Final review questions for graphs

1. Name two applications for undirected graphs *(Local area networks: edges are cables and vertices are routers and computers. Printed circuit boards: vertices are junctions and edges are the traces. World Wide Web.)*, and two more applications for directed graphs *(Flight network: edges are flights and vertices are airports. Transportation network: vertices are intersections and edges are one-way streets. Inheritance hierarchy in a UML. Vertices are classes, edges are inheritance relationships).*

2. Among the edges of an undirected graph G are: (A,B) and (B,C). Consider the path p: A, B, C, B, A. Is p a cycle *(No)*? Explain *(Because edges of the path are not distinct)*.

3. Suppose an undirected graph has 10 vertices. What is the maximum number of edges such a graph could have?

$$n = 10 \rightarrow m_{max} = \binom{n}{2} = \frac{n(n-1)}{2} = \frac{10 \times 9}{2} = 45\ edges$$

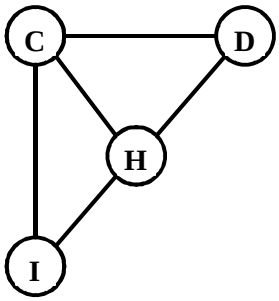
4. Suppose an undirected graph has 10 vertices, v1, v2, . . ., v10, and 20 edges. Joe is counting how many edges come out of each vertex. He discovers that there are d1 edges incident to v1, d2 edges incident to v2, . . ., d10 edges incident to v10. What is the sum d1 + d2 + . . . + d10?

$$\sum_{i=1}^{10} d_i = \sum_{v=1}^{10} \deg(v_i) = 2m = 2 \times 20 = 40$$

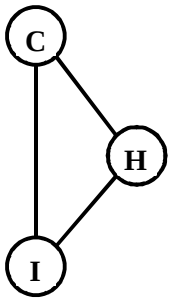
5. Suppose an undirected graph G has 10 vertices and 75 edges. Must G be connected? Explain.
   We calculate the value $\binom{n-1}{2} = \binom{10-1}{2} = \frac{(10-1)(10-2)}{2} = 36 < m = 75$
   ∴ We conclude that this graph is connected.

6. For the graph G below, if W = {C, H, I}, draw the graph G[W].

G:

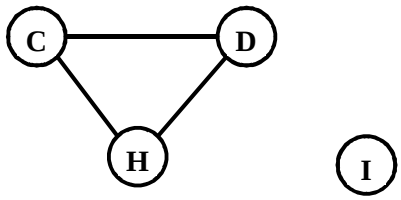


G[W]:



7. If an undirected graph has n vertices and n-1 edges, must it be a tree? If so, prove your answer. If not, give an example to prove your point.

(A tree must be: Acyclic "contains no simple cycles" and connected "there is a path between any two vertices". The shown graph has n = 4 vertices and n – 1 = 3 edges, but it is neither acyclic nor connected. So, it is not a tree).



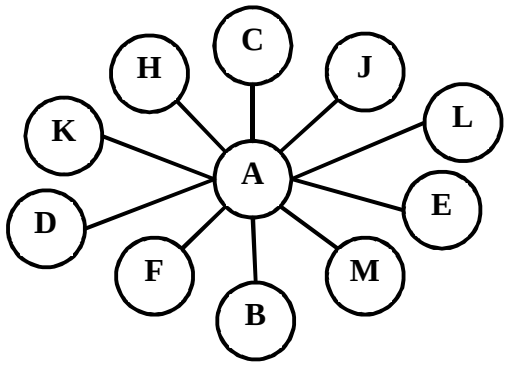
8. Suppose T is a tree with n vertices. How many different rooted trees can be identified using T (without adding or removing any vertices or edges)? *(Since the tree is connected "by definition of tree", then we can perform the "Breadth-First-Search" algorithm on it, starting with any of the vertices, and redraw the whole tree. This can be done uniquely once per vertex, and in total "n" ways).*

9. Is there a connected (undirected) graph having n vertices and n-2 edges? Explain
   No, there isn't.
   Consider a tree with one vertex (n = 1). It has zero edges. If we add another vertex (now n = 2) then to keep the graph connected we need to connect the two vertices with at least one edge (m ≥ 1). Adding another vertex (n = 3) necessitates connecting it to the graph by at least one new edge (m ≥ 2). We notice that each time we add a vertex (n increased by 1) we need an additional edge (m increased by at least 1) to connect this vertex to the graph. And since we started the graph with one node and zero edges, then the minimum number of edges must be n – 1.

10. Does every undirected graph have a vertex cover? Explain.
    Vertex cover: a set of vertices, U, where every edge in the graph is incident to at least one element of U.

    For every graph G= (V, E) there is a vertex cover U = V.

11. Is it possible for a graph with 11 vertices and 10 edges to have a vertex cover consisting of only one vertex?
    Yes. As shown below:



12. What is an adjacency list used for?
    It is used as a guide for traversal through the tree in the Breadth-First-Search algorithm. Using the visited vertex's adjacency list enables visiting all vertices in that list.

~~13. Suppose G is an undirected graph and v is a vertex in V. How many times is v examined during BFS?~~

14. Why is it helpful, when implementing graphs in code, to represent the BFS algorithm as a separate class? In what way is this a useful approach?
    Because the BFS class serves as a super class for other classes that perform operations depending on the BFS operation, e.g. finding the shortest path between two vertices, checking if the graph is connected, knowing number of connected components, checking if the graph contains a cycle. The "Template" design pattern is implemented by adding "`processEdge()`*", "*`processVertex()`*" and "*`additionalProcessing()`*" methods to the class to allow overriding them in the sub classes.*

15. Suppose G is an undirected graph having exactly two connected components. One of the components has n/3 vertices, the other has 2n/3 vertices (assume n is a multiple of 3). Assume also that n < m, where m is the number of edges. What is the asymptotic running time to perform BFS on G?
    Assume the graph has n vertices, m edges, and the components have n_1 and n_2 vertices, and m_1 and m_2 edges, respectively.
    Running time for the first component is $O(m_1 + n_1)$.
    Running time for the second component is $O(m_2 + n_2)$.
    But the BFS is conducted on all components anyway, so its running time is:
    $O(m_1 + n_1 + m_2 + n_2) = O(m + n)$.

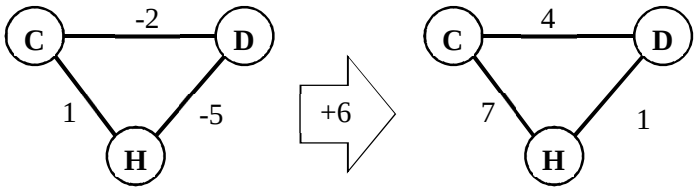
16. Suppose you are given a connected graph G, with vertex set V and edge set E. Give an O(1) algorithm for determining whether G contains a cycle.

    Get no. of edges, m
    Get no. of vertices, n
    If $m \geq n$ then return "the graph has a cycle"
    else return "the graph doesn't contain a cycle"

17. A discrete graph is a graph that has no edges. What is the smallest size of a vertex cover for a discrete graph? *(The empty set {}).*

18. Explain why Dijkstra's Algorithm does not work when it is used on an undirected graph with one or more negative edge weights.
    It doesn't work for two reasons:

a. *Consider the shown graph. We could add a value of 6 to the edge weights to make them positive and create the graph on the right, then apply Dijkstra's algorithm.*



According to the right graph, the shortest path between C & D is edge CD with cost of "4", then if we subtract 6 to get the original cost we get "-2".
However, if we solve it using the original weights, the shortest path between C & D is through edges CH & HD with total cost of -4. So, neither the path nor the cost was correct.

b. *If we have at least one edge with negative weight, then any algorithm (not only Dijkstra's) might give an answer, but it won't be a correct one.*
Ex. *Consider edge CD in the graph above. Dijkstra's algorithm will return its greedy length as -2, but in fact (and since the graph is undirected), we can still find a path with a better greedy length, e.g. C-D-C with cost of -4, and C-D-C-D with cost -6, and we can continue like that infinitely without reaching a proper answer.*

19. Suppose v1, v2, v3, . . ., vm is a shortest path from v1 to vm in a weighted undirected graph. Explain why v1, v2, v3, . . ., vm-1 must be a shortest path from v1 to vm-1.

Assume the following:
q: the length of the path from v_1 to v_m.
r: the length of the path from v_1 to v_{m-1}.
t: the length of the path from v_{m-1} to v_m.
Which means that q = r + t.
Assume that r isn't the shortest path, and that there is another shorter path, r'. This means that there is another path q' = r' + t with shorter length from v_1 to v_m. This contradicts our initial assumption that q = r + t is the shortest path.
So, path $v_1, v_2, \ldots, v_m$ with length q is the shortest path between v_1 and v_m.

20. Explain in a simple way what Kruskal's algorithm does. *(Kruskal's algorithm builds a collection T of edges by doing the following: At each step, add an edge e to T of least weight, such that adding e to T does not create a cycle in T. the algorithm continues until T is a spanning tree (i.e. contains all vertices in the input graph)).* What is the input to the algorithm? *(A simple connected weighted graph G = V, E with n vertices and m edges).* Does Kruskal's algorithm work if there are negative edge weights? *(Kruskal's algorithm works fine with negative numbers).* What is the output of the algorithm? *(A minimum spanning tree T of G).* Without using any kind of optimizations (in particular, without implementing a DisjointSets data structure), what is the running time of Kruskal's algorithm?
Time to sort the edges: O(m.log n).
Time of the while loop that checks the clusters and merges different ones: O(m.n) because:

For each edge (x, y):

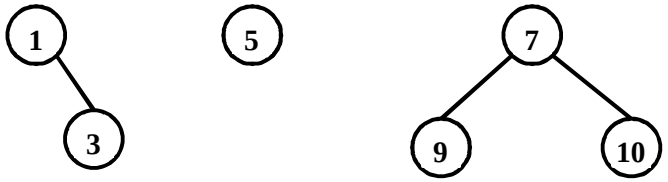
1. *Comparing c(x) with c(y) with hash table implementation of sets: O(n).*
2. *Merging c(x) & c(y): min(|c(x)|, |c(y)|): O(n).*

Therefore, running time of the whole algorithm is O(m.n).

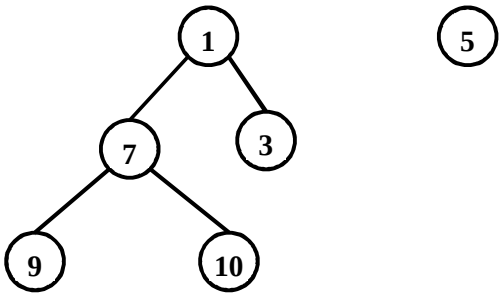
21. Explain why Kruskal's algorithm is an example of a greedy algorithm. Where precisely in the algorithm are greedy decisions made? *(The algorithm is greedy because it considers the edge with the next smallest weight as the next element in the spanning tree (and adds it if possible). It does this by sorting the edges by weight in an ascending order).*

22. What is the DisjointSets data structure? What data does it store? What are its primary operations? What is the running time of each? What is this data structure used for in general? (Do not mention Kruskal's algorithm.)
It is a data structure for maintaining a partition of a set into disjoint subsets. It is sometimes called Partition rather than DisjointSets. Its features are:
    - *Data:*
        1. *Universe U: the base set that is being partitioned. It is never altered.*
        2. *Collection C = {X 1 , X 2 , …, X n } of subsets of the universe. The subsets are disjoint, and their union is U. These subsets are modified when the data structure is used. Size of C shrinks because of repeated union operations.*
    - *Operations:*
        1. *find(x): returns the subset X_i to which x belongs. Its running time is O(log n).*
        2. *union(A,B): replaces the subsets A, B in C with a new subset = A U B. Its running time is O(1).*

23. Suppose U = {1,3,5,7, 9, 10} and C = {{1,3}, {5}, {7,9,10}}. Express these sets as trees (as was done in class). Then show how these trees are modified if we form the union of {1,3} and {7,9,10}.
Sets represented as trees:



Trees after performing the specified union:



24. In Kruskal's algorithm, the first step is to arrange the edges in increasing order of edge weight. Why is this done? What if this step were not taken? How would this affect the output?
The Kruskal's algorithm is a greedy algorithm and its objective is to optimize on a large scale by optimizing on a small scale. It finds the path with the minimum weight by finding the edges with the minimum weights and joining them to form the spanning tree. The sorting step guarantees that and if it is not done then the algorithm fails because it might find a spanning tree, but it will not be a minimum spanning tree.

25. In the optimized version of Kruskal's algorithm given in class, the DisjointSets data structure is used. How is it used? What problem does it solve? How exactly does this data structure improve the performance of Kruskal's algorithm?
    - *The DisjointSet is used in its tree form as follows:*
        - *The elements of each set X in the collection C are represented by nodes in a tree T_X ; the set X itself is referenced by its root r_X.*
        - *find(x) returns the root of the tree to which x belongs*
        - *union(x, y) joins the tree that x belongs to the tree that y belongs to by pointing root of one to the root of the other.*
    - *The DisjointSet solves the problem of slow performance. It does this because:*
        - *The running time of comparing clusters in the original algorithm is O(n), and O(log n) when implementing the DisjointSets data structure.*
        - *The running time of merging clusters in the original algorithm is O(n), and O(1) when implementing the DisjointSets data structure.*
26. In a weighted undirected graph G, if T1 and T2 are disjoint trees why must it be true that if there is an edge (x, y) in G where x is in T1 and y is in T2, then the union T1 U T2 U {(x, y)} is also a tree?
The proof consists of two parts:
    1. *Assume T_1 has n_1 vertices, T_2 has n_2 vertices*
    Then, T_1 has $m_1 = n_1 - 1$ vertices, and T_2 has $m_2 = n_2 - 1$ vertices.
    Let $T_3 = T_1 \cup T_2 \cup (x, y)$.
    Then T_3 has n_3 vertices and m_3 edges, where:
    $m_3 = m_1 + m_2 + 1 = n_1 - 1 + n_2 - 1 + 1 = n_1 + n_2 - 1$
    Since $n_1 + n_2$ is the total number of vertices in the newly formed graph,

i.e. $n_1 + n_2 = n_3$,

$\therefore m_3 = n_3 - 1 \rightarrow (1)$

2. *We know that there is a path* q_1 *from any vertex "a" in* T_1 *to x because* T_1 *is connected. Also, there is a path* q_2 *from any vertex "b" in* T_2 *to y because* T_2 *is connected.*
   It follows that there is a path q_3 *from "a" in* T_1 *to "b" in* T_2*, where:*
   $q_3 = q_1 \cup (x, y) \cup q_2$.
   Therefore, $T_3 = T_1 \cup T_2 \cup (x, y)$ *is connected* $\rightarrow (2)$
   From (1) and (2) we conclude that joining two trees with an edge results in a tree.