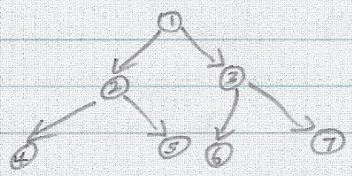
Kelsey Neis - neis@umn.edu
Assn 6
Section 001 22.1 - ~~1~~, 2, 3, ~~4~~ 5; 23.1 - ~~1~~, ~~3~~; 23.2 - ~~1~~, 2; 24.3 - 6

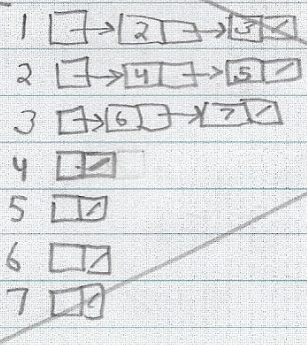
## 22.1

1) For out-degrees, $\Theta(V)$, because the out degrees are listed for each vertex. All that is needed is to look at each vertex and count the listed values.
For in-degrees, $\Theta(V \cdot E)$, because for each vertex, you have to check all edges to count the number of edges that point at it.

2) Binary tree (7):



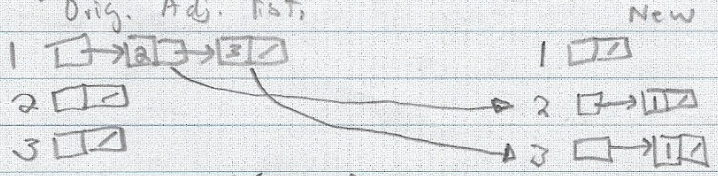

Adj. List:




Matrix:

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

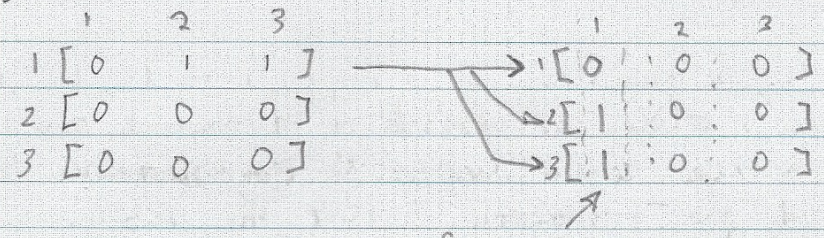
3) Adj. List: For each vertex, loop through the edges, adding the value of the current vertex to the entry of the edges on an empty copy of the adjacency list.




Runtime: $\Theta(V+E)$ — to create the new list → V, to copy each edge to the new list → E

Matrix: Create a blank copy of the matrix (assuming it's an array of arrays). Iterate through each array in the original matrix and add each element to the $i$th array in the new matrix.
Since we're iterating over all elements in the matrix, the runtime is $\Theta(V^2)$.

3 continued




4) Create a new adjacency list with the vertices of the multigraph. Then, for each vertex of the multigraph, force the list of edges into a set, which will remove duplicates. Then, iterate through the set, removing an element if it equals the current vertex (to remove self-loops). Since we removed duplicates, we can break once the self-loop is removed. Add each set created to the new adjacency list

Runtime:
- Force lists of edges to set: V
- Remove self loops: O(E)

$\Rightarrow O(V+E)$

5) Adj. list: Iterate through the vertices. For each vertex, loop through its edges. Retrieve the list of edges from each vertex corresponding to that edge and append to the corresponding vertex on the new adjacency list. Also append its original edges.

runtime: Assuming we can append the edges of the connected vertex to the new list of edges without having to iterate over them, this should take $C \cdot E$, the constant time being to add the current edge to the new array, then follow it to the connected vertex and add its list of edges to the new list. $\rightarrow \Theta(E)$

Matrix: Create a copy of the matrix with all zeros. Starting at the last row, iterate through each array. When a 1 is found, add the index to a new array with the key being the current row index, and the value being the list of indeces where there is a 1 ex. [{7: 8, 9}]. (This will prevent the need for

iterating over the same row twice). Then union the current index with the corresponding array in the new matrix. Then, union the list corresponding to the index with the new array. Using the example above, when processing row 6, if there is an edge to 7, then 7 would be added to the entry for 6 in the reference array and 7, 8, 9 would be added to the new matrix in the corresponding array.

runtime: $V^2$ for iterating over each element in the matrix
- Constant time for adding the index to the new matrix and to the reference array
→ $CV^2 = \Theta(V^2)$

*It could be more efficient by only processing the upper right triangle of the matrix, but the theta runtime would not change.

23.1

1) - Suppose $T$ = MST of $G$, $g \notin T$ ($g$ = min wt. edge)
- Add $g$ to $T$
- Since $T$ was an MST, adding $g$ will create a loop
- Remove another edge $e$ in the loop. Since $g$ is of minimum weight, $g \leq e$
- The resulting MST, $T'$ will have weight:
$w(T) + w(g) - w(e)$
→ $\therefore w(T') \leq w(T)$
- Since $T$ is MST, $T'$ must be MST as well

3) - Suppose $(U, V)$ is not a light edge crossing some cut $(S, V-S)$, but is part of the MST $T$
- Then, there exists an edge, say $(X, y)$, which
- is the light edge crossing the cut $(S, V-S)$. So $w(x,y) < w(u,v)$
- Let $T' = T \cup (x,y) \setminus (U,V)$
- $w(T') = w(T) + w(x,y) - w(U,V)$
- since $w(x,y) < w(U,V)$, the original assumption that $T$ is an MST is proved wrong. Therefore, $(U,V)$ must be a light edge.
note: we must remove $(U,V)$ when adding $(X,y)$, otherwise there will be a cycle

23.2

1) If there is a tie for minimum weight edge between all trees, then one of two scenarios is possible. The T and non-T edges connect to one other tree, or they connect to two different trees. If they connect to just one tree, then sorting the min weight edge(s) which are part of T to be before the other min weight edges will lead to them being included in the MST over the other edges. If they connect to different trees, then sorting the min weight edges in T before those not in T will also return T, since the order of min-weight edges in T does not matter, as long as they are all before the min-weight edges not in T.

2) Prim-matrix (G, r)

```
1    mst, mins = []
2    mins.add(MIN(G[r]))        ← V
3    while |mst| ≤ |G[r]|
4        x = MIN(mins) // incl. edge (u,v), w ← 1 ≤ T ≤ V
5        mst.add(x)
6        G[x.u][x.v] = ∞ and G[x.v][x.u] = ∞
7        mins.add(MIN(G[x.u]))  ← V
8        mins.add(MIN(G[x.v]))  ← V
9    return mst
```

runtime: The main heavy operations are on lines 2, 4, 7, and 8. Each of those implicitly loops through at most V elements, and the while loop runs V times, so in total, this implementation takes $CV^2 \Rightarrow \Theta(V^2)$.

* Note: In lines 7 and 8, adding to mins will replace the existing entry for a given key, if applicable

24.3

6) An algorithm similar to Dijkstra's, but rather than the greedy choice comprising of the minimum of the possible paths from the source to the external vertex, it would be the maximum of the average weight of the edges within possible paths to the external vertex. This should work, since the probabilities are independent.