Question. Is there a path from \( s \) to \( e \)?

Def. An undirected graph is **connected**, if there is a path between any pair of nodes.

**Task**: Given a source node \( s \), find all nodes connected to \( s \).

Algorithm 1: GraphSearch\((G(V, E), s)\)

```plaintext
/* G is an undirected graph, s a source node */
1 conn ← \{s\} /\* set of nodes connected to s \*
2 while there is edge \((u, v)\) with \( u \) in conn and \( v \) not in conn do
3     add \( v \) to conn;
4 return conn
```

**Output**: set of vertices

- \( s - c - d \) (**simple path**)
- \( s - c - g - f - c - d \) (not **simple path**)

**Note**: What order to consider the edges? *obscure way to implement things
Graph representation: adjacency matrix

Adjacency matrix: nxn binary matrix $A$, such that $A[i,j]=1$ iff $(i,j)$ is an edge

- **Symmetric**: undirected graph
- **Asymmetric**: directed graph

Directed adjacency matrix:
rows: node of origin of edge $e$
cols: destination node
Adjacency matrix in Good Will Hunting.

https://www.imdb.com/title/tt0119217/
Operations on an adjacency matrix

Adjacency matrix: nxn binary matrix $A$, such that $A[i,j]=1$ iff $(i,j)$ is an edge

\[
\begin{bmatrix}
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0
\end{bmatrix}
\]

How do we perform a lookup? e.g. check whether $(u,v)$ is an edge

Check the value of the matrix $A[u,v]$ at the intersection of row $u$ and column $v$:

- if $A[u,v] = 1$ → connected
- if $A[u,v] = 0$ → disconnected

How many operations is that?

$\Theta(1) = \text{constant time}$
Operations on an adjacency matrix — TopHat

Adjacency matrix: nxn binary matrix A, such that A[i,j]=1 iff (i,j) is an edge

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Question 1.: (multiple choice!)

Complexity of listing all neighbors of vertex v?

A. \( \Theta(1) \)
B. \( \Theta(n) \)
C. \( \Theta(n^2) \)

Complexity of listing every edge in G?

D. \( \Theta(n) \)
E. \( \Theta(n^2) \)
F. \( \Theta(m) \)
Operations on an adjacency matrix — directed

How would you implement the following operations and what is their complexity?

• Lookup: verify whether the pair \((u,v)\) form a directed edge
  
  Check if the intersection of row \(u\) and column \(v\) is 1

• Find the out-neighbors of \(v\), or find the out-degree of \(v\). (degree = number of edges directed away from \(v\))
  
  Count the amount of 1's in the row \(v\)

• Find the in-neighbors of \(v\)
  
  Count the amount of 1's in the column \(v\)

• List every directed edge in \(G\)
  
  ```
  count = 0
  for i=1 to n do:
    for j=1 to n do:
      if A[i,j] == 1 then:
        count += 1
  return count
  ```
**Graph representation: adjacency matrix**

**Adjacency matrix**: nxn binary matrix $A$, such that $A[i,j]=1$ iff (i,j) is an edge

![Graph diagram]

Adjacency matrix:

$$
\begin{bmatrix}
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0
\end{bmatrix}
$$

<table>
<thead>
<tr>
<th>Operation</th>
<th>time complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>check edge (u,v)</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>list all neighbors of u</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>list all edges</td>
<td>$\Theta(n^2)$</td>
</tr>
</tbody>
</table>

**space complexity**: $\Theta(n^2)$
Graph representation: adjacency lists

Adjacency list: For each node \( v \) there is a record listing the nodes to which \( v \) is connected.

Note: vertices in an adjacency list need not appear in any particular order.

Note: uses less space than an adjacency matrix because we only store connected neighbors.
**Graph representation: adjacency list — weighted graphs**

**Adjacency list:** For each node $v$ there is a record listing the (node, weight) pairs to which $v$ is connected.

![Graph diagram](image)

<table>
<thead>
<tr>
<th>NodeId</th>
<th>Neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2 2 3 5 2 6</td>
</tr>
<tr>
<td>2</td>
<td>1 2 4 2 5 7 3 3</td>
</tr>
<tr>
<td>3</td>
<td>2 3 1 5 5 4 7 2 8 3</td>
</tr>
<tr>
<td>4</td>
<td>2 2 5 1</td>
</tr>
<tr>
<td>5</td>
<td>3 3 7 6</td>
</tr>
<tr>
<td>6</td>
<td>3 2 8 7</td>
</tr>
<tr>
<td>7</td>
<td>2 7 4 1 3 4 6 1</td>
</tr>
</tbody>
</table>

If $(u,v)$ is an edge with weight $w$, then we store $(v,w)$ instead of $(v)$ in $u$’s neighbor list.
Graph representation: adjacency list

Adjacency list: For each node $v$ there is a record listing the nodes *towards* which $v$ has a directed edge.

<table>
<thead>
<tr>
<th>NodeId</th>
<th>Neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>a (10), b (5)</td>
</tr>
<tr>
<td>a</td>
<td>b (4), c (15), d (9)</td>
</tr>
<tr>
<td>b</td>
<td>c (8)</td>
</tr>
<tr>
<td>c</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>c (15)</td>
</tr>
</tbody>
</table>
Question:

Let A be an adjacency list. What is the total number of keys stored in A?

A. $\Theta(|V|)$  C. $\Theta(|E|)$  E. $\Theta(|V| + |E|)$
B. $\Theta(|V|^2)$  D. $\Theta(\max\{\text{degree}(v)\})$
Graph representation: adjacency lists

<table>
<thead>
<tr>
<th>NodeId</th>
<th>Neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2, 3</td>
</tr>
<tr>
<td>2</td>
<td>1, 4</td>
</tr>
<tr>
<td>3</td>
<td>2, 5</td>
</tr>
<tr>
<td>4</td>
<td>2, 5</td>
</tr>
<tr>
<td>5</td>
<td>3, 8</td>
</tr>
<tr>
<td>6</td>
<td>2, 4</td>
</tr>
<tr>
<td>7</td>
<td>3, 8</td>
</tr>
<tr>
<td>8</td>
<td>3, 8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
<th>time complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>check edge (u,v)</td>
<td>$\Theta(\delta(u))$</td>
</tr>
<tr>
<td>find all neighbors of u</td>
<td>$\Theta(\delta(u))$</td>
</tr>
<tr>
<td>find all edges</td>
<td>$\Theta(</td>
</tr>
</tbody>
</table>

- space complexity: $\Theta(|V| + |E|)$
Adjacency list — space complexity analysis

Claim. The size of the adjacency list is $\Theta(n + m)$

Note. using $n+m$ is the preferred notation. $\Theta(n + m) = \Theta(\max\{n, m\})$

$\Theta(n) =$ the size of list containing nodes id
$\Theta(m) =$ each edge in this data structure is represented as a node on the right side of the list of nodes (counted twice)

We need at least $\Theta(n)$ to represent the nodes
We need at least $\Theta(m)$ to represent the edges

Thus, we need $\Theta(\max\{n, m\})$ to represent the adjacency list $\Rightarrow \Theta(\max(n, m)) = \Theta(n + m)$
Adjacency list — space complexity analysis

**Claim.** The size of the adjacency list is $\Theta(n + m)$

**Note.** using $n+m$ is the preferred notation. $\Theta(n + m) = \Theta(\max\{n, m\})$

**proof 1:**
- $\Theta(n)$ : there is a slot in the main list for every vertex $v$ even if their degree is 0
- $\Theta(m)$ : there is a list of size $\delta(v)$ for every $v$ in the main list
  - For each $v$ the size of the list containing $v$'s neighbors is degree($v$)
  - We know that each edge ($v,w$) contributes once to the degree of $v$ and once to $w$

$$\sum_{v \in V} \text{degree}(v) = 2m = \Theta(m)$$

**proof 2:**
- $\Theta(n)$ : there is a slot in the main list for every vertex $v$ even if their degree is 0
- $\Theta(m)$ : every neighbor of $v$ is represented on the right side of $v$
  - Each edge ($v,w$) is represented in the table twice; $w$ is in the neighbor list of $w$ and vice versa.

**Exercise.** Repeat the same proof on directed graphs and the directed adj list.
**n^2 vs. m in graphs**

Why is the use of adjacency lists, with $\Theta(n + m)$ complexity considered better than the $\Theta(n^2)$ of the adjacency matrix?

Intuitively, most of the time a graph contains more edges than vertices. So, why use $\Theta(n+m)$ instead $\Theta(n^2)$?

\[
\Theta(n^2) = \frac{n(n-1)}{2} \leq c \cdot n^2
\]

(Because every edge is counted twice)

What is the minimum amount of edges in a connected graph?

$n - 1 \leq m \leq n^2$

Exercise. Compute the minimum number of edges in a connected graph on n vertices.
Computing the running time in loops

A is the adjacency matrix of a graph, G is the adjacency list of the same graph.

Algorithm 1: MatrixCount(A)

1. \( \text{count} \leftarrow 0; \)
2. \( \textbf{for} \ i = 1 \ \textbf{to} \ n \ \textbf{do} \)
3. \( \quad \textbf{for} \ j = 1 \ \textbf{to} \ n \ \textbf{do} \)
4. \( \quad \quad \textbf{if} \ A[i,j] == 1 \ \textbf{then} \)
5. \( \quad \quad \quad \text{count} + = 1; \)
6. \( \textbf{return} \ \text{count}/2 \)

Algorithm 2: ListCount(G)

1. \( \text{count} \leftarrow 0; \)
2. \( \textbf{for} \ v \ \text{in} \ G \ \textbf{do} \)
3. \( \quad \textbf{for} \ w \ \text{in} \ G[v] \ \textbf{do} \)
4. \( \quad \quad \text{count} + = 1; \)
5. \( \textbf{return} \ \text{count}/2 \)

Same algorithm, but using different data structures.
Both return the number of edges.
Computing the running time in loops - TopHat

A is the adjacency matrix of a graph, G is the adjacency list of the same graph.

Algorithm 1: MatrixCount(A)

1. \( count \leftarrow 0; \)
2. for \( i = 1 \) to \( n \) do \( \Rightarrow n \) times
3. \hspace{1em} for \( j = 1 \) to \( n \) do \( \Rightarrow n \) times
4. \hspace{2em} if \( A[i, j] == 1 \) then \( \Theta(1) \)
5. \hspace{2em} \hspace{1em} count++ = 1;
6. return \( count/2 \)

(multiple choice!)

What is the running time of MatrixCount?

A. \( \Theta(n) \)
B. \( \Theta(n^2) \)

What is the running time of ListCount?

C. \( \Theta(n + m) \)
D. \( \Theta(nm) \)
E. \( \Theta(n \cdot deg_{\text{max}}) \)

The length of the loop depends on the amount of neighbors of \( v \).
Computing the running time in loops

A is the adjacency matrix of a graph, G is the adjacency list of the same graph.

**Algorithm 1: MatrixCount(A)**

1. `count ← 0;`
2. `for i = 1 to n do`
   3. `for j = 1 to n do`
      4. `if A[i, j] == 1 then`
         5. `count++ = 1;`
3. `return count/2`

If the number of iterations in the inner loop is *fixed*, then we can multiply to get the number of iterations.:  

n x n iterations, O(1) each results in O(n^2)

**Algorithm 2: ListCount(G)**

1. `count ← 0;`
2. `for v in G do`
   3. `for w in G[v] do`
      4. `count++ = 1;`
3. `return count/2`

If the number of iterations is *different* each time then compute the total number of operations across the iterations.:  

- the outer loop visits each node ones: O(n)
- the inner loop iterates over each edge in G, the counter is increased once for each edge: O(m)
- total: O(n+m)
Breadth First Search (BFS)

Breadth first search.

- Task: Given a source node $s$, find the shortest path from $s$ to each node $v$ that it is connected to.

\[
\text{BFS Tree (represented by red edges)}
\]
- Contains all nodes of the original graph
- Has the minimum amount of edges

shortest path between nodes $u$ and $v = \text{the connecting path with the least edges}
Breadth first search.

- Task: Given a source node \( s \), find the \textit{shortest} path from \( s \) to each node \( v \) that it is connected to.

\[\text{BFS search tree: an edge connecting each node to the neighbor through which it was discovered.}\]
Breadth first search — order of node processing

Green: alternative order of node processing.

BFS search tree