Scientific Computation
Autumn 2024
Problem sheet 2 solution

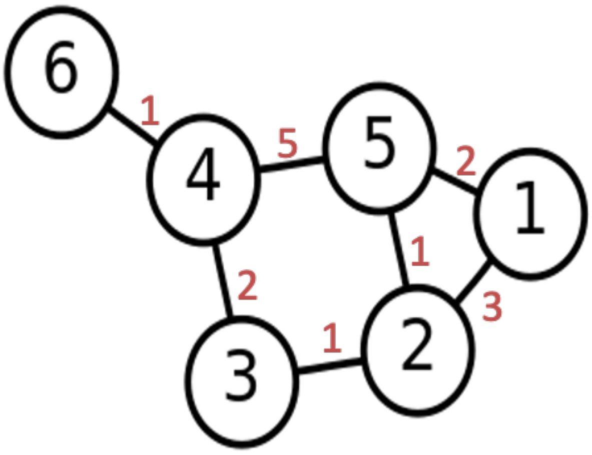

Figure 1: Graph for question 1

1. Apply Dijkstra's algorithm to find the distance between nodes 4 and 5 in the graph above. Set node 4 as the source and terminate the search once the distance to node 5 has been found. What is the sequence in which nodes are finalized?
   Solution: The distance is 4. The nodes are finalized in the following order: $(4, 6, 3, 2, 5)$.

2. Consider the code shown below:

```python
import networkx as nx
def graph_code(G,x,L = []):
    """
    G: networkx graph with nodes numbered from 0 to N-1
    x: node number
    """
    if L==[]:
        N = G.number_of_nodes()
        L = [-1]*N
        L[x]=1
    for i in G.adj[x]:
        if L[i]==-1:
            L[i]=1
            L=graph_code(G,i,L)
    return L
```

What is the problem that this code is attempting to solve? Discuss the correctness of the code and its big-O time complexity.

Solution: This is a variant of depth-first search (DFS) used to find all reachable nodes from the source node, x. The difference from conventional DFS is that when an explored node's neighbors are examined, if there are multiple unexplored neighbors, then the earlier an unexplored neighbor is examined, the earlier the neighbor's neighbors will be examined. So this doesn't quite follow the last in, first out principle required by a stack. The code is consistent with the general search algorithm described in lecture 6, so the correctness of this code follows from the correctness of that algorithm. The cost is $\mathcal{O}(L+N)$. Each reachable node will be relabeled once, each link connecting a pair of reachable nodes will be examined twice. Aside from the graph, there is one container, `L`. There is an $\mathcal{O}(N)$ cost associated with the initialization of `L`, but otherwise, there are only direct accesses to specific elements of the list which each have $\mathcal{O}(1)$ cost.

3. You would like to find all connected components in an undirected graph with N nodes, L links, and N_c components. The graph is provided as an adjacency matrix. What is the cost of applying BFS to this problem?

   Solution: Each time we examine the neighbors of a node, we require N steps rather than k where k is the degree of the node (we have to iterate through an entire row of the adjacency matrix). Since we will have to examine the neighbors of all N nodes, the cost will be $\mathcal{O}(N^2)$.

4. Explain how to sort the following list in non-decreasing order using a binary heap: `L = [3,5,8,9,4,2,1]`. Can your approach be used generally? What is the big-O time complexity of your approach applied to a list with n elements? When analyzing the cost, you may assume that the list has $n = 2^m - 1$ elements where m is a positive integer.

   Solution: First heapify the list ($\mathcal{O}(n)$ cost) to form a tree where node 1 is at the top. The rest of the tree is not uniquely determined, and one possibility is that node 1 had nodes 4 and 2 as children. The children of 4 could be 9 and 5, while the children of 2 could be 3 and 8. After creating the heap, we would just need to pop the minimum element one iteration at a time ($\mathcal{O}(m)$ cost) and append it to a new list ($\mathcal{O}(1)$ cost). There will be n iterations, so the overall cost will be $\mathcal{O}(mn)$.

5. Consider the following system of 2 ODEs:

$$\frac{d\theta}{dt} = \phi \tag{1}$$

$$\frac{d\phi}{dt} = -\sin\theta. \tag{2}$$

Find an approximate solution for the initial evolution of the system given the initial condition, $\theta(0) = \pi + 0.009$, $\phi(0) = -0.01$. Provide a qualitative description of the system's behavior. Explain why the approximation loses validity at sufficiently large times. (It's fine to use Python to help you with the calculations, but in principle, you should be able to work through this on paper.)

Solution: $(\pi, 0)$ is a fixed point, and the initial condition is close to this fixed point, so we can linearize around this point and consider the resulting dynamics. Let $\theta = \pi + \delta(t)$. Then, if $\delta = \mathcal{O}(\epsilon)$ with $\epsilon \ll 1$,

$$\sin\theta = \sin\pi + \delta\cos\pi + \mathcal{O}(\epsilon^2),$$

and the ODEs can be approximated as,

$$\frac{d\delta}{dt} = \phi \tag{3}$$

$$\frac{d\phi}{dt} = \delta. \tag{4}$$

Let $\mathbf{x} = \{\delta, \phi\}$. We can rewrite this system as,

$$\frac{d\mathbf{x}}{dt} = \mathbf{A}\mathbf{x},$$

where $\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$, and $\mathbf{x}(0) = \{0.009, -0.01\}$. The general solution is

$$\mathbf{x} = c_1\mathbf{v}_1\exp(\lambda_1 t) + c_2\mathbf{v}_2\exp(\lambda_2 t).$$

Here, λ_i is the ith eigenvalue of $\mathbf{A}$, $\mathbf{v}_i$ is the corresponding eigenvector, and c_1 and c_2 are determined using the initial conditions. The solution for the given initial conditions is $\lambda_1 = 1, \lambda_2 = -1$, $\mathbf{v}_1 = \{1, 1\}$, $\mathbf{v}_2 = \{-1, 1\}$, $c_1 = -0.000707$, $c_2 = -0.013435$. The fixed point is a saddle point, and the initial condition is nearly aligned with the second eigenvector which is why c_2 is much larger than c_1. Consequently, the second term in the solution dominates and the overall solution decays until the exponential growth of the first term allows it to "catch up" with the decaying second term. At long times, this exponential growth dominates, and at sufficiently large times, δ becomes so large that the approximation for $\sin\theta$ loses validity.