klassen, metoderna calcEdges och "controlWord. Den förstnämnda metoden används för att beräkna bågarna för varje klass och den gör detta genom att gå igenom hela listan med noder två gånger i en nested for loop. controlWord metoden tittar ifall de fyra sista bokstäverna från en nod finns i en annan nod. Ifall detta är sant, läggs denna nod till som en båge. Detta genomförs för alla noder.

Bredd-först-sökning

Vår implementation av breadth-first search för uppgiften:
Första loopen återställer värdet för varje nod, kanske inte den smartaste implementationen eller lösningen men det fungerade bra och gav korrekt utdata.
Vi använder oss av en länkad lista för queue med .remove() och .add(element).

[1] https://algs4.cs.princeton.edu/code/edu/princeton/cs/algs4/Graph.java.html

While loopen kör så länge listan har element. Första iterationen är nod N start noden, om N har värdet av mål noden returneras noll, detta är dock ett “edge case”. Om inget returneras loopar den över alla edges för N, och lägger till alla noder i queuen(länkad lista), vidare sätts dessa noder som visited (true) och distance för noderna blir distance+1. Dessa noder kommer därefter genomgå samma process till N = värdet av goal noden (strängen) eller inga noder finns i queuen, då returneras -1.

```
public int BFS(Node start, Node goal){

    for (int i = 0; i < vertices.size(); i++) {
        Node n = vertices.get(i);
        n.setDistance(0);
        n.setVisited(false);
    }

    Queue<Node> queue = new LinkedList<>();
    queue.add(start);
    while (!queue.isEmpty()){
        Node n = queue.remove();
        if(n.getName().equals(goal.getName())){
            System.out.println(start.getName() + " " + goal.getName());
            start.setVisited(false);
            start.setDistance(0);
            return n.getDistance();
        }
        for (int i = 0; i < n.getEdges().size(); i++) {
            Node e = n.getEdges().get(i);
            if(!e.isVisited()){
                e.setDistance(n.getDistance()+1);
                e.setVisited(true);
                queue.add(e);
            }
        }
    }
    start.setVisited(false);
    start.setDistance(0);
    return -1;
}
```

Tidskomplexitet

Kvadratisk tidskomplexitet, calcEdges() nested for loop där båda for-looparna loopar över storleken av N(arraylist<Node> nodes.size()). Bredd

Skapandet av Graph-klassen har en tidskomplexitet på O(V^2)

BFS har en tidskomplexitet på O(V+E)

Testfall har en tidskomplexitet på (O(V+E) * testfall) + O(V^2)

```
public void calcEdges(ArrayList<Node> nodes){
    for (int i = 0; i < nodes.size(); i++) {
        Node n = nodes.get(i);
        for (int j = 0; j < nodes.size(); j++) {
            if(!nodes.get(i).getName().equals(nodes.get(j).getName())){
                Node otherN = nodes.get(j);
                if(controlWord(n, otherN)){
                    n.addEdge(otherN);
                }
            }

        }
    }
}
```

```java
public boolean controlWord(Node current, Node comparedTo){
    String word = current.getName();
    String compareWord = comparedTo.getName();
    int counter = 0;
    char[] compared = compareWord.toCharArray();

    for (int i = 1; i < word.length(); i++) {
        char ch = word.charAt(i);
        for (int j = 0; j < compareWord.length(); j++) {
            if(ch == compareWord.charAt(j) && compared[j] != 0){
                compared[j] = 0;
                counter++;
                break;
            }
        }
    }
    if(counter == 4){
        return true;
    }
    else return false;
}
```