Algorithmics 3 Assessed Exercise

Status and Implementation Reports

Adam Kurkiewicz
1102499k

November 19, 2013

Status report

The implementation of both Dijkstra's and Backtrack search algorithms are correct. This has been tested against the results obtained by other students for certain inputs, which have been posted and discussed on official course's facebook group.

Implementation report

To solve both problems I have used a classical adjacency list implementation. Every vertex is represented by an instance of Node.java class. Every object of this class carries information about its immediate children and its immediate parents. The instance of Graph.java carries information about every vertex belonging to the graph. That information can be accessed using the name of the vertex. The data structure used to implement this solution is a hash map. In order to instantiate the graph with properly connected nodes I use an interesting approach, which has got a worst-time complexity $O(|V| + |E|^2)$, where V is the set of vertices and E is the set of edges. The approach is again using a hashmap. For every word of length n and for every k ¡= n I create n hashmap keys with kTH character in the word replaced by a quotation mark character "?". Every key created in such manner maps onto a list which gets populated with all words differing by exactly one character. Because the number of edges depends on the maximum number of characters, which is constant, we can say that the complexity of graph creation is $O(|V|)$.

(a) the Dijkstra's algorithm implementation is absolutely standard with one modification. Instead of a fibonacci heap I used Java's standard library PriorityQueue.

(b) Backtrack search implementation is absolutely standard. I use a breadth first search, which starts from the vertex representing word1 and which terminates upon reaching the vertex representing word2 or once the list of descendants has been fully exhausted.

Empirical results

The program terminates in about 500 ms for most types of the input. Two example interactions with the program follow

(a) input: javac Dijkstra.java
```
input: java Dijkstra words5.txt mouse click
click
crick
trick
trice
trite
write
wrote
wroth
troth
tooth
sooth
south
sough
rough
rouge
rouse
mouse

chain length in elements: 16
chain total weight sum: 105

Elapsed time: 451 milliseconds
```

(b) input: javac WordLadder.java

```
input: java WordLadder words5.txt mouse click
block
clock
chock
check
cheek
cheep
sheep
sheet

chain length: 7

Elapsed time: 393 milliseconds
```