Graphs - used to model pairwise relations between entities.

Graph \( G = (V, E) \)

- \( V \): vertices/nodes
- \( E \): edges/arcs/links \((u,v)\) where \( u, v \in V\)

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

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

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

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) or (v,u) ∈ E

adjacency (directed): (v,u) → v

⇒ u is a neighbor of v

path sequence of vertices v₁, v₂, ..., vₙ s.t. an edge exists between every consecutive pair in the sequence

ex: P: c → a → b → c → d → 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: dist of c, b = 2 (path: c → a → b)

(cycle - path of n vertices v₁, v₂, ..., vₙ where vₙ = v₁)

weighted graph - edges have weights/cost.

wᵤᵥ = weight of edge (u,v).

amount of time it takes to traverse the edge.

Can also have weighted undirected graphs.
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 \( c - d - e - b = 2 + 2 + 1 = 5 \)

\( b \) to \( c = 4 \neq 5 \)

How to represent/implement graphs?

Pairwise, so natural to use a matrix:

\[
A = \begin{bmatrix}
10 & - & - & - & - & - \\
- & 4 & 5 & - & - & - \\
3 & - & 2 & - & - & - \\
- & - & - & 2 & - & - \\
- & 1 & - & - & - & - \\
- & - & - & - & - & - \\
\end{bmatrix}
\]

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)
$a \Rightarrow [b(10)]$
$b \Rightarrow [c(4), d(5)]$
$c \Rightarrow [a(3), d(2)]$
$d \Rightarrow [e(2)]$
$e \Rightarrow [b(1)]$
$f \Rightarrow []$

**Space:** $O(|V| + |E|)$

Storing an entry for every node.
Storing every edge.

**Searching a graph**

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

```
\[ \begin{array}{c}
\text{a} \\
\text{b} \rightarrow \text{c} \rightarrow \text{h} \\
\text{c} \leftarrow \text{g} \rightarrow \text{e} \rightarrow \text{f} \\
\text{g} \\
\text{h} \\
\text{i} \\
\end{array} \]
```

Color nodes:
- **white** - not discovered
- **gray** - discovered

**Pseudocode on next page**

Try on above example:
- **white**
- **gray**

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

```
Q: [ \text{a} \rightarrow \text{e} \rightarrow \text{h} \rightarrow \text{f} ]
```

$v = x \rightarrow ^{1} a \rightarrow ^{1} x \\
$
Run Time? | V, E |
---|---|
(s,a) | (c,e) | (c,h) |
(s,c) | (c,h) | |
| s | a | c | b | e | h | f |
| (a,b) |

Notice: every edge appears once

every node considered constant number of times

⇒ Total: O(|V| + |E|)

How to use BFS to find shortest paths in unweighted graph:

Keep distance variable with each node.

Initially: s.dist = 0

v.dist = ∞ (for all other verts)

Deque a node

When checking neighbors, if dist = ∞ (hasn’t been discovered yet), set dist = dequeued node’s dist + 1
BFS \((G, s, x)\) //Searches \(G\) for \(x\), starting at \(s\).

1. For all \(v\), \(v\.color = \text{white}\)
2. \(s\.color = \text{gray}\)
3. \(Q\.enqueue(s)\)
4. while \((Q \text{ not empty})\)
5. \(v = Q\.dequeue()\)
6. \(\text{if } v = x\)
   \hspace{1cm} return \(v\)
7. for each neighbor \(u\) of \(v\):
8. \(\text{if } u\.color = \text{white} \quad \text{instead of line 6}\)
9. \(u\.color = \text{gray} \quad \text{could also check}\)
10. \(Q\.enqueue(u) \quad \text{if } u = x \text{ here}\)
    \hspace{1cm} but would not work if \(x = s\).
BFS_SP(G, s) // Finds shortest paths from s to all

// other nodes in unweighted graph G

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

\( Q: s, a, d, v, e, h, f \)