Graphs - used to model pairwise relations between entities.

Graph \( G = (V, E) \) where \( V \) is the set of vertices (nodes) and \( E \) is the set of edges (arcs/links) \( (u,v) \) where \( u, v \in V \).

Un directed graph - no particular ordering of vertices of an edge.

Vertices: \( V = \{ a, b, c, d, e \} \)

Edges: \( E = \{ (a, b), (a, c), (b, c), (b, d), (b, e), (c, d), (d, e) \} \).

Directed graphs - pair of vertices of an edge are ordered.

Edges: \( E = \{ (a, b), (b, c), (b, d), (c, a), (c, d), (d, e), (e, b) \} \).
adjacency (undirected): \( u \) adjacent to \( v \) if \((u,v) \text{ or } (v,u) \in E\)

adjacency (directed): \( (v,w) \) \( \rightarrow \)

A sequence of vertices \( v_1, v_2, \ldots, v_n \) such that an edge exists for every adjacent pair in the sequence.

ex: \( c-a-b-c-d-e \equiv (c,a), (a,b), (b,c), (c,d), (d,e) \)

path length: number of edges in path
ex: \( |P| = 5 \)

distance of vertices \( u, v \): length of shortest path from \( u \) to \( v \).
ex: dist of \( c, b \) = 2

(cycle - path of \( n \) vertices \( v_1, \ldots, v_n \) where \( v_n = v_1 \))

weighted graph - edges have weights/cost.
\( w_{uv} \): weight of edge \((u,v)\)

Can also have weighted undirected graphs.

adjacency in directed graph: \( a \) adjacent to \( b \) if directed edge \( (b,a) \) exists. \( a \) is directly reachable from \( b \).
path length (weighted) - sum of weights on edges of the path
(difference is still length of shortest path)

weighted distance from $c$ to $b$?

not $c-a-b = 3+10$
but $c-d-e-b = 2+2+1 = 5$

Notice: distance from $b$ to $c: 4 + 5$

How to represent/implement graphs?

Pairwise, so natural to use a matrix


adjacency matrix

$A$: size: $|V| \times |V|$

Problem? Space $O(|V|^2)$

Better for when graph is dense

dense graph: $|E| \approx O(|V|^2)$
typically graphs are sparse: $|E| \approx O(|V|)$, so we use adjacency lists.

adjacency list: for each vertex, store a list of adjacent vertices (and weights)
a ⇒ [b(10)]
b ⇒ [c(4), d(5)] \text{ Space: } O(|V| + |E|)
c ⇒ [a(3), d(2)]
d ⇒ [e(2)] \text{ Storing an entry for every node.}
e ⇒ [b(1)] \text{ Storing every edge.}
f ⇒ []

Searching a graph

BFS: start at source, search 1 hop away, 2 hops away...

\[ \begin{array}{c}
\text{src} \\
\downarrow \\
\text{a} \\
\downarrow \\
\text{c} \\
\downarrow \\
\text{b} \\
\downarrow \\
\text{c} \\
\downarrow \\
\text{e} \\
\downarrow \\
\text{f} \\
\end{array} \]

color nodes white - not discovered
gray - discovered

<Pseudocode on slides / next page>

Try on above example:

\[ \begin{array}{c}
\text{white} \\
\text{gray} \\
\end{array} \]

BFS(C, s):

\[ \begin{array}{c}
C: \text{e} \text{ b} \text{ h} \text{ f} \\
\end{array} \]
Run Time? \( |V|, 1E1 \)

\[
\begin{array}{ccc}
(s,a) & (c,e) & (e,h) \\
(s,b) & (c,h) &
\end{array}
\]

Notice every edge appears once, every node considered constant number of times.

\( \Rightarrow \text{Total} \cdot O(|V| + 1E1) \)

How to use BFS to find shortest paths in unweighted graphs?

Keep distance variable with each node.

Initially: \( S: \text{dist} = 0 \)

\( \forall \text{dist} = \infty \) (for all other verts).

Dequeue a node

When checking neighbors, if \( \text{dist} = \infty \) (hasn't been discovered yet), set \( \text{dist} = \text{dequeued node's dist} + 1 \)
BFS(G, s)  // Searches G starting at s

1. For all v, v.color = white
2. s.color = gray
3. Q.enqueue(s)
4. while (Q not empty):
   5. v = Q.dequeue()
   6. for each neighbor u of v:
      7. if u.color == white
      8. u.color = gray
      9. Q.enqueue(u)
BFS-SP(\(G, s\)) // Finds shortest paths from \(s\) to all other nodes in unweighted graph \(G\):

1. For all \(v\), \(v.dist = \infty\).
2. \(s.dist = 0\).
3. \(Q.enqueue(s)\).
4. while (\(Q\) not empty):
   5. \(v = Q.dequeue()\).
   6. for each neighbor \(u\) of \(v\):
      7. if (\(u.dist = \infty\))
      8. \(u.dist = v.dist + 1\)
      9. \(Q.enqueue(u)\).

\[\text{Diagram with nodes and edges labeled with distances}\]

\(Q = s, d, e, h, f\)