Vinay Shah
vss452

Programming Assignment 2 Report

From a space complexity and time complexity perspective, does it make more sense to use an adjacency matrix or an adjacency list in order to represent the intensity graph in part 1? Explain your answer.

Adjacency matrices are better suited to dense graphs, whereas adjacency lists are better suited to sparse graphs. While an adjacency matrix allows for O(1) time when checking if an edge exists between two vertices, it does occupy O(V^2) space, and adding a vertex to the graph takes O(V^2) time. An adjacency list, however, occupies O(|V| + |E|) space. In the worst-case scenario, E becomes V^2 and it can take up O(V^2) space. Adding vertices to graph takes less time as it is a list and not a matrix. The downside is that it can take O(V) time to check if an edge exists between 2 vertices.
Since our graph is sparse, with a maximum of 4 edges per vertex, having an adjacency matrix doesn't make sense as our adjacency list will actually end up taking O(V) space, and O(V) time to create, whereas it would take O(V^2) space for the adjacency matrix.
While I did not end up creating the graph in part 1, I created it in part 2. I was able to traverse the image matrix and do a sum of the difference in intensities without explicitly creating the graph.

Describe the algorithm you implemented in part 2 (both text and pseudocode are fine).

I implemented Prim's MST algorithm upon the intensity graph in adjacency list form and modified the algorithm a bit so that we generate a sum of the weights of the MST. The pseudocode is as follows:

```
constructPrunedGraph(int[][] image)
Graph G = createAdjacencyList(image)
create PriorityQueue pq
sum = 0
size = image.length*image[0].length //number of pixels
create Set mstSet
smallestKeyNode is the first node in G
set key of smallestKeyNode to 0
add smallestKeyNode to pq

while mstSet.size < size
    smallestKeyNode = pq.poll
        if smallestKeyNode not in mstSet
          then add smallestKeyNode to mstSet and add the key value to sum
             for each neighbor in smallestKeyNode's adjacency list
                do if neighbor not in mstSet
```

Vinay Shah
vss452

then if absolute difference in intensity of neighbor and node > neighbor.key
then remove neighbor from pq, neighbor.key = difference,
and add neighbor back to pq

return sum

In Big-O notation, state the runtime complexity of the algorithm you implemented in part 2, in terms of the number of pixels, p, in a given input image.

The maximum number of edges each pixel p can have is 4 in our graph, making our edge set E upper bounded by 4p.
Size defined in the pseudocode above is equivalent to the value of **p**

It takes me O(**p**+4**p**) = O(5**p**) = O(**p**) time to construct the adjacency list representation of the sparse graph. After entering the while loop, it takes constant time get the smallestKeyNode, and to check if it is not in the mstSet. Adding the smallestKeyNode to the mstSet, and adding it's key to the sum also take constant time. Iterating through each neighbor in the adj[p] takes O(4) = O(1) time. Checking if the neighbor is not in the mstSet is again constant time. Checking if the difference in intensities is less than the current neighbor.key is also constant time. The removal and addition of the neighbor from the pq is 2*O(log**p**) = O(log**p**). Since this can happen O(p) times due to the condition on the outermost loop being mstSet.size < p. Therefore using aggregate analysis, our total runtime of the algorithm is O(**p**log**p**).