Graphs - used to model pairwise relations between entities.

Review

$$\text{Graph} \ (G = (V, E)) \ (V: \text{vertices/nodes})$$

$$\text{E: edges/arc/links} \ (u, v) \ \text{where} \ u, v \in V$$

"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-

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

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

"Certain roads blocked off b/c of snow."

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

adjacency (directed): \( u \) adjacent to \( v \) if \((u,v)\), i.e. \((v,u)\) \( \notin E \).

path: sequence of vertices \( V_1, V_2, \ldots, V_n \) such that an edge exists for every adjacent pair in the sequence.

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

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 \( V_1, \ldots, V_n \) where \( V_1 = V_n \))

weighted graph - edges have weights/cost.

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

Can also have weighted undirected graphs.

edge \((b,c)\) can be directly reachable from \((b, d)\).
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 = 4.75

How to represent/implement graphs?

Pairwise, so natural to use a matrix

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

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 sparse - \( |E| \approx O(|V|) \), so we use adjacency lists.

Adjacency list - for each vertex, store a list of adjacent vertices (and weights)
\[ a \xrightarrow{} [b(10)] \]
\[ b \xrightarrow{} [c(4), d(5)] \]
\[ c \xrightarrow{} [a(3), d(2)] \]
\[ d \xrightarrow{} [e(2)] \]
\[ e \xrightarrow{} [b(1)] \]
\[ f \xrightarrow{} [] \]

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

\text{Storing an entry for every node.}

\text{Storing every edge.}

Searching a graph:

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

\[ \text{color nodes white - not discovered} \]

\[ \text{gray - discovered} \]

\text{<Pseudocode on next page>}

Try on above example:

\[ \circ \text{- white} \]

\[ \circ \text{- gray} \]

**BFS** \((G, s, f)\)

\[ Q: [s, a, c, e, b, e, h, f] \]
Run Time? \( \|V\|, |E| \)

\((s, a) \quad (c, e) \quad (e, h)\)

\((s, c) \quad (c, h)\)

\[
\begin{array}{cccccc}
\text{s} & \text{a} & \text{c} & \text{b} & \text{e} & \text{h} & \text{f} \\
\end{array}
\]

\((0, 5)\)

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 dist = \(\infty\) (hasn't been discovered yet), set dist = 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

5 3 4

3 5 4

3 4 5
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)

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): $W_{u,v} > 0$ for all $(u,v) \in E$.

Differences between unweighted and weighted?

1. **Unweighted**

   ![Unweighted Graph]

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

   $\Rightarrow$ We will never re-update $v$'s dist is set to 5
   $u$: $\cdots$ 6
   $x$: $\cdots$ $\cdots$

   So we should update $u$!

2. **Weighted**

   ![Weighted Graph]

   Possible to have $\phi 18/12$

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 vs in the queue.

\[ 5 \quad \text{(} x \text{) \text{dist}=6} \]

```
5 (v) -> (x) dist=6
```

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

\[ 5 \quad (w) \text{dist}=100 \]

```
5 (v) -> (x) dist=6
```

```
5 (w) -> (x) dist=100
```

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 \( u \).