Universal Sink Finding Algorithm

Subhabrata Samajder

IIIT, Delhi
Winter Semester,
10^{th} May, 2023

Data Structures for Graphs

Convention

- **Vertices:** Are always numbered $1, 2, \ldots, n$ or $0, 1, \ldots, n-1$.
- $|E| = m$.
- **Size of Input:** Usually measured in terms of the number of vertices $|V|$ and the number of edges $|E|$ of the graph.
- That is, there are two relevant parameters describing the size of the input, not just one.

Representation

There are two standard ways to represent a graph $G = (V, E)$*:*

1. as a collection of adjacency lists or
2. as an adjacency matrix.

Adjacency-list Representation

- Consists of an array Adj of $|V|$ lists, one for each vertex in V.
- For each $u \in V$, the adjacency list

$$Adj[u] = \{v : (u, v) \in E\}.$$

- That is, $Adj[u]$ consists of all the vertices adjacent to u.
- Alternatively, it may contain pointers to these vertices.
- **Note:** The vertices in each adjacency list are typically stored in an arbitrary order.

Adjacency-list Representation (Cont.)

- **Directed Graphs:** Sum of the lengths of all adjacency lists is $|E|$, since an edge (u, v) appears only in $Adj[u]$.
- **Undirected Graphs:** Sum of the lengths of all adjacency lists is $2|E|$, since an edge (u, v) appears both in $Adj[u]$ and $Adj[v]$.
- **Memory Requirement:** For both directed and undirected graphs, it requires $\Theta(|V| + |E|)$ memory.
- **Weighted Graphs:** The weight $w(u, v)$ of the edge $(u, v) \in E$ is simply stored with vertex v in u's adjacency list.

Adjacency-matrix Representation

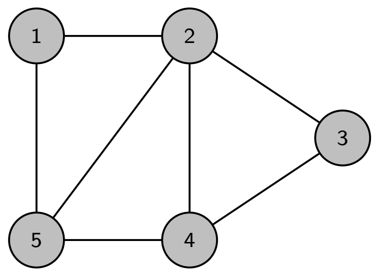
- **Assumption:** Vertices are numbered $1, 2, \ldots, |V|$.
- Consists of a $|V| \times |V|$ matrix $A = (a_{ij})$, s.t.,

$$a_{ij} = \begin{cases} 1; & \text{if } (i,j) \in E \\ 0; & \text{otherwise.} \end{cases}$$

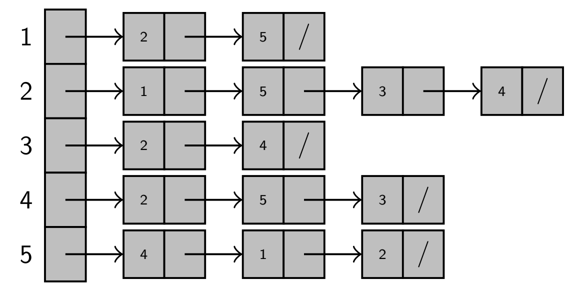
- **Memory Requirement:** $\Theta(|V|^2)$
- **Undirected Graph:** Symmetric matrix.
  - Suffices to store only the entries on and above the diagonal.
  - Cutting the memory needed to store the graph almost by half.
- **Directed Graph:** Not necessarily a symmetric matrix.
- **Weighted Graphs:** Store

$$a_{ij} = \begin{cases} w(i,j); & \text{if } (i,j) \in E \\ 0; & \text{otherwise.} \end{cases}$$

Example: Undirected Graph



An undirected graph G



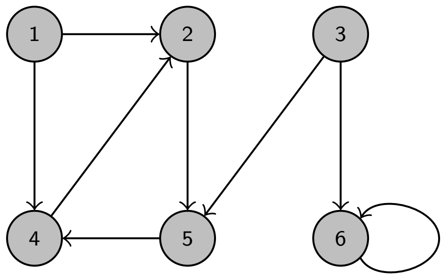
An adjacency-list representation of G

	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

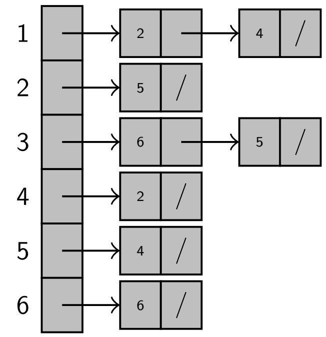
The adjacency-matrix representation of G

Note: The adjacency-matrix is symmetric!

Adjacency-list Representation: Directed Graph Example



An directed graph G



An adjacency-list representation of G

	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

The adjacency-matrix representation of G

Note: The adjacency-matrix is not necessarily symmetric!

The Adjacency-list Representation: Pros vs. Cons.

Pros:

- Quite Robust: Can be modified to support many other graph variants.
- **Space Efficient:** Takes $\Theta(|V| + |E|)$ amount of memory.
- Neighbors of a vertex can be computed in optimal time.

Cons:

- No quicker way to determine if a given edge (u, v) is present in the graph than to search for v in the adjacency list $Adj[u]$.
  - **Complexity:** $\mathcal{O}(|V|)$.
  - Can be remedied by an adjacency-matrix representation of the graph, at the cost of using asymptotically more memory.

The Adjacency-matrix Representation: Pros vs. Cons.

Pros:

- Determining whether there is an edge from u to v takes $\mathcal{O}(1)$.

Cons:

- Computing all neighbors of a given vertex v takes $\mathcal{O}(|V|)$ time.
- Takes $\Theta(|V|^2)$ amount of memory.

The Adjacency-list vs. Adjacency-matrix Representation

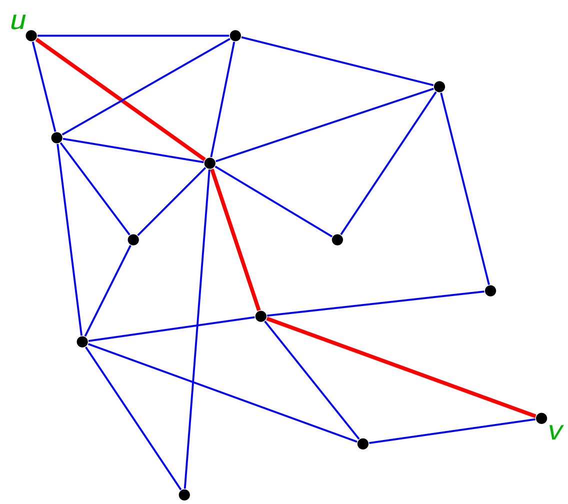
- Although the adjacency-list representation is asymptotically at least as efficient as the adjacency-matrix representation, the simplicity of an adjacency matrix may make it preferable when graphs are reasonably small, i.e., $|V|$ is small.
- **Unweighted Graphs:** Rather than using one word of computer memory for each matrix entry, one bit can be used per entry of the matrix, thus making it more space efficient.
- **Commonly Used:** Adjacency-lists.

  Reasons:
  - Graphs in real life are sparse ($|E| \ll |V|^2$).
  - Most algorithms require processing neighbours of each vertex.

  But, there are a few exceptions.

Graph Traversal

Definition: A vertex v is said to be **reachable** from u if there is a path from u to v.



Graph traversal from vertex u: Visit all vertices which are reachable from u.

Non-triviality of Graph Traversal

- **Avoiding Loops:** How to avoid visiting a vertex multiple times?

Non-triviality of Graph Traversal

- **Avoiding Loops:** How to avoid visiting a vertex multiple times?

  Ans: Keeping track of vertices already visited.

Non-triviality of Graph Traversal

- **Avoiding Loops:** How to avoid visiting a vertex multiple times?

  Ans: Keeping track of vertices already visited.

- **Finite number of steps:** The traversal must stop in finite number of steps.

- **Completeness:** We must visit all vertices reachable from the start vertex u.

Some Interesting Graph Algorithmic Problems

- Are two vertices u and v connected?
- Find all connected components in a graph.
- Is there is a cycle in a graph?
- Compute a path of shortest length between two vertices?
- Is there is a cycle *passing through all vertices*?

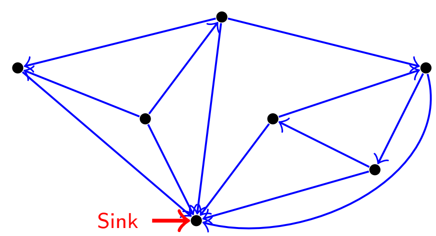
Some Interesting Graph Algorithmic Problems

- Are two vertices u and v connected?
- Find all connected components in a graph.
- Is there is a cycle in a graph?
- Compute a path of shortest length between two vertices?
- Is there is a cycle *passing through all vertices*?
  - Hamiltonian cycle problem.

An Interesting Graph Problem: Finding a Sink

Definition: A vertex s in a given directed graph is said to be a **universal sink** if

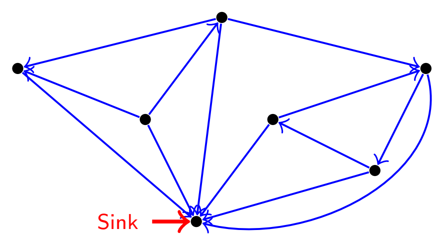
- there is no edge emanating from (leaving) s and
- every other vertex has an edge into s.



An Interesting Graph Problem: Finding a Sink

Definition: A vertex s in a given directed graph is said to be a **universal sink** if

- there is no edge emanating from (leaving) s and
- every other vertex has an edge into s.

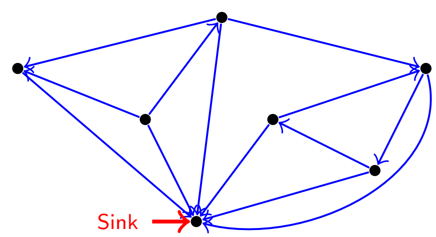


How many sinks can there be in a graph G?

An Interesting Graph Problem: Finding a Sink

Definition: A vertex s in a given directed graph is said to be a **universal sink** if

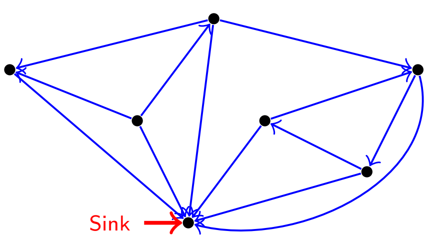
- there is no edge emanating from (leaving) s and
- every other vertex has an edge into s.



How many sinks can there be in a graph G? At most 1.

An Interesting Graph Problem: Finding a Sink

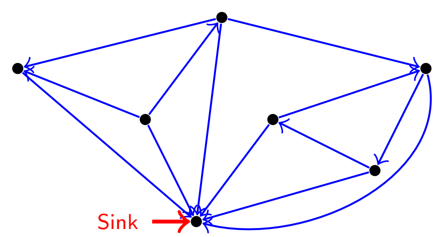
Problem: Given a directed graph $G = (V, E)$ in an adjacency matrix representation, design an $\mathcal{O}(|V|)$ time algorithm to determine if there is any sink in G.



An Interesting Graph Problem: Finding a Sink

Problem: Given a directed graph $G = (V, E)$ in an adjacency matrix representation, design an $\mathcal{O}(|V|)$ time algorithm to determine if there is any sink in G.

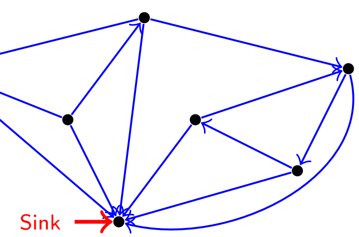
Hint: We can only look into $\mathcal{O}(|V|)$ entries of the Adjacency matrix M.



An Interesting Graph Problem: Finding a Sink

Problem: Given a directed graph $G = (V, E)$ in an adjacency matrix representation, design an $\mathcal{O}(|V|)$ time algorithm to determine if there is any sink in G.

Hint: We can only look into $\mathcal{O}(|V|)$ entries of the Adjacency matrix M.

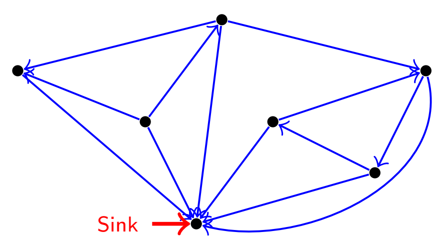


Question: Can we efficiently verify whether any given vertex i is a sink?

An Interesting Graph Problem: Finding a Sink

Problem: Given a directed graph $G = (V, E)$ in an adjacency matrix representation, design an $\mathcal{O}(|V|)$ time algorithm to determine if there is any sink in G.

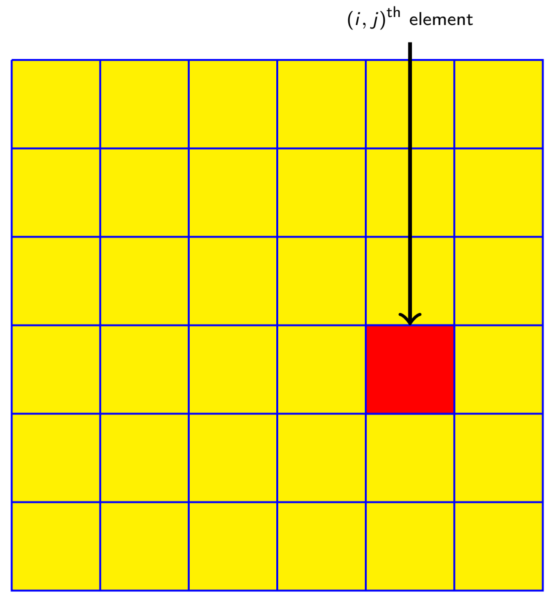
Hint: We can only look into $\mathcal{O}(|V|)$ entries of the Adjacency matrix M.



Question: Can we efficiently verify whether any given vertex i is a sink?

Answer: Yes, in $\mathcal{O}(|V|)$ time only. (Look at i^{th} row and j^{th} column of M.)

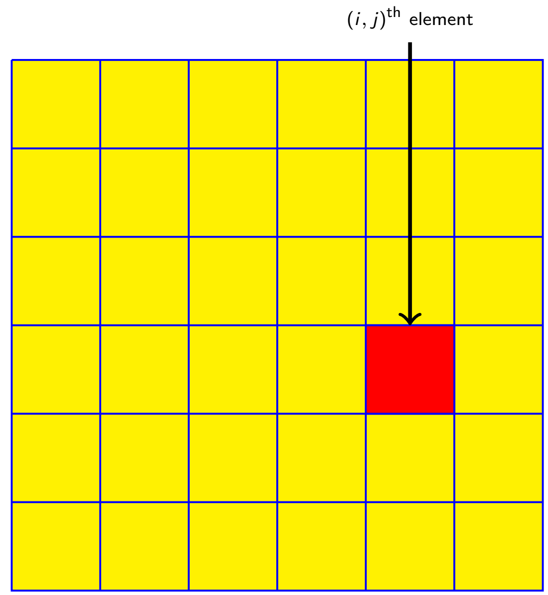
Main Idea



Adjacency Matrix M

Question: Can we eliminate $|V| - 1$ non-sink vertices in $\mathcal{O}(|V|)$ look-up into the Adjacency Matrix M?

Main Idea



Adjacency Matrix M

Question: Can we eliminate $|V|-1$ non-sink vertices in $\mathcal{O}(|V|)$ look-up into the Adjacency Matrix M?

Note:

- If $M[i,j]=0$, then j cannot be a sink.
- If $M[i,j]=1$, then i cannot be a sink.

Algorithm: UNIVERSALSINK(M)

I/P: A $|V| \times |V|$ adjacency-matrix of a graph $G = (V, E)$.

```
Begin
   i = 1;
   j = 1;
   while (i ≤ |V| and j ≤ |V|) {
      if (M[i, j] == 1)
         i = i + 1;
      else
         j = j + 1;
   }
   if (i > |V|)
      print "there is no universal sink"
   else
      if (IsSink(M, i) == False)
         print "there is no universal sink"
      else
         print "i is the universal sink"
End
```

$\textsc{UniversalSink}(M)$: Correctness and Complexity

Note:

- Loop terminates when either $i > |V|$ or $j > |V|$.
- Upon termination, only i could possibly be a sink.
  - If $i > |V|$: There is no sink.
  - If $i \leq |V|$: Then $j > |V|$. Note that,
    - Vertices k where $1 \leq k < i$ cannot be sinks.
    - Vertices k where $i < k \leq |V|$ cannot be sinks.
- Check whether i is a sink or not.
  - Out Degree is 0? i^{th} row must be an all zero row.
  - In Degree is $|V| - 1$? i^{th} column must contain all 1's except for the $(i, i)^{\text{th}}$ entry, which must be zero by the first condition.

UniversalSink(M): Correctness and Complexity

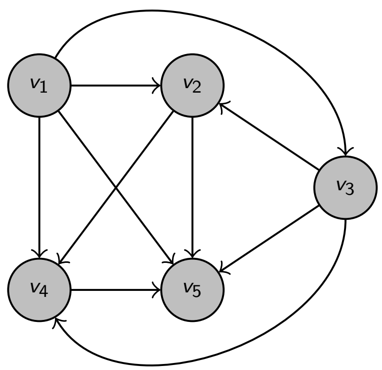
Note:

- Loop terminates when either $i > |V|$ or $j > |V|$.
- Upon termination, only i could possibly be a sink.
  - If $i > |V|$: There is no sink.
  - If $i \leq |V|$: Then $j > |V|$. Note that,
    - Vertices k where $1 \leq k < i$ cannot be sinks.
    - Vertices k where $i < k \leq |V|$ cannot be sinks.
- Check whether i is a sink or not.
  - Out Degree is 0? i^{th} row must be an all zero row.
  - In Degree is $|V| - 1$? i^{th} column must contain all 1's except for the $(i, i)^{\text{th}}$ entry, which must be zero by the first condition.

Worst-case Complexity: $\mathcal{O}(|V|)$

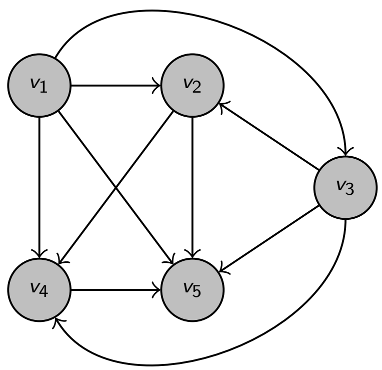
- At most $2|V|$ for the while loop.
- $2|V|$ for IsSink(M, i).

UniversalSink(M): Example



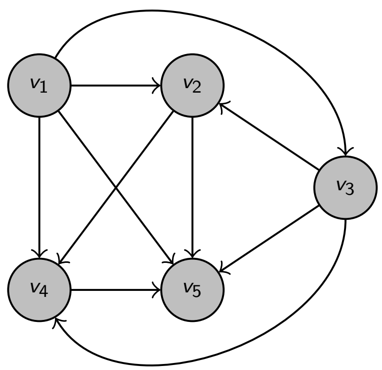
$$
\begin{array}{cc}
\begin{array}{ccccc} \mathbf{v_1} & \mathbf{v_2} & \mathbf{v_3} & \mathbf{v_4} & \mathbf{v_5} \end{array} & \\
\begin{pmatrix}
0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0
\end{pmatrix} & \begin{array}{c} \mathbf{v_1} \\ \mathbf{v_2} \\ \mathbf{v_3} \\ \mathbf{v_4} \\ \mathbf{v_5} \end{array}
\end{array}
$$

UNIVERSALSINK(M): Example



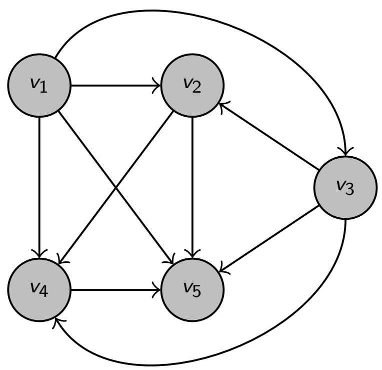
$$
\begin{array}{cc}
\begin{array}{ccccc} \mathbf{v_1} & \mathbf{v_2} & \mathbf{v_3} & \mathbf{v_4} & \mathbf{v_5} \end{array} & \\
\begin{pmatrix}
0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0
\end{pmatrix} &
\begin{array}{c} \mathbf{v_1} \\ \mathbf{v_2} \\ \mathbf{v_3} \\ \mathbf{v_4} \\ \mathbf{v_5} \end{array}
\end{array}
$$

UniversalSink(M): Example



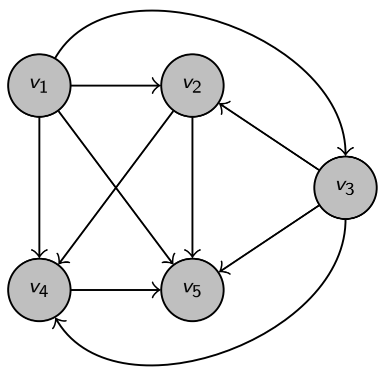
$$
\begin{array}{cc}
\begin{array}{ccccc} \mathbf{v_1} & \mathbf{v_2} & \mathbf{v_3} & \mathbf{v_4} & \mathbf{v_5} \end{array} & \\
\begin{pmatrix}
0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0
\end{pmatrix} &
\begin{array}{c} \mathbf{v_1} \\ \mathbf{v_2} \\ \mathbf{v_3} \\ \mathbf{v_4} \\ \mathbf{v_5} \end{array}
\end{array}
$$

$\textsc{UniversalSink}(M)$: Example



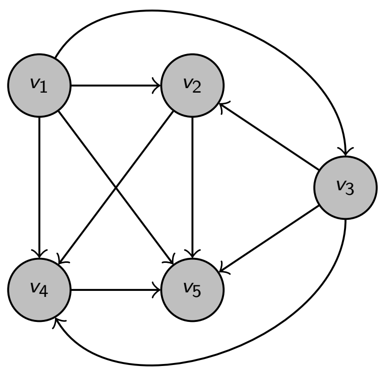
$$
\begin{array}{c@{}c}
\begin{array}{ccccc} \mathbf{v_1} & \mathbf{v_2} & \mathbf{v_3} & \mathbf{v_4} & \mathbf{v_5} \end{array} & \\
\begin{pmatrix}
0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0
\end{pmatrix} &
\begin{array}{c} \mathbf{v_1} \\ \mathbf{v_2} \\ \mathbf{v_3} \\ \mathbf{v_4} \\ \mathbf{v_5} \end{array}
\end{array}
$$

UniversalSink(M): Example



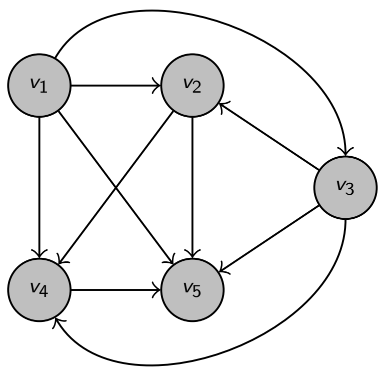
$$
\begin{array}{cc}
\begin{array}{ccccc} \mathbf{v_1} & \mathbf{v_2} & \mathbf{v_3} & \mathbf{v_4} & \mathbf{v_5} \end{array} & \\
\begin{pmatrix}
0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0
\end{pmatrix} & \begin{array}{c} \mathbf{v_1} \\ \mathbf{v_2} \\ \mathbf{v_3} \\ \mathbf{v_4} \\ \mathbf{v_5} \end{array}
\end{array}
$$

UniversalSink(M): Example



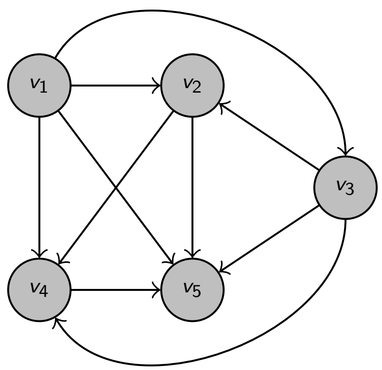
$$
\begin{array}{c|c}
\begin{matrix} v_1 & v_2 & v_3 & v_4 & v_5 \end{matrix} & \\
\begin{pmatrix}
0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0
\end{pmatrix} & \begin{matrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{matrix}
\end{array}
$$

UNIVERSALSINK(M): Example



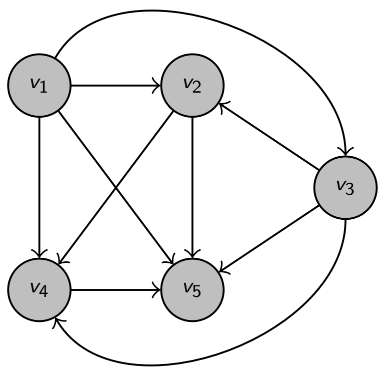
$$
\begin{array}{cc}
\begin{matrix} \mathbf{v_1} & \mathbf{v_2} & \mathbf{v_3} & \mathbf{v_4} & \mathbf{v_5} \end{matrix} & \\
\begin{pmatrix}
0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0
\end{pmatrix} & \begin{matrix} \mathbf{v_1} \\ \mathbf{v_2} \\ \mathbf{v_3} \\ \mathbf{v_4} \\ \mathbf{v_5} \end{matrix}
\end{array}
$$

UniversalSink(M): Example



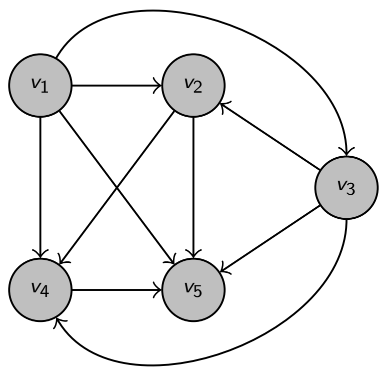
$$
\begin{array}{c c}
\begin{array}{ccccc} \mathbf{v_1} & \mathbf{v_2} & \mathbf{v_3} & \mathbf{v_4} & \mathbf{v_5} \end{array} & \\
\begin{pmatrix}
0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0
\end{pmatrix}
&
\begin{array}{c} \mathbf{v_1} \\ \mathbf{v_2} \\ \mathbf{v_3} \\ \mathbf{v_4} \\ \mathbf{v_5} \end{array}
\end{array}
$$

$\textsc{UniversalSink}(M)$: Example



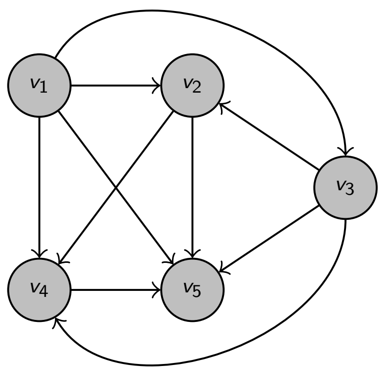
$$
\begin{array}{c c}
\begin{array}{ccccc} \mathbf{v_1} & \mathbf{v_2} & \mathbf{v_3} & \mathbf{v_4} & \mathbf{v_5} \end{array} & \\
\begin{pmatrix}
0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0
\end{pmatrix} & \begin{array}{c} \mathbf{v_1} \\ \mathbf{v_2} \\ \mathbf{v_3} \\ \mathbf{v_4} \\ \mathbf{v_5} \end{array}
\end{array}
$$

UNIVERSALSINK(M): Example



$$
\begin{array}{c}
\begin{array}{ccccc} \mathbf{v_1} & \mathbf{v_2} & \mathbf{v_3} & \mathbf{v_4} & \mathbf{v_5} \end{array} \\
\left(\begin{array}{ccccc}
0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0
\end{array}\right)
\end{array}
\begin{array}{c} \\ \mathbf{v_1} \\ \mathbf{v_2} \\ \mathbf{v_3} \\ \mathbf{v_4} \\ \mathbf{v_5} \end{array}
$$

$\textsc{UniversalSink}(M)$: Example



$$
\begin{array}{cc}
\begin{array}{ccccc} \mathbf{v_1} & \mathbf{v_2} & \mathbf{v_3} & \mathbf{v_4} & \mathbf{v_5} \end{array} & \\
\begin{pmatrix}
0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0
\end{pmatrix} &
\begin{array}{c} \mathbf{v_1} \\ \mathbf{v_2} \\ \mathbf{v_3} \\ \mathbf{v_4} \\ \mathbf{v_5} \end{array}
\end{array}
$$

Thank You for your kind attention!

Books and Other Materials Consulted

1. *Introduction to Algorithms* by Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein.

2. Graph Theory part taken from Discrete Mathematics Lecture Notes (M. Tech (CS), Monsoon Semester, 2007) taught by Prof. Palash Sarkar (ASU, ISI Kolkata).

3. Taken from Prof. Surendar Baswana (CSE, IIT Kanpur) lecture slides.

Questions!!