CS512 LECTURE NOTES - LECTURE 9

1 Graph Algorithms

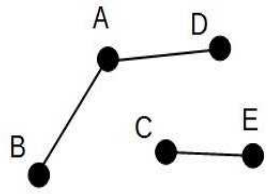
1.1 Graphs

A graph G consists of a set V of vertices and a set E of edges that represents a relation on those vertices: $E \subseteq V \times V$.

$$G = (V, E)$$

Undirected Graph: Each edge is seen as a set of two vertices $e = \{u, v\}, \quad , e \in E, \quad u, v \in V$ where the edge $\{u, v\}$ is equal to the edge v, u. A graphical representation of an undirected graph will represent each vertex as a point and each edge $\{u, v\}$ as a line connecting those points.

Here is an example of an undirected graph:

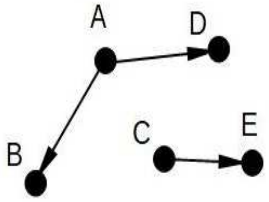


where $G = (V, E)$ and

$$V = \{A, B, C, D, E\} E = \{\{A, B\}, \{A, D\}, \{C, E\}, \}$$

Directed Graph: Each edge is seen as a list of two vertices $e = (u, v), \quad , e \in E, \quad u, v \in V$. A graphical representation of an undirected graph will represent each vertex as a point and each edge (u, v) as an arrow pointing from u to v.

Here is an example of a directed graph:



1.2 Nomenclature

- **Adjacent vertex:** v is adjacent to u if $(u, v) \in E$
- **Neighbors:** u and v are neighbors if $(u, v) \in E$
- **Degree:** Degree of a vertex v is equal to the number of neighbors of v (undirected graph).
- **in-Degree** of vertex v is equal to the number of directed edges pointing to v
- **out-Degree:** number of directed edges pointing away from v
- **Sparse graph:** A graph such that $|E| \in O(|V|)$
- **Dense graph:** A graph such that $|E| \in \Theta(|V|^2)$
- **Complete Graph:** A graph such that if $u, v \in V$, $u \neq v$ then $(u, v) \in E$. A complete graph with i vertices is called a K_i graph.
- **Simple Graph:** A graph that does not contain self loops and does not have multiple edges connecting two nodes. The graphs that we will use during this course are *Simple Graphs*.

1.3 Graph data structures

1.3.1 Adjacency Matrix

An adjacency matrix is a square matrix where the rows and columns represent the vertices, and such that

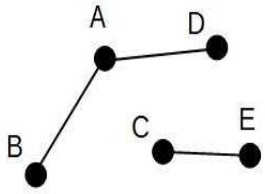
$$M_{i,j} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 0 & \text{if } (v_i, v_j) \notin E \end{cases}$$

Notice that the graphs that we will be studying are simple graphs, so there are no self-loops and no multiple edges connecting the same pair of vertices. Hence, the diagonal of the adjacency matrix should only contain 0's.
Question: How long does it take to determine if a pair of vertices is an edge?

Using an adjacency matrix representation it takes $O(1)$ time since we only have to look for an entry in the matrix

---

EXAMPLE. :
Given the following graph



Its adjacency matrix representation is:

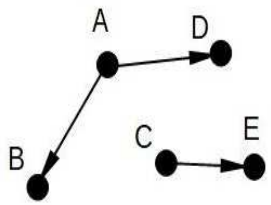
$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Where the rows/columns are ordered in alphabetical order of the vertex labels. This means that row 1 corresponds to vertex A, row 2 to vertex B, and so on.

The adjacency matrix of an undirected graph is symmetric. Notice that each edge appears twice, in $M_{u,v} = 1$ and in $M_{v,u} = 1$

EXAMPLE. :

Given the following graph



Its adjacency matrix representation is:

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

As in the case of undirected graphs the rows/columns are ordered in alphabetical order of the vertex labels. This means that row 1 corresponds to vertex A, row 2 to vertex B, and so on.

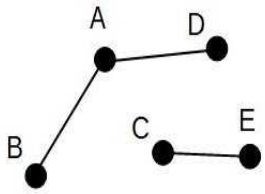
The adjacency matrix of a directed graph is not necessarily symmetric. Notice that each edge appears only once.

1.3.2 Adjacency List

An adjacency list is represented by an array of lists. Each list contains the vertices adjacent to a particular vertex.

EXAMPLE. :

Given the following graph



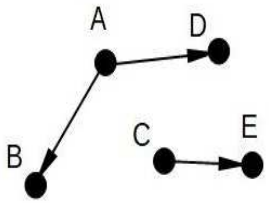
Its adjacency list representation is:

$$\begin{array}{lcl} A & \rightarrow & (B, D) \\ B & \rightarrow & (A) \\ C & \rightarrow & (E) \\ D & \rightarrow & (A) \\ E & \rightarrow & (C) \end{array}$$

In the adjacency list of an undirected graph each edge $\{u, v\}$ appears twice, once on the list of those vertices adjacent to u and once on the list of those vertices adjacent to v.

EXAMPLE. :

Given the following graph



Its adjacency list representation is:

$$\begin{array}{lcl} A & \rightarrow & (B, D) \\ B & \rightarrow & () \\ C & \rightarrow & (E) \\ D & \rightarrow & () \\ E & \rightarrow & () \end{array}$$

In the adjacency list of a directed graph each edge $\{u, v\}$ appears only once.

Question: How long does it take to determine if $(u, v) \in E$?

Using an adjacency list representation it takes $O(n)$ time since we only have to search the list of edges adjacent to u.

2 Exploring a Graph

2.1 Undirected Graphs

Suppose that we want to find (mark) all the vertices of a graph that can be reached from a starting vertex x. The idea is to mark that vertex, and then for each unmarked vertex v adjacent to x recursively visit v.

The algorithm is the following:

```
Explore(x)
    Mark(x)
    for each (x, v) ∈ E
       If not marked(v)
          Explore(v)
```

Path: A path from $v_0 \in V$ to $v_k \in V$ is a sequence of vertices

$$v_0, v_1, v_2, \ldots, v_k$$

where $v_i \in V \quad \forall i = 0 \ldots k$ and $(v_{i-1}, v_i) \in E \quad \forall i = 1 \ldots k$. The length of the path is equal to the number of edges traversed to go from v_0 to v_k. A single vertex represents a path of length 0

Connected Graph: An undirected graph is said to be connected iff $\forall u, v \in V$ there is a path from u to v.

The *Explore* algorithm given above can be used to determine if a graph G is connected or not. Notice that if a graph is connected then all vertices of the graph can be reached (there is a path) from any other vertex. So we have the following algorithm:

```
Connected(G)

  let v ∈ V
  Explore(v)
  for each u ∈ V
     if not marked(u) return "Not Connected"
  return "Connected"
```