The Technion – Faculty of Biology
Computational Approaches for Life Scientists - 134141
Spring 2014

Home assignment 2 – Due April 29th, 23:55

Submission instructions:

- The assignment should be submitted via moodle.
- Starting this exercise, it is allowed to submit in pairs:
  - In such a case, only one of the students will upload the required files to moodle. The files' names will be id1_id2, when id1 and id2 are the ID numbers of the 2 students.
  - Both students MUST be fully involved in preparing the exercise, writing the executing code, etc.
- All questions will be submitted in 2 files:
  1. The "dry" part will include answers to all the questions and the required explanations. This part will be submitted in a single doc/docx file.
  2. The "wet" part, with all the code you have written to answer the questions will be submitted in a py file. The code in this file must support your answers and conclusions, such that when run yields the results provided in the dry part. The grade for this part will be given for a correct and reasonable code, which avoids unnecessary complications and bad styling.
- The 2 files should be uploaded to moodle by the deadline. You may submit late though, losing 5 points for each day. For that matter, submission after 23:55 is a day late.
- Note: while each question normally has a single correct answer, the code you'll write to answer then may be written in various ways. There is no single correct code.
  However, to get full score you must make an effort to write your code in a reasonable manner, in terms such as complexity, meaningful names, etc.

  You are requested to comment (#) your code, and you may even leave old code that you have written and no longer use (commented out!). This way I will be able to track your thinking process, at least partially.
- Bonuses: a maximal amount of 10 points may be given as a bonus to your grade, for interesting comments, mostly biological ones, that you find relevant. For example:
  - A relevant application for the question's topic
  - New information the sheds interesting light on the question, or puts it in an interesting biological context
- You may bring information from previous courses, a seminar you are taking, your own research, or even the internet. You must state your references.

Question 1 – Sequencing by hybridization

The following list of 4-mers originates in a single DNA sequence. Use the code for *sequencing by hybridization* taught in class to find the original piece of DNA. You will first need to build the corresponding edge-overlap graph (you can do this manually if you prefer), and then run the functions `eulerPath` and `path2string`.

['TATA', 'ATAA', 'TAAT', 'AATT', 'ATTA', 'TTAT', 'TATT']

Can you identify what this sequence is?

Question 2 – Eulerian cycles

In class we saw the function `eulerPath`, which returns an Eulerian path in a graph. It was mentioned that this implementation will not work properly if the Eulerian path is actually a *cycle*. Find out where the problem is, and fix it (hint: an addition of a simple line to one of the functions will do the job).
In your answers, mention what you added to which function, and give an output example, before and after the fix, to show it is correct.

Question 3 – Dijkstra's algorithm with negative edges

Recall that Dijkstra's algorithm does not guarantee correctness when the graph has negative edges. A professor of Computer Science suggests the following solution to this limitation:

> *Find the minimal weight in the graph (which must be negative), and add the absolute value of that minimum to all weights in the graph. Now all the edges have weight >=0, so we can run Dijkstra. Then, reduce from all the distances found the minimum we added before.*

Is this suggestion good? Explain why it is, or give a simple counter example in which the algorithm returns incorrect distances.

Question 4 – Graph theory notions

Following are definitions for several common notions in graph theory.
Given a directed weighted graph G:

- The **distance** from node u to node v (denoted $d(u,v)$ or $dist(u,v)$) is the weight of the shortest path from u to v, or ∞ if no such path exists. This definition was taught in class.
- The **eccentricity** of some node v is the greatest distance between v and any other node in the graph. It can be thought of as how far a node is from the node most distant from it in the graph.
- The **radius** of a graph is the minimum eccentricity of any node.
- The **center** of a graph is a node whose eccentricity equals the radius of the graph.

The file hw2.py in the site contains an implementation of a function eccentricity(G, v), which returns the eccentricity of node *v* in a graph *G*. This function uses Dijkstra's algorithm that we saw in class.

Some of the following sections relate to the graph on slide 25 in class 5 presentation. The representation of this graph (as an adjacency matrix) appears in hw2.py, for your convenience. Denote this graph *H*.

a. What is the eccentricity of node v_0 in the graph *H*? And of node v_4? Use the above mentioned function to answer, and verify your answer by inspection.
b. Implement the function radius(G), which returns the radius of a graph *G*. What is the radius of *H*?
c. Implement the function center(G), which returns a center of a graph *G*. Find a center of *H*. Can a graph have more than one center? Can all the nodes in a graph be its centers?

Given an undirected non-weighted graph *G*, and some source node *v* in it:

- The **level structure** of *G* is a partition of the nodes into subsets that have the same distance from *v*. For example, level 0 includes only *v*, level 1 includes all nodes which are reachable from *v* by 1 edge, etc.

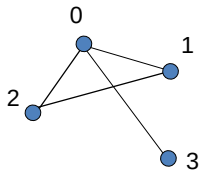
d. How many levels does *H* have, with respect to v_0 as source? And v_4?
e. True or False: In a level structure of a graph *G*, each edge connects 2 nodes that are located at consecutive levels (e.g. level 4 and level 5).
   Explain why this claim is True, or give a counter example to falsify it.

Question 5 – shortest paths recreation

Following is a partial implementation of the function spath, which recreates the shortest paths identified by Dijkstra's algorithms (specifically, it uses the list of "previous nodes" returned by Dijkstra's algorithm). The function spath receives a graph *G*, represented as a matrix, a source node and a target node, and returns a list representing a shortest path from source to target.

For example:

```
G=[ [0,1,1,1],
    [1,0,1,0],
    [1,1,0,0],
    [1,0,0,0]  ]
>>> spath(G, 3, 1)
[3,0,1]
```



a. Complete this function in the missing places, in the file hw2.py.

b. Run it and find the shortest path in the graph H (mentioned in Question 1), from v_0 to v_1.

```
def spath(G, source, target):
    ''' return the shortest path in G from source to target.
        The output is a list of nodes '''

    dists, prevs = dijkstra(G, source) #we don't care about dists here

    v = target
    path = [v]
    while v != _____:
        v = _____ #previous node in the path
        path += _____

    path.reverse()
    return path
```

Question 6 - Reductions

Consider the following version of the shortest paths problem:

> *Find shortest paths from a source node in a graph G, which has weights on both its edges* ***and nodes****. The weight of a path in such a graph is defined as the sum of weights along the path on both edges and nodes.*

Describe how Dijkstra's algorithm can be used as a "black-box" to solve this version. For that, describe by a schematic figure (such as those shown in class) a *reduction* from the new problem to the original shortest paths problem. Briefly explain your figure and why the reduction is correct (recall this argument must be bi-directional).

The End!