Carsten Hood
HW10
CSCE 221-200

- -

1 Problem R-13.8 in the text (p. 655). Would you use the adjacency list structure or the adjacency matrix structure in each of the following cases? Justify your choice. (30 points)

(a) The graph has 10,000 vertices and 20,000 edges, and it is important to use as little space as possible.

adjacency list structure - this takes O(n+m) space, where n = 10,000 and m = 20,000. There are relatively few edges per vertex, so this is preferred over the O(n^2) space of the matrix structure. An adjacency matrix would occupy plenty of extra wasted space.

(b) The graph has 10,000 vertices and 100,000,000 edges, and it is important to use as little space as possible.

I would likely use an adjacency matrix, but no method is clearly superior. An adjacency list structure takes O(10^4 + 10^8), and a matrix structure takes O((10^4(^2)) = O(10^8), so both have a similar space cost of 10^8. However, I prefer the matrix structure because it wastes no memory in this case (because 10^8 is the maximum number of edges, assuming there are no duplicate edges).

(c) You need to answer the query isAdjacentTo as fast as possible, and the amount of space you use is not a concern.

adjacency matrix structure - the isAdjacentTo operation takes O(1), compared to the list structure's cost: min(deg(v1), deg(v2)). The grid-like matrix structure provides instant access to the location of an edge between two vertices.

- -

1 Carefully prove the following proposition (this is also Proposition 13.11 in the text, p. 598). (30 points) Let G be an undirected graph with n vertices and m edges. Then we have the following:

If G is a tree, then m = n-1.
Proof: Base case: consider the tree: A—B. It clearly has 1 edge and 2 vertices. 1 = 2-1
Inductive step: Assume m = n-1 true for all trees with less than N vertices. A tree T has N vertices. Remove one edge, resulting in two smaller trees. Use the inductive hypothesis.

If G is connected, then m >= n-1.
Proof: If G has no cycles then it is a tree. Above is shown that for a tree, m = n-1.
If G has cycles then we can remove an edge while retaining connectivity and continue to remove edges until there are no cycles left. The result is a tree with the same number

of vertices, but fewer edges. Therefore, a graph with cycles must have more edges than a tree that has the same number of nodes but no cycles. So m >= m_tree = n_both-1. Since the number of nodes is the same, we have: m >= n-1.

If G is a forest, then m <= n-1.
Proof: It is shown above that for a tree, m = n-1. A forest is a tree that does not necessitating connectivity. In other words, a forest can have fewer edges than a tree (nodes do not need to be connected) but no more (then it is no longer a forest). Therefore, m_forest <= m_tree = n_both-1. So m <= n - 1.

- -

3 An independent set of an undirected graph G=(V,E) is a subset I of V such that no two vertices in I are adjacent. That is, if u and v are in I, then (u,v) is not in E. A maximal independent set M is an independent set such that, if we were to add any additional vertex to M, then it would not be independent anymore. Give an efficient algorithm that computes a maximal independent set for a graph G. Provide an argument that your algorithm is correct and carefully analyze its running time. (40 points)

```
Algorithm maximal(G)
    max_set = empty set of vertices
    boolean adjacent = false
    for every vertex X in G
        for every vertex Y in max_set
            if (X.isAdjacent(Y))
                adjacent = true
                break
        if (adjacent == true)
            adjacent = false
        else
            max_set.push(X)
```

Create an empty set, max_set, to store the vertices of the output maximal independent set. Loop through every vertex in G and compare it to each vertex in max_set; if it is not adjacent to any vertex in S, add it to S. The first vertex will automatically be added since the set is initially empty. Then every other vertex will be added to the output set as long as it does not violate the independence of the output set. This will result in a maximal independent set.

The algorithm examines each of n vertices in G; in each iteration, another loop iterates through the output set. This loop performs an isAdjacent check; using a matrix structure this takes O(1). The worst case for the number of iterations through the output set is that the graph has no edges and the resulting maximal independent set contains every vertex. In this case, each successive value is added to the output set. Then, with each iteration of the first loop, the second loop iterates once more.
We have: n(0) + n(1) + n(2) + … + n(n-1) = n(n-1). Since all other operations within the loops are constant (using an adjacency matrix) the cost is O(n(n-1)), or O(n^2).