Graphs - used to model pairwise relations between entities.

\[ \text{Graph } G = (V, E) \]
\[ V : \text{vertices/nodes} \]
\[ E : \text{edges/arcs/links} \]
\[ (u, v) \text{ where } u, v \in V \]

"Nodes ~ cities"

"Edges ~ roads"

Undirected graph - no particular ordering of vertices of an edge.

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

-or-

\( \{b, a\}, (c, a), (c, b), (d, b), (e, b), (d, c), (e, d)\)

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

"Certain roads blocked off due to snow."

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

adjacency (directed): \( u \rightarrow v \) if \( (u,v) \) \( \in E \). \( v \rightarrow u \).

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

ex: \[ P = c \rightarrow a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \equiv (c,a), (a,b), (b,c), (c,d), (d,e) \]

path length - number of edges in path.

ex: \( |P| = 5 \)

distance of vertices \( u, v = \) length of shortest path from \( u \) to \( v \).

ex: \( \text{dist of } c, b = 2 \)

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

weighted graph - edges have weights/cost.

\( w_{u,v} \) = weight of edge \( (u,v) \).

\[ \begin{array}{c}
\text{(a)} \xrightarrow{10} \text{(b)} \\
\downarrow \quad \downarrow \quad \downarrow \\
\text{(c)} \xrightarrow{3} \quad \text{(e)} \xrightarrow{4} \text{(d)} \xrightarrow{2} \text{(f)}
\end{array} \]

amount of time it takes to traverse the edge.

Can also have weighted undirected graphs.

edge \( (b,c) \) exists if \( c \) is directly reachable from \( b \).
path length (weighted) = sum of weights on edges of the path

distance ~ 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:

\[
A = \begin{bmatrix}
a & b & c & d & e & f \\
10 & - & - & - & - & - \\
4 & 5 & - & - & - & - \\
3 & - & 2 & - & - & - \\
- & - & 2 & - & - & - \\
- & 1 & - & - & - & - \\
- & - & - & - & - & - \\
\end{bmatrix}
\]

A of 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 \underline{spare}: \(|E| \approx O(|V|)\), so we use adjacency lists.

adjacency list: for each vertex, store a list of adjacent vertices (and weights)


\[
\begin{align*}
\text{a} & \rightarrow [b(10)] \\
\text{b} & \rightarrow [c(4), d(5)] \\
\text{c} & \rightarrow [a(5), d(2)] \\
\text{d} & \rightarrow [e(2)] \\
\text{e} & \rightarrow [b(1)] \\
\text{f} & \rightarrow [] \\
\end{align*}
\]

\text{Space: } O(|V| + |E|)

\text{Storing an entry for every node, storing every edge}

\text{Searching a graph}

\text{BFS: start at source, search 1 hop away, 2 hops away}

\begin{tikzpicture}
\node (a) at (0,0) {$\text{src}$};
\node (b) at (1,-1) {}; 
\node (c) at (2,-1) {}; 
\node (d) at (3,0) {}; 
\node (e) at (3,1) {}; 
\node (f) at (4,0) {}; 
\node (g) at (4,-1) {}; 
\node (h) at (5,0) {}; 
\node (i) at (5,1) {}; 
\draw (a) -- (b) node [midway, above] {}; 
\draw (a) -- (c) node [midway, above] {}; 
\draw (b) -- (d) node [midway, above] {}; 
\draw (c) -- (e) node [midway, above] {}; 
\draw (e) -- (f) node [midway, above] {}; 
\draw (e) -- (g) node [midway, above] {}; 
\draw (f) -- (h) node [midway, above] {}; 
\draw (h) -- (i) node [midway, above] {}; 
\end{tikzpicture}

\text{color nodes white - not discovered}
\text{gray - discovered}

\text{Pseudocode on next page}

Try on above example:

\begin{itemize}
\item $\bigcirc$ - white
\item $\bigotimes$ - gray
\end{itemize}

\text{BFS} (G, s, f)

\text{Q: } [s, e, f]
Run Time? \( |V|, |E| \)

- (s,a)
- (c,e)
- (c,h)
- (s,c)
- (c,h)

\[ \begin{array}{cccc}
 s & a & c & b & e & h & f \\
 \hline
 & & & \uparrow & & \uparrow & \uparrow \\
 (0,5) \\
\end{array} \]

Notice every edge appears once

every node considered constant number of times

\[ \Rightarrow \text{Total} \cdot O(|V| + |E|) \]

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

Keep distance variable with each node.

Initially:
- \( s \) \( \text{dist} = 0 \)
- \( v \) \( \text{dist} = \infty \) (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)
if (first < second & first < last) 
    A[0] = first
else if (second < first & second < last) 
    A[0] = second
BFS-SP(G,s) // Finds shortest paths from s to all other nodes in unweighted graph G

1. For all v, v.dist = ∞
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 = ∞)
         8. u.dist = v.dist + 1
         9. Q.enqueue(u)

Q: s a d b e h f

How to modify this for weighted graphs?

Many changes...
Specifically:

Find shortest paths from source \( s \) to all other vertices in \( G \), where \( G \) is:

- directed
- weighted (positive weights), \( w_{uv} \geq 0 \) for all \((u,v) \in E\).

Differences btw unweighted and weighted?

Once we update a distance, what is possible to have time about that node?

\[ \text{Possible to have } \]  
\[ \text{time about that node?} \]

\[ \Rightarrow \text{We will never re-update } \]
\[ \text{v's dist is set to 5} \]
\[ \text{v's dist is set to 5} \]
\[ \text{So we should update } u! \]

(Traversing an edge in the future cannot yield shorter path to v, u, x)
Difference #1

When to update u.dist?

For vertex x, with neighbor u: \( x \rightarrow u \)

If \( u \cdot \text{dist} > x \cdot \text{dist} + w_{x,u} \)

\[ u \cdot \text{dist} = x \cdot \text{dist} + w_{x,u} \]

Another difference?

First, notice we process nodes in ascending order of distance. Why?

\( \Rightarrow \) Want to process a node that is closer to \( s \) before one that is further since the closer node may yield a shorter path to the farther one.

For unweighted, this means processing the next node \( v \) vs in the queue.

For weighted, should process \( x \) before \( u \).

For unweighted, this means

\[ |v| u | x| \]

How to store nodes?

Min Binary Heap ordered by distance

Difference #2

Store nodes in Min Binary Heap ordered by distance.

Turns out (will prove later) when \( u \cdot \text{dist} \) is minimum distance of all non-processed nodes, \( u \cdot \text{dist} \) is shortest distance from \( s \) to \( u \).