

DSC40B:
Theoretical Foundations of Data
Science II

Lecture 10: *Graphs: basics and
representations*

Instructor: Yusu Wang

News

- ▶ Midterm 1 grades released
- ▶ Redemption of midterm- 1 will be in the final exam week
- ▶ Recap video with solutions/explanations will be posted soon.



Graphs: directed and undirected



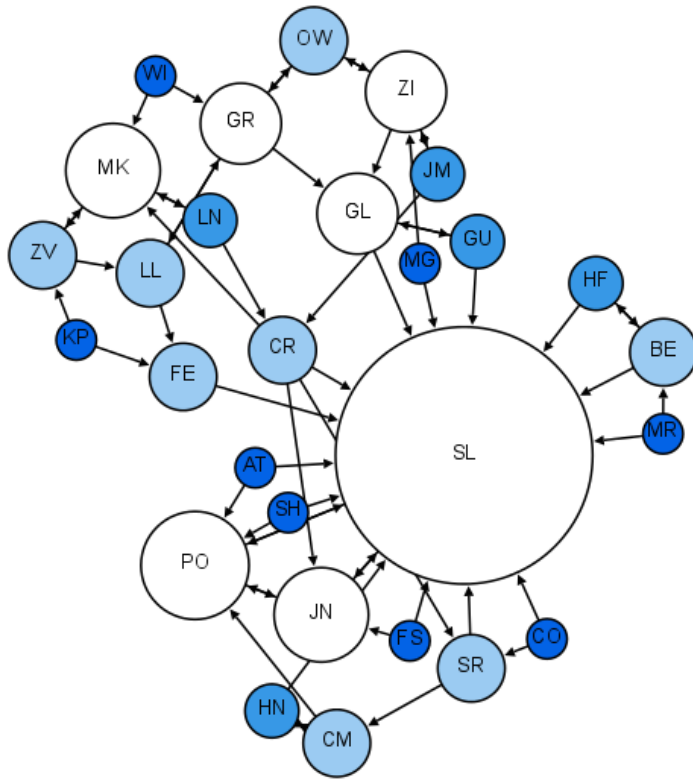
Common data types

- ▶ A set of feature vectors in some feature space (e.g, R^d)
 - ▶ A collection of patients' record, each represented by a feature vector containing information such as name, age, health history etc.
 - ▶ Focus on attributes of individuals
- ▶ Graph data:
 - ▶ Focus on (pairwise) relations among individuals
 - ▶ Social networks, co-authorship networks, knowledge graphs



Examples

► Social networks



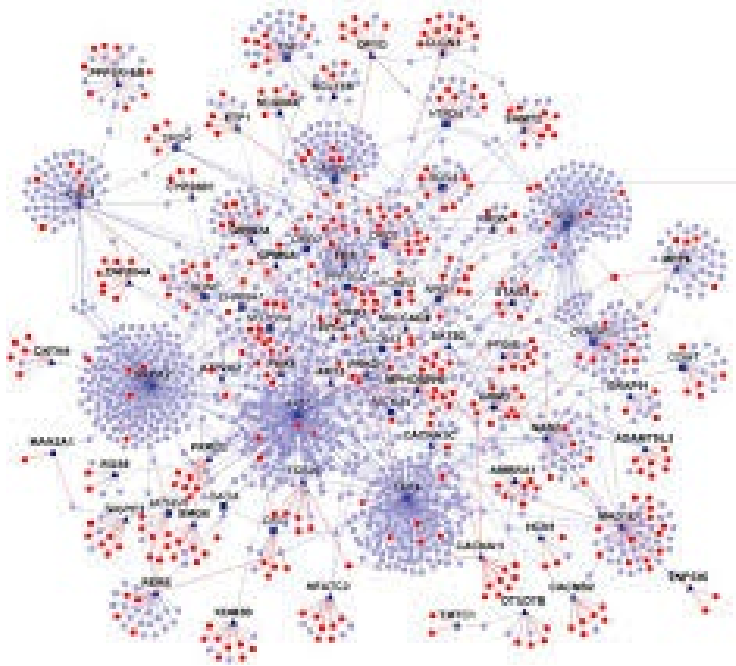
Moreno's sociogram of a 2nd grade class



Facebook friendship social graph

Examples

► Biological networks



Protein-protein interaction networks

► Road networks



Tokyo subway map

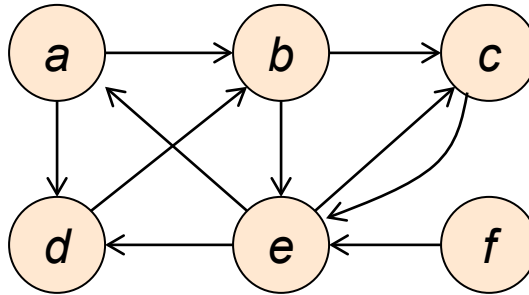
What is a graph?

- ▶ On the high level, a **graph** G is a pair $G = (V, E)$
 - ▶ V : a set of graph **nodes** (or vertices)
 - ▶ $E \subset V \times V$: a set of graph **edges**
 - ▶ each edge $(a, b) \in E$ represents a certain relation between the pair of graph nodes $a, b \in V$
- ▶ Directed vs undirected graphs



Directed graphs

- ▶ A **directed graph** (or **digraph**) G is a pair (V, E) where V is a finite set of **nodes** and E is a set of **ordered pairs** called (directed) **edges**.



- ▶ $G = (V, E)$ where
 - ▶ $V = \{a, b, c, d, e, f\}$;
 - ▶ $E = \{(a, b), (a, d), (b, c), (b, e), (c, e), (d, b), (e, a), (e, c), (e, d), (f, e)\}$



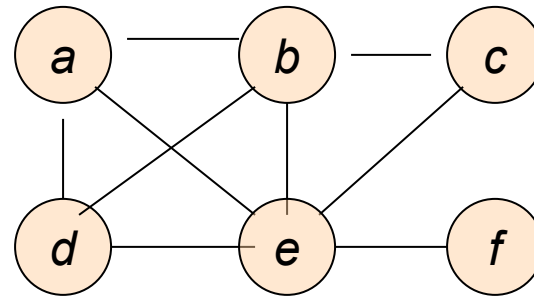
Remarks

- ▶ Note that each edge $(a, b) \in V \times V$ is an ordered pair
 - ▶ meaning that there is an edge from a to b
- ▶ Hence edges $(a, b) \neq (b, a)$
 - ▶ e.g, Alex follows Elon Musk in twitter, so $(\text{Alex}, \text{Elon})$ is an edge in E . But Elon does not follow Alex, so $(\text{Elon}, \text{Alex}) \notin E$
 - ▶ Note that both (a, b) and (b, a) could be in the edge set
- ▶ There can also be self-loop: (a, a)
- ▶ Simple graph:
 - ▶ for any **ordered pair**, there can be at most one edge in E



Directed graphs

- ▶ A **undirected graph** G is a pair (V, E) where V is a finite set of **nodes** and E is a set of *unordered pairs* called **edges**.



- ▶ $G = (V, E)$ where
 - ▶ $V = \{a, b, c, d, e, f\}$;
 - ▶ $E = \{(a, b), (a, d), (a, e), (b, c), (b, d), (b, e), (c, e), (d, e), (e, f)\}$



Remarks

- ▶ An edge is a subset of nodes V with cardinality 2.
 - ▶ Hence the formal way to represent an edge is $\{a, b\}$
 - ▶ By convention however, we often still write it as (a, b) .
- ▶ There is no order for each pair
 - ▶ Thus edge $(a, b) = (b, a)$
- ▶ Simple graphs:
 - ▶ There is no self-loops of the form (a, a)
 - ▶ There is at most one edge for each pair of nodes $\{a, b\}$



Summary

- ▶ Edge direction ?

- ▶ directed graph: Yes
- ▶ undirected graph: No

- ▶ Self-loop?

- ▶ directed graph: Yes
- ▶ undirected graph: No

- ▶ Opposite edges: (a, b) and (b, a) ?

- ▶ directed graph: Yes
- ▶ undirected graph: No (they are the same edge)

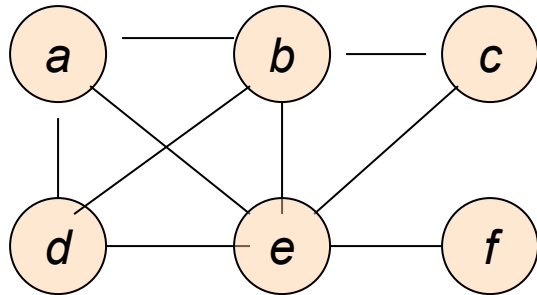


Graphs: More notations



Node degree in undirected graph

- ▶ Given an undirected graph $G = (V, E)$
 - ▶ given an edge $e = (u, v) \in E$, we refer u, v as end-points of e
 - ▶ we say that edge e is **incident on** node u if u is an end-point of e
- ▶ Given an undirected graph $G = (V, E)$, the **degree** of a node $v \in V$ is
 - ▶ $\deg(v) :=$ number of edges incident on v



- ▶ **Example:**
 - ▶ $\deg(e) = 5, \deg(f) = 1$

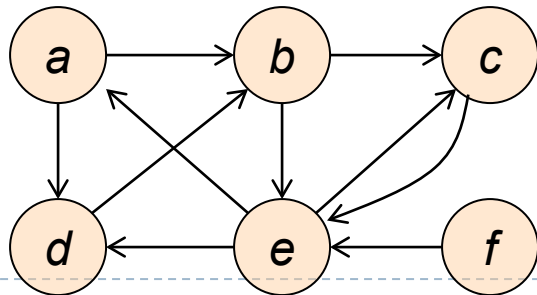
Observations

- ▶ Given a undirected graph $G = (V, E)$ with $n = |V|$
 - ▶ $0 \leq \deg(v) \leq n - 1$, for any node $v \in V$
 - ▶ $\sum_{v \in V} \deg(v) = 2|E|$
- ▶ Given a undirected graph $G = (V, E)$ with $n = |V|$
 - ▶ The maximum number of edges is $\frac{n(n-1)}{2}$
 - ▶ This also implies that $|E| = O(n^2)$
- ▶ A undirected graph $G = (V, E)$ is a **complete graph** iff
 - ▶ There is one edge between every pair of distinct nodes in V
 - ▶ i.e, $|E| = \frac{n(n-1)}{2}$



Node degree in directed graph

- ▶ Given a directed graph $G = (V, E)$, the **in-degree** of a node $v \in V$ is
 - ▶ $\text{indeg}(v) :=$ number of edges **entering** v
- ▶ Given a directed graph $G = (V, E)$, the **out-degree** of a node $v \in V$ is
 - ▶ $\text{outdeg}(v) :=$ number of edges **leaving** v
- ▶ Sometimes we use **degree** to denote the sum
 - ▶ $\text{deg}(v) = \text{indeg}(v) + \text{outdeg}(v)$



- ▶ **Example:**
 - ▶ $\text{indeg}(e) = 3, \text{outdeg}(e) = 3$
 - ▶ $\text{indeg}(f) = 0, \text{outdeg}(f) = 1$

Observations

- ▶ Given a directed graph $G = (V, E)$ with $n = |V|$
 - ▶ $0 \leq \text{indeg}(v), \text{outdeg}(v) \leq n$, for any node $v \in V$
 - ▶ $\sum_{v \in V} \text{indeg}(v) = \sum_{v \in V} \text{outdeg}(v) = |E|$

- ▶ Given a directed graph $G = (V, E)$ with $n = |V|$
 - ▶ The maximum number of edges is n^2
 - ▶ This also implies that $|E| = O(n^2)$



More notations

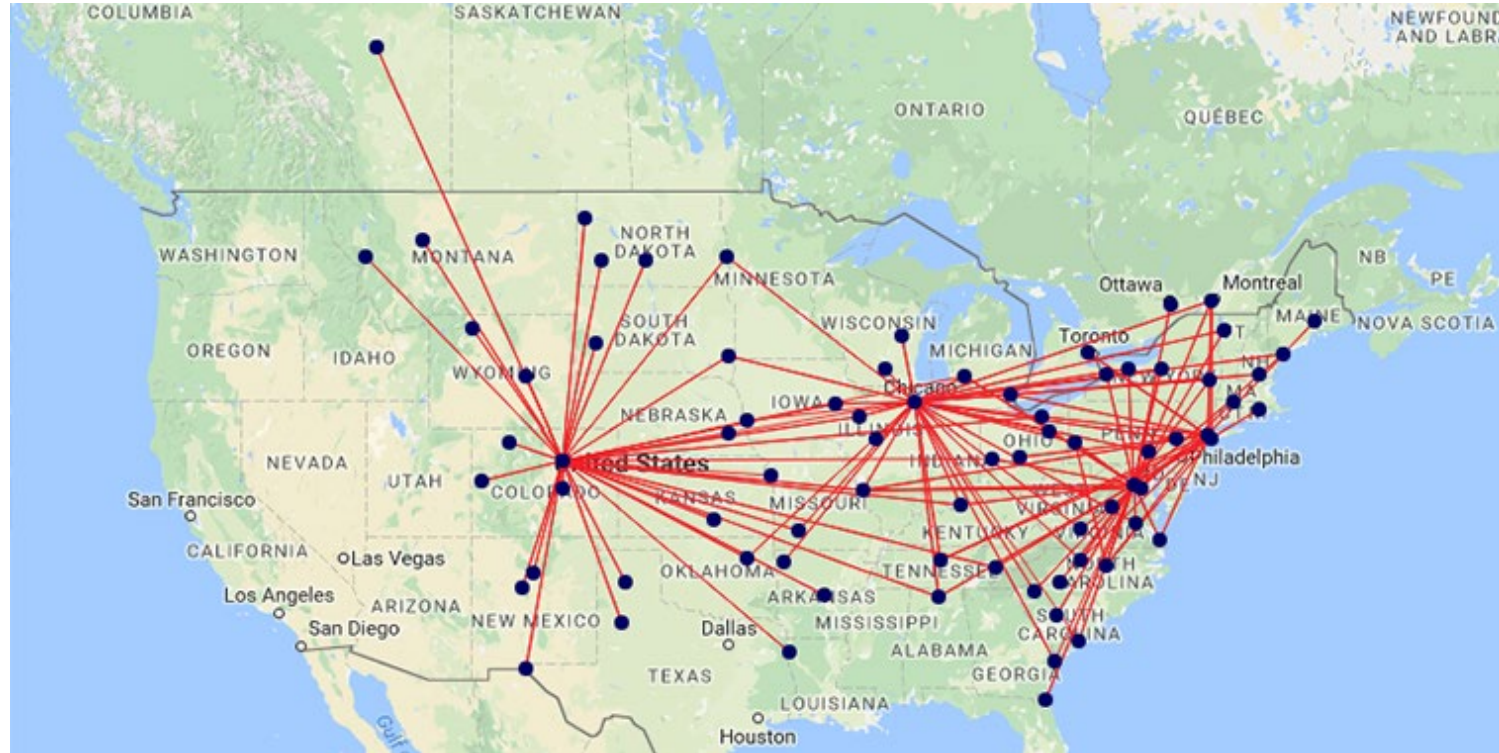
- ▶ Given a undirected graph $G = (V, E)$
 - ▶ the set of **neighbors** of $v \in V$ is the set of all nodes in V that share an edge with v
- ▶ Given a directed graph $G = (V, E)$
 - ▶ the set of **successors** of $v \in V$ is the set of all nodes at the end of an edge leaving v
 - ▶ the set of **predecessors** of $v \in V$ is the set of all nodes at the start of an edge entering v
- ▶ By convention, for a directed graph
 - ▶ the set of **neighbors** of a node often refers to the set of **successors** of this node.



Paths, reachability and connectivity



Example

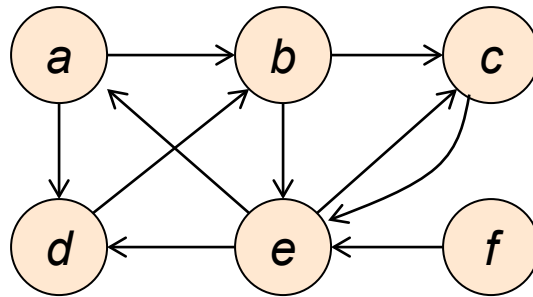


Can we fly from Columbus, Ohio to Denver, Colorado?



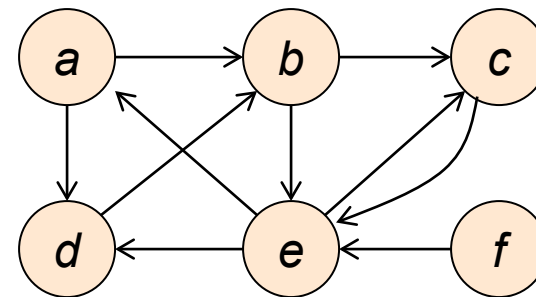
Paths

- ▶ A **path** from u to u' in a (directed or undirected) graph $G = (V, E)$ is a sequence of one or more nodes $u = v_0, v_1, \dots, v_k = u'$ such that there is an edge between each consecutive pair of nodes in the sequence.



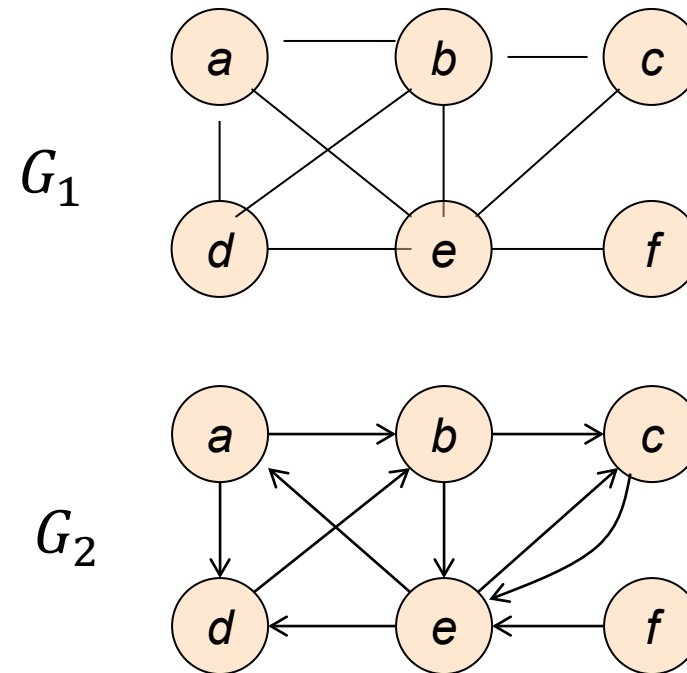
Paths

- ▶ The **length** of a path = # of nodes $- 1$ = # of edges in a path
 - ▶ The length of a path could be **0**
- ▶ A path is **simple** if it visits each node only once
 - ▶ In this class, we usually consider only simple paths.
- ▶ A **cycle** is a path where the first and last nodes are the same



Reachability

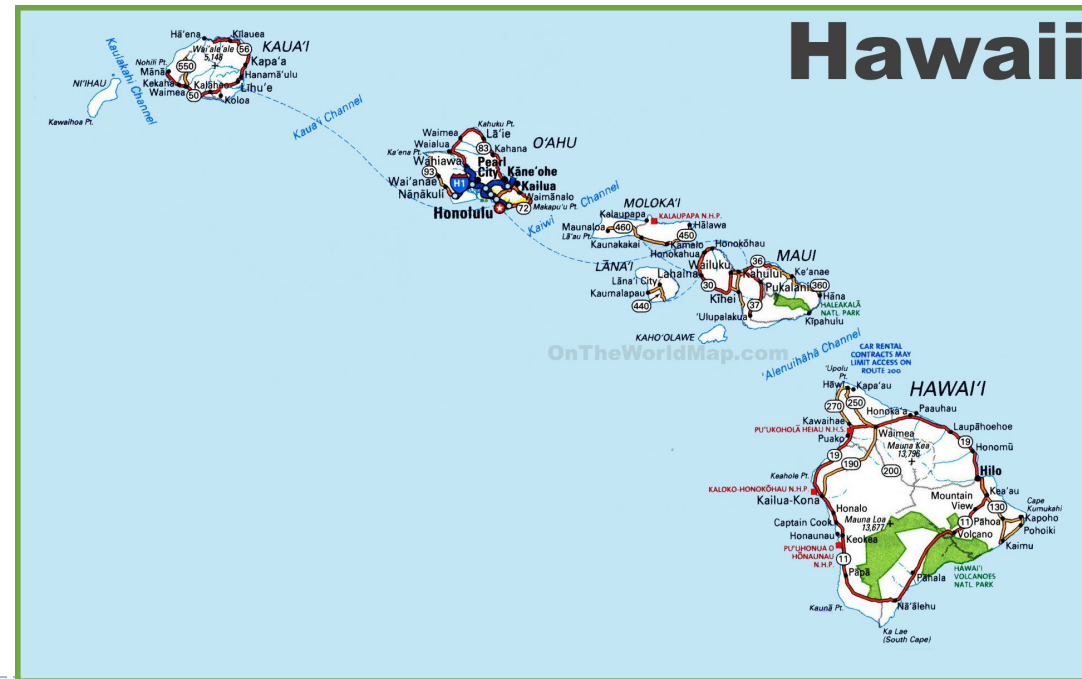
- ▶ Node u is **reachable** from node v if there is a path from v to u
- ▶ In undirected graph,
 - ▶ If u is reachable from v , then v is reachable from u
- ▶ In a directed graph
 - ▶ u is reachable from v does not imply that v is necessarily reachable from u
 - ▶ Reachability is **not symmetric** for directed graphs!



Connectivity

- ▶ This concept is for **undirected graphs**.
- ▶ A undirected graph is **connected** if every node is reachable from every other node. Otherwise, it is **disconnected**.

A graph is disconnected means that there exists at least one pair of nodes u, v such that u is not reachable from v .



Connected components

- ▶ A connected component of a graph $G = (V, E)$ is a maximally-connected subset of nodes of V .
- ▶ That is, given a undirected graph $G = (V, E)$, a connected component is a set $C \subseteq V$ such that
 - ▶ any pairs $u, v \in C$ are reachable from one another; and
 - ▶ if $u \in C$ and $z \notin C$, then u and z are not reachable from one another.
- ▶ If a undirected graph is connected, then it has only one connected component.
- ▶ For directed graphs, there is a concept called **strongly connected components**.



Representations of graphs



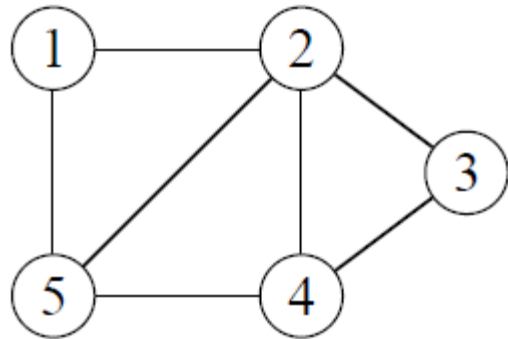
Representation of graphs

- ▶ How do we represent a graph in the computer
 - ▶ e.g, to be used as input to an algorithm
- ▶ Three representations
 - ▶ Adjacency matrix
 - ▶ Adjacency list
 - ▶ Dictionary set



Adjacency matrix representation

- ▶ Assume $V = \{v_0, v_1, \dots, v_{n-1}\}$ with $n = |V|$
- ▶ Adjacency matrix of a graph is a $n \times n$ matrix **adj**
 - ▶ **adj** $[i, j] = \begin{cases} 1, & (v_i, v_j) \in E \\ 0, & \text{otherwise} \end{cases}$



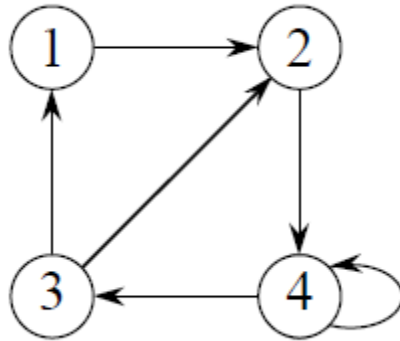
	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0



Adjacency matrix representation

- ▶ Assume $V = \{v_0, v_1, \dots, v_{n-1}\}$ with $n = |V|$
- ▶ Adjacency matrix of a graph is a $n \times n$ matrix **adj**

$$\text{adj}[i, j] = \begin{cases} 1, & (v_i, v_j) \in E \\ 0, & \text{otherwise} \end{cases}$$



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

Observation:

- If the graph is undirected, then this matrix is a symmetric matrix
- If the graph is directed, then the adjacency matrix is not necessarily symmetric

Complexity

- ▶ Size of Adjacency matrix

- ▶ $\Theta(|V|^2)$

- ▶ Time complexity of operations

- ▶ Edge query:

- ▶ is the edge $(v_i, v_j) \in E$?

- ▶ Degree query:

- ▶ what is the **degree** of v_i (in undirected graph), or what is the **outdegree** of v_i (in directed graph)

operation	code	time
edge query	<code>adj[i,j] == 1</code>	$\Theta(1)$
<code>degree(i)</code>	<code>np.sum(adj[i,:])</code>	$\Theta(V)$



Remarks

► Pros:

- Support very efficient edge queries
- Simple and easy to use
 - only need to allocate a $|V| \times |V|$ (Numpy) array
- Easy to manipulate via linear algebra
 - e.g, (i,j)-th entry of A^2 gives number of hops of length 2 between v_i and v_j

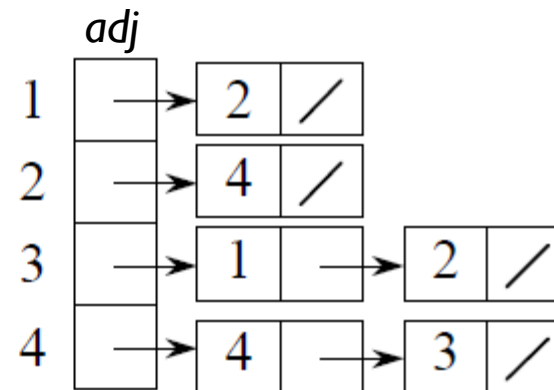
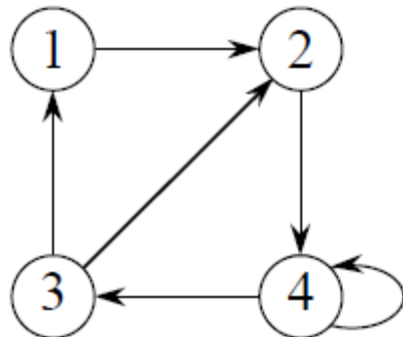
► Cons:

- Take $\Theta(|V|^2)$ no matter what the graph looks like
- In real-life, graphs are often **sparse**, with far fewer number of edges
 - e.g, facebook has 2.7 billion users
 - $|V|^2 = (2.7 \times 10^9)^2 \approx 7.3 \times 10^{18}$ bits \approx 11862 years of video at 1080p
 \approx 109 copies of the internet as it was in 2000



Adjacency list representation

- ▶ Adjacency matrix allocates a bit for each $|V|^2$ potential edge
 - ▶ What if we only store the edges in the graph?
- ▶ **Adjacency lists**
 - ▶ Each vertex u has a list, recording its neighbors
 - ▶ i.e., all v 's such that $(u, v) \in E$
 - ▶ An **array** of $|V|$ lists
 - ▶ $adj[i].size$ = size of AdjList for node v_i
 - ▶ $adj[i]$ = adjacency list for node v_i



Size complexity

- ▶ For each vertex v_i , its adjacency list $\text{adj}[i]$ has size =
 - ▶ $\deg(v_i)$ if the graph is undirected
 - ▶ $\text{outdeg}(v_i)$ if the graph is directed
- ▶ Hence each edge will be stored
 - ▶ **twice** in the adjacency list if the graph is undirected
 - ▶ **once** if the graph is directed
- ▶ Total size:
 - ▶ $\Theta(|V| + |E|)$
 - ▶ where $\Theta(|V|)$ for the outer array, and $\Theta(|E|)$ for the total lengths of inner lists .



Time complexity

▶ Time complexity of operations

- ▶ Accessing the list of neighbors for a specific vertex v_i
 - ▶ takes $\Theta(1)$ time, as just return the list $adj[i]$
- ▶ edge query: is edge $(v_i, v_j) \in E$?
 - ▶ takes $\Theta(\text{length of } adj[i]) = \Theta(\deg(v_i)) = O(|V|)$
 - (deg should be out-deg for directed graphs)
- ▶ degree query: what is the **degree** of v_i (in undirected graph), or what is the **outdegree** of v_i (in directed graph)
 - ▶ takes $\Theta(1)$ time
 - ▶ **However**, for directed graph, checking **indegree** takes $O(|V| + |E|)$ time!



► Summary of time complexity:

- Note: below “degree” refers to the “**degree**” of a node for an undirected graph, but “**out-degree**” of a node for a directed graph!

operation	code	time
edge query	<code>j in adj[i]</code>	$\Theta(\text{degree}(i))$
<code>degree(i)</code>	<code>len(adj[i])</code>	$\Theta(1)$



Remarks

▶ Pros:

- ▶ Optimal space complexity
 - ▶ $\Theta(|V| + |E|)$ is the least possible (asymptotically), thus space complexity is optimal
- ▶ Fast for (out-)degree queries.

▶ Cons:

- ▶ Slow for edge queries
- ▶ No linear algebra operations such as A^2



-
- ▶ **Adjacency matrix:**
 - ▶ Fast edge queries, potentially large space
 - ▶ **Adjacency list:**
 - ▶ Slow edge queries, but efficient in space

Can we take the advantage of both?



Dictionary-set representation

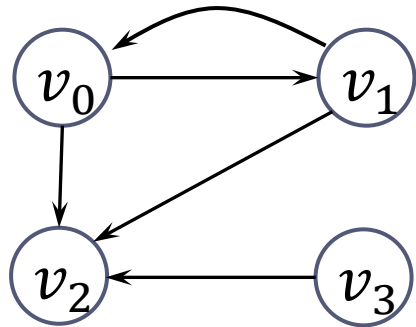
▶ Idea:

- ▶ why not changing the inner list to a **Hash table** to store the list of neighbors (i.e, adjacency list) of each node
 - ▶ E.g, using the **Set** data structure in python.
- ▶ this will make membership query (which is essentially what edge query is about) efficient!
 - ▶ to check whether (v_i, v_j) is in E , we just need to check whether v_j is in the adjacency list of v_i , which is a membership query in the $\text{adj}[i]$
- ▶ We further change the outer array to also a **Hash table**
 - ▶ e. g, using the **dict** data structure in python
 - ▶ this also allows us to map non-integer indexed nodes



Dictionary-of-set implementation

- ▶ Dictionary-of-set implementation
 - ▶ Using an outer hash table (**dict**) to represent all non-empty adjacency list for all nodes
 - ▶ For each node with non-zero neighbors, using a hash table (**set**) to store all its neighbors



Complexity

- ▶ Space complexity

- ▶ $\Theta(|V| + |E|)$

- ▶ Time complexity (in expectation)

operation	code	time
edge query	<code>j in adj[i]</code>	$\Theta(1)$ average
<code>degree(i)</code>	<code>len(adj[i])</code>	$\Theta(1)$ average

- ▶ However,

- ▶ note that time complexity is expected time, and also there will be overhead of using Hash tables.



Dictionary-of-set implementation

- ▶ Install with `pip install dsc40graph`
 - ▶ Or you can also download it and put in the same directory of you code. See Docs below
- ▶ Import with:
 - ▶ `import dsc40graph`
- ▶ Docs:
 - ▶ <https://eldridgejm.github.io/dsc40graph/>
- ▶ Source code:
 - ▶ <https://github.com/eldridgejm/dsc40graph>
- ▶ Will be used in HW coding problems



FIN

