European Funds Knowledge Education Development · Wrocław University of Science and Technology · European Union European Social Fund

ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej

PWr Week 12
Data Structures and Algorithms,
Laboratory – **List 11**

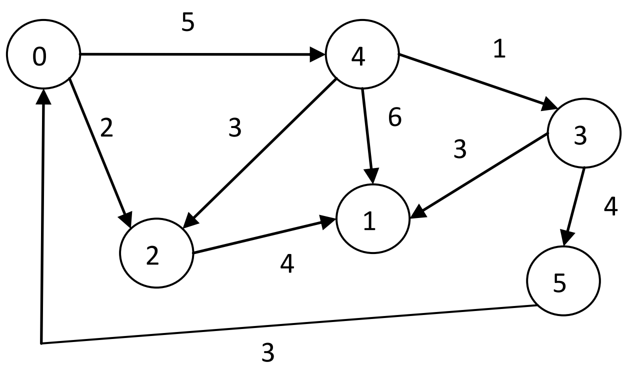
Introduction

One of problem from graph theory is to find a path from one node to another. This problem can be expressed wider: to find a path from a source vertex to all vertices. One the best known algorithm was constructed by Dijkstra. The algorithm uses greedy techniques and store intermediate information and final results in two arrays: predecessor, which remember from which node it was come to a node, and with minimum distances found so far for each node. During execution there are also two structures used: array of colors of the node (WHITE - which marks if the node was not analyzed till now, GREY – a node is analyzed, but the solution was not found till now, BLACK – the solution for a node was found) and a min-heap of nodes with minimum distances as a key to order. The array of distances and the min-heap structures can be in clever way stored in one structure.

In the first step the algorithm analyzes all vertices accessible directly from a source vertex. The information is transferred to minimum distances array.There rest of nodes have assigned value infinity. All of them are inserted into min-heap. Source vertex is marked BLACK, all adjacent vertices – GREY, and the rest – WHITE. The rest steps are all the same:

- select the vertex with minimum distance from a min-heap, removing it from this structure. Let’s call this node x.
- For all not BLACK vertices check if the distance from source vertex throw the x vertex is smaller than stored in the array of distances – if yes update the state of the min-heap, the array of distances and the array of predecessor, because it was found a new smaller path. Color all updated vertices as GREY.
- Color vertex x as BLACK.

Let’s assume a below graph:



If we will start from a vertex 0, the initial value of the minimum distance for the vertex 4 will be 5 and for the vertex 2 will be 2. The rest of vertices will have assigned value infinity. In the predecessor array for vertices 4 and 2 will be stored value 0 and for the rest will be **`null`**. The color of the vertex 0 has to be BLACK, of the vertices 2 and 4 – GREY, of the rest ones – WHITE. All the distances will be inserted into min-heap and in the next step vertex 2 with value 2 will be selected. In this case we can find a shorter path for a vertex 1 if we will go throw the vertex 2. Before the distances for vertex 1 was infinity, now the path 0->2->1 has a distance 6 which is better than infinity. This changes have to be done on the min-heap (and the array of distances) and the array of predecessor. As a last action the color of the node 2 has to changed to BLACK.

List of tasks.

1. Use an implementation of class `Document` and `Graph` from previous list. Maybe it is needed to modify the way to store information about the graph in weighted form in two dimensional array.
2. In the `Main` class use any representation of `Document` objects collection.
3. Add to class `Graph` method `DijkstraSSSP(String startVertexStr)`, which implements a Dijkstra algorithm for single-source shortest paths problem. In case you have a choice during executing algorithm, analyze vertices in a lexicographical order.

For 100 points present solutions for this list till Week 13.
For 80 points present solutions for this list till Week 14.
For 50 points present solutions for this list till Week 15.
After Week 15 the list is closed.

There is an appendix on next pages.

Appendix
The solution will be automated tested with tests from console of presented below format. The test assumes, that there are documents in a collection.

If a line is empty or starts from `'#'` sign, the line have to be ignored.
In any other case, your program should print an exclamation mark and write (copy) introduced a line and then, depending on the command follow the correct procedure / function.

If the current document is equal **`null`**, the program has to write "no current document".

If a line has a format:
`getdoc <str>`
your program has to choose a document with a name `str`, and all next functions will operate on this document.

If a line has a format:
`ld <name>`
your program has to call a constructor of a document with parameters `name` and `scan` for current position. The `name` have to be a correct identificator (in the constructor the `name` have to be stored in lower case letters). If it is not true, write in one line "`incorrect ID`". If the identificator is correct, the loaded document have to be now the current document. Try to add it to the hashtable. If it is impossible write "error" on the output.

If a line has a format:
`add <str>`
your program has to call `add(new Link(str))` for the current document. If the string is correct, on the output write "`true`". The `str` can be in the same format like a link in this task: only identificator or identificator with a weight in parenthesis. If there is such a name of link in a document, overwrite it.

If a line has a format:
`get str`
your program has to get the link with the reference equals `str` for current document's links and write returned `Link` object. If the `str` is incorrect, write in one line "error".

If a line has a format:
`clear`
your program has to call `clear()` for current document, to remove all links.

If a line has a format:
`show`
your program has to write on the screen the result of caling `toString()` for the current document. The first line has to present text "Document: <name>", the rest of lines will depend on choosen collection of links. **This command will be not executed in automatic tests.**

If a line has a format:

`rem <idStr>`
your program has to remove a link with reference `idStr` for current document and write on the screen returned value in one line.

If a line has a format:
`size`
your program has to write on screen the result of `size()` for current document and write on the screen returned value in one line.

If a line has a format:
`ha`
your program has to end the execution, writing as the last line “END OF EXECUTION”. Every test ends with this line.

If a line has a format:
`bfs <str>`
your program has to execute the `bfs(str)` method, which will return the string of visited documents. Names of documents have to be separated by comma and one space. The returned value present on the output stream.

If a line has a format:
`dfs <str>`
your program has to execute the `dfs(str)` method, which will return the string of visited documents. Names of documents have to be separated by comma and one space. The returned value present on the output stream.

If a line has a format:
`cc`
your program has to execute the `connectedComponents(str)` method, which will return integer value. The returned value present on the output stream.

If a line has a format:
`sssp <str>`
your program has to execute the `Diksjtra(str)` method, which will a string with presentation of the paths. For every vertex lexicographically sorted there will be one lineof the format:

- For starting vertex <startVertex>
  `<startVertex>=0`
- For vertex <vertexName> to which there is no path:
  `no path to <vertexName>`
- For every other <vertexName> vertex presentation of the path in the below format ended with equal sign and value of the shortest path:
  `<startVertex>-><vertex1>->…-><vertexName>=<weightOfPath>`

A simple test for this task:
INPUT:
```
#Test for Lab11
ld x
```

```
link=y(9)
eod
ld y
eod
ld a
link=b(2)
link=c(5)
eod
ld c
link=d(5)
link=f(6)
eod
ld b
link=d(3)
link=e(4)
eod
ld e
link=d(3)
link=f(4)
link=h(2)
link=g(8)
eod
ld d
link=e(3)
link=f(1)
eod
ld h
link=e(2)
link=g(1)
eod
ld g
link=h(1)
link=f(7)
eod
ld f
link=d(1)
link=e(4)
link=g(7)
eod
sssp a
sssp d
sssp w
ha
```

OUTPUT:

```
START
!ld x
!ld y
!ld a
!ld c
!ld b
!ld e
!ld d
!ld h
!ld g
!ld f
!sssp a
a=0
a->b=2
a->c=5
a->b->d=5
```

```
a->b->e=6
a->b->d->f=6
a->b->e->h->g=9
a->b->e->h=8
no path to x
no path to y
!sssp d
no path to a
no path to b
no path to c
d=0
d->e=3
d->f=1
d->e->h->g=6
d->e->h=5
no path to x
no path to y
!sssp w
error
!ha
END OF EXECUTION
```