Graphs

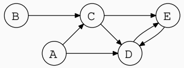
Shiv Shankar Dayal

February 20, 2024

Applications of graphs

- How to find fastest route between different cities of the world?
- How can n jobs be given to n people with maximum total utility?
- How to lay telephone network cables at minimum cost?
- How can a map be colored using four colors in such a way that neghboring regions receive different colors?
- How should a salesman travel so that he can minimize travel time when visiting different cities?

Few Graphs



Terminology

A graph consists of *nodes* or *vertices*. The set of vertices is called *vertex set* and is typically denoted by $V(G) = \{v_1, \dots, v_n\}$, where G is represents the graph.

Nodes or vertices are connected by *arcs* or *edges*. The set of edges or arcs is called *edge set* and is typically denoted by $E(G) = \{e_1, \dots, e_n\}$.

In the diagram (a) shown $V(G) = \{A, B, C, D, E, F, G, H\}$ and $E(G) = \{(A, B), (A, C), (A, D), (C, D), (C, F), (E, G), (A, A)\}$.

When the node pairs, of which, the edges constitute are ordered pairs, the graph is said to be a *directed graph* or *digraph*. Graphs shown (b), (c) and (d) are digraphs.

The head of each arrow represents the second vertex of the ordered pair of vertices making up an edge, and the tail of each arrow represents the first vertex of the ordered pair.

The edge set for the graph (b) is given by $E(G) = \{< A, B >, < A, C >, < A, D >, < C, D >, < F, C >, < E, G >, < A, A >\}$.

Note that a graph is not always a tree, but a tree is always a graph. Also, a vertex need not have a connection mandatorily. As you can see in (a) node H is on its own.

Terminology Contd.

A vertex v is incident to an edge e if v is one the two vertices in the ordered pair of vertices that constitute e. This is written as $u \leftrightarrow v$ to mean "u is adjacent to v". Here u is called *successor* of v and v is called *predecessor* of u.

A *loop* is an edge whose endpoints are equal. For example, $< a, a >$ is a loop. *Parallel edges* or *multiple edges* are the edges that have same pair of endpoints. For example as shown in graph (d).

If n edges are incident on a vertex then that vertex has a *degree* of n. If a vertex has n incoming edge then that is called *indegree* of the vertex and no. of outgoing edges are called *outdegree*.

A *relation* R on a set A is a set of ordered pairs of elements of A.

A graph in which edges have a number associated with them is called a *weighted graph* or a *network* and the number is called the *weight*.

A path of length l from vertex a to vertex b is defined as a sequence of $l + 1$ vertices $v_1, v_2, \ldots, v_{l+1}$ such that $v_1 = a, v_{l+1} = b$ and `adjacent(`$v_i, v_i + 1$`)` is `true` for all i between 1 and l. A path from a vertex to itself is called a *cycle* like vertex A in (a). If a graph contains a cycle, it is *cyclic*m otherwise it is *acyclic*. A directed acyclic graph is called a dag from its acronym. For example, a git source tree is DAG.

Warshall’s Algorithm

Define $\texttt{path}_k[\texttt{i}][\texttt{j}]$ to be `true` if and only if there is a path from vertex `i` to `j` that does not pass through any vertices higher than k(except for `i` and `j`).

Clearly, if $\texttt{path}_k[\texttt{i}][\texttt{j}]$ is `true` then $\texttt{path}_{k+1}[\texttt{i}][\texttt{j}]$ will also be `true`. However, if $\texttt{path}_k[\texttt{i}][\texttt{j}]$ is `false` then for $\texttt{path}_{k+1}[\texttt{i}][\texttt{j}]$ to be `true` there has to be a path from `i` to `j` through `k + 1` but there is no path from `i` to `j` through nodes 1 to k. That is $\texttt{path}_k[\texttt{i}][k+1] == \texttt{true}\&\&\texttt{path}_k[k+1][\texttt{j}] == \texttt{true}$.

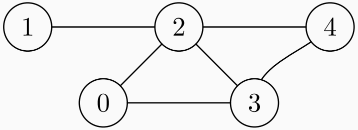
Adjacency List Representation

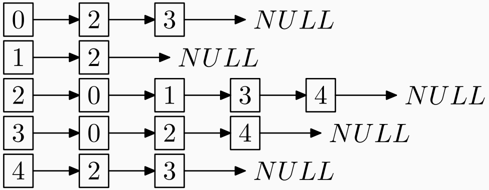
We have seen adjancency matrix representation of graphs. The main problem is that the memory consumption is very high for adjancency matrix representation. To reduce the memory consumption we have an alternative representation for graphs using linked lists which is known as adjacency list representation. An array/vector of

lists stores edges between two vertices. Array's size is equal to the no. of vertices (i.e, `n`). Each index represents a specific vertex in the graph. The entry at the index `i` of the array points to a linked list containing the vertices that are adjacent to vertex `i`.

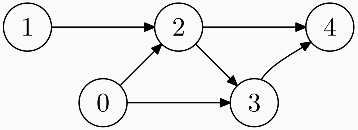
If there are `n` vertices in the graph, then we create an array of list of same size as `adj_list[n]`.

Undirected graph to adjacency list



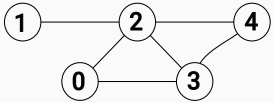


Directed graph to adjacency list

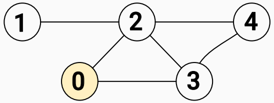


$$\begin{tikzcd}
\boxed{0} \arrow[r] & \boxed{2} \arrow[r] & \boxed{3} \arrow[r] & NULL \\
\boxed{1} \arrow[r] & \boxed{2} \arrow[r] & NULL & \\
\boxed{2} \arrow[r] & \boxed{3} \arrow[r] & \boxed{4} \arrow[r] & NULL \\
\boxed{3} \arrow[r] & \boxed{4} \arrow[r] & NULL & \\
\boxed{4} \arrow[r] & NULL & &
\end{tikzcd}$$

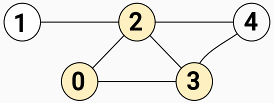
Breadth First Search



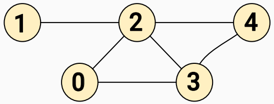
visited					
reached					



visited	0				
reached	0				

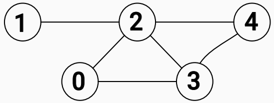


visited	0	2	3		
reached	2	3			

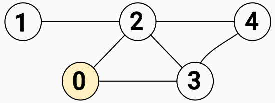


visited	0	2	3	1	4
reached	2	3	1	4	

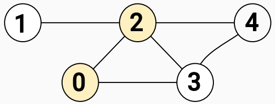
Depth First Search



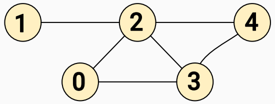
visited					
reached					



visited	0				
reached	2	3			



visited	0	2			
reached	4	1	3		



visited	0	2	4	1	3
reached					

Properties of search algorithms

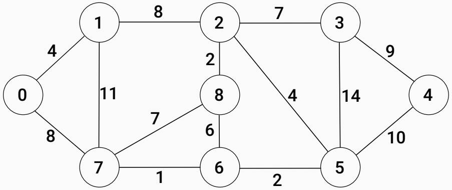
- **Time Complexity:** How much time does the algorithm take?
- **Space Complexity** How much memory does the algorithm consume?
- **Completeness:** Is the algorithm guaranteed to find a solution when there is one, and to correctly report failure when there is not?
- **Cost optimality:** Does it find a solution with the lowest path cost of all solutions?

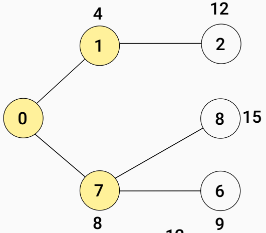
BFS is complete if b is finite, and the state space either has a solution or is finite and cost-optimal if action costs are all identical. b is the branching factor or number of successors of a node that need to be considered.

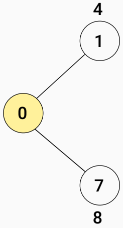
Dijkstra’s Shortest Path Algorithm

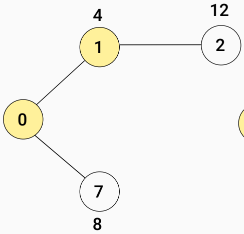
- Create a shotest path tree set that tracks the vertices in the shortest path tree. It is empty to begin with.
- Assign distance values to all vertices in the graph. Initially distance values are infinite except the distance of source from itself which is initiallized to 0.
- while shortest path tree does not include all vertices
    - Pick a vertex u that is not there in shortest path tree and has a minimum distance value and include it to shortest path tree
    - Update the distance value of all adjacent vertices of u
    - For every adjacent vertex v, if the sum of distance value of u from source and weight of the edge u-v is less than the distance value of v, then update the distance value of v.

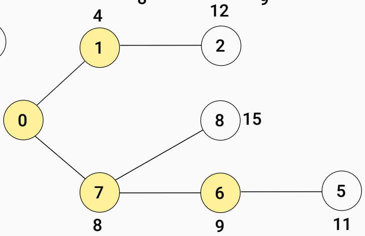
Dijkstra’s Shortest Path Algorithm


0
1
2
3
4
5
6
7
8
4
8
8
7
11
2
7
6
4
14
9
10
1
2


4
12
1
2
0
8
15
7
6
8
9






4
12
1
2
0
8
15
7
6
5
8
9
11