Overview

• HW#6 due now & Pick up HW#7 handout
• HW7 is due in one and half week on 11/16.
• MT3 scheduled right after the break on 11/28
• Last Lecture
  – Graph Theory: Definitions using Sets, Handshaking theorem
  – Graph types: Complete Graph, n-Cube, Bipartite graph, Complete bipartite graph
• This Lecture: More on graph theory
  – Operations: Subgraph, Union
  – Adjacency matrix, Adjacency list
  – Isomorphism (equivalency of graphs)
  – Connectivity, connected graphs
  – Shortest path problem,
  – Dijkstra’s algorithm
  – Travel Salesman Problem

Operations on Graphs

• Definition: A subgraph of a graph \( G = (V, E) \) is a graph \( H = (W, F) \) where \( W \subseteq V \) and \( F \subseteq E \).
• Note: \( H \) must be a valid graph. i.e. For each \( \{u,v\} \) in \( F \), u and v are vertices in \( W \)
• Example:

![Graphs and Subgraphs](image)
• **Definition:** The union of two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is the simple graph with vertex set $V_1 \cup V_2$ and edge set $E_1 \cup E_2$.

The union of $G_1$ and $G_2$ is denoted by $G_1 \cup G_2$.

![Graphs](image)

Representing Graphs numerically.

• **Definition:** Let $G = (V, E)$ be a graph with $|V| = n$. Suppose that the vertices of $G$ are listed in arbitrary order as $v_1, v_2, \ldots, v_n$.

The adjacency matrix $A_G$ of $G$, with respect to this listing of the vertices, is the $n \times n$ zero-one-valued matrix with 1 as its $(i, j)$-th entry when $v_i$ and $v_j$ are adjacent, and 0 otherwise.

In other words, for an adjacency matrix $A = [a_{ij}]$, $a_{ij} = 1$ if $\{v_i, v_j\}$ is an edge of $G$, $a_{ij} = 0$ otherwise.

• **Example:** What is the adjacency matrix $A_G$ for the following graph $G$ based on the order of vertices $a, b, c, d$?

**Solution:**

\[
A_G = \begin{bmatrix}
0 & 1 & 1 & 1 \\
1 & 0 & 0 & 1 \\
1 & 0 & 0 & 1 \\
1 & 1 & 1 & 0
\end{bmatrix}
\]
Definition: **Adjacency list** representation consists of a *list* of its vertices together with a separate *list* for each vertex that contains all the vertices adjacent to that vertex.

Example: let 1 = a, 2 = b, 3 = c and 4 = d

<table>
<thead>
<tr>
<th>Vertex</th>
<th>List of adjacencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a, 2, 3, 4</td>
</tr>
<tr>
<td>2</td>
<td>b, 1, 4</td>
</tr>
<tr>
<td>3</td>
<td>a, 1, 4</td>
</tr>
<tr>
<td>4</td>
<td>1, 2, 3</td>
</tr>
</tbody>
</table>

Note: Adjacency Matrix and Adjacency Lists can be used to defined directed graphs also.

---

Isomorphism of Graphs

- **Definition:** The graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are **isomorphic** if there is a **bijection** (an one-to-one and onto function) $f$ from $V_1$ to $V_2$ with the property that $a$ and $b$ are adjacent in $G_1$ if and only if $f(a)$ and $f(b)$ are adjacent in $G_2$, for all $a$ and $b$ in $V_1$. Such a function $f$ is called an **isomorphism**.

- In other words, $G_1$ and $G_2$ are isomorphic if their vertices can be **ordered** in such a way that the adjacency matrices $M_{G_1}$ and $M_{G_2}$ are identical.

- From a visual standpoint, $G_1$ and $G_2$ are isomorphic if they can be rearranged in such a way that their **displays are identical**.
• Unfortunately, for two graphs, each with \( n \) vertices, there are \( n! \) possible isomorphisms that we have to check in order to show that these graphs are isomorphic.

• One way to show that two graphs are not isomorphic is to check invariants, that is, properties that two isomorphic simple graphs must both have. For example, they must have

  - the same number of vertices,
  - the same number of edges, and
  - the same degrees of vertices (there are more....)

Note that two graphs that differ in any of these invariants must be “not isomorphic” (SO CHECK THIS FIRST ALWAYS!), but two graphs that match in all of them are not necessarily isomorphic.

---

**Example I:** Are the following two graphs isomorphic?

\[
\begin{array}{c}
\text{a} & \text{b} & \text{c} & \text{d} & \text{e} \\
\text{f(a)} & \text{f(b)} & \text{f(c)} & \text{f(d)} & \text{f(e)}
\end{array}
\]

**Solution:** Yes, they are isomorphic, because they can be arranged to look identical, i.e. \( f(a) = e, f(b) = a, f(c) = b, f(d) = c, f(e) = d \).

**Example II:** How about these two graphs?

**Solution:** No, they are not isomorphic, because they differ in the degrees of their vertices. Vertex d in right graph is of degree one, but there is no such vertex in the left graph.
Connectivity

- **Definition:** A path of length \( n \) from \( u \) to \( v \), where \( n \) is a positive integer, in an undirected graph is a sequence of edges \( e_1, e_2, ..., e_n \) of the graph such that \( e_1 = \{x_0, x_1\}, e_2 = \{x_1, x_2\}, ..., e_n = \{x_{n-1}, x_n\} \), where \( x_0 = u \) and \( x_n = v \). We can also denote this path by its vertex sequence \( x_0, x_1, ..., x_n \).

- The path is a **Circuit** if it begins and ends at the same vertex, that is, if \( u = v \). The path or circuit is said to **pass through** or **traverse** \( x_1, x_2, ..., x_{n-1} \).

- A path or circuit is **simple** if it does not contain the same edge more than once. A **simple circuit** is also called **Cycle**.

- A cycle that contains all the vertices of a graph is called a **Hamiltonian (Hamilton) Cycle**.

- A circuit that includes each edge of a graph exactly once is called an **Eulerian Circuit**.

- **Theorem:** Let \( G \) be a (connected) graph. \( G \) has a **Eulerian circuit** if and only if **each vertex is even degree**

- Example: The Königsberg Bridges graph.

- Example: 1-2-3-1-4-5-1-6-7-8-6-2-9-1

This graph is not Eulerian, therefore, a solution does not exist.
Connectivity

• **Definition:** An undirected graph is called **connected** if there is a path between every pair of distinct vertices in the graph.

• For example, any two computers in a network can communicate if and only if the graph of this network is connected.

• **Note:** A graph consisting of only one vertex is always connected, because it does not contain any pair of distinct vertices.

• **Examples:**

  - Yes.
  - No.

• **Definition:** A graph that is not connected is the union of two or more connected subgraphs, each pair of which has no vertex in common. These disjoint connected subgraphs are called the **connected components** of the graph.

• **Definition:** A **connected component** of a graph $G$ is a maximal connected subgraph of $G$.

• **Example:**

  - Yes.
  - No.

  The connected components are the graphs with vertices \{a, b, c, d\}, \{e\}, \{f\}, \{i, g, h, j\}.


• **Definition**: An directed graph is **strongly connected** if there is a path from \( a \) to \( b \) and from \( b \) to \( a \) whenever \( a \) and \( b \) are vertices in the graph.

• **Definition**: An directed graph is **weakly connected** if there is a path between any two vertices in the underlying undirected graph.

![Diagram](image)

**Weakly connected**, because, for example, there is no path from \( b \) to \( d \). But underlying undirected graph is connected

**Strongly connected**, because there are paths between all possible pairs of vertices.

---

A **planar graph** is one that can be represented (on a sheet of paper, that is, in the plane) so that its edges intersect only at endpoints.

![Diagram](image)

Right graph = Left graph. It is clearly a planar graph.

**K\(_4\)**

**K\(_5\)**

what about **K\(_4\)** and **K\(_5\)**?
Application: Shortest Path Problems \( G=(V,E,W) \)

- We can assign **weights** to the edges of graphs, for example to represent the distance between cities in a railway network:

![Graph with cities](image)

- One of the most interesting questions that we can investigate with such graphs is:

  What is the **shortest path** between two vertices in the graph, that is, the path with the **minimal sum of weights** along the way?

  This corresponds to the shortest train connection or the fastest connection in a computer network (edge weight = response time).

---

Dijkstra’s Algorithm

- Dijkstra’s algorithm is an iterative procedure that finds the shortest path between two vertices (e.g., \( a \) to \( z \)) in a weighted graph.

- It proceeds by finding the length of the shortest path from \( a \) to successive vertices.

- The algorithm terminates once it reaches the vertex \( z \).

  **The final shortest path can be back-tracked from \( z \).**

**Example:**

![Graph with edges](image)

**Answer:** \( a, c, b, d, e, z \)  keep min cost value on each vertex
1 function Dijkstra(Graph, source(a), target(z)):

2 for each vertex v in Graph: // Initializations
3   dist[v] := infinity     // Unknown distance function from source to v
4   prev[v] := undefined   // Previous node in optimal path from source
5   dist[source] := 0       // Distance from source to source
6   Q := the set of all nodes in Graph
7 while Q is not empty:     // The main loop
8     u := node in Q with smallest dist[]
9     remove u from Q
10    if u=target, exit, done!
11   for each neighbor v of u and v still in Q.
12       alt := dist[u] + dist_between(u, v)
13       if alt < dist[v]                // Relax (u,v)
14           dist[v] := alt            // update v’s data
15           prev[v] := u           // update v’s previous node to u

\[
\begin{align*}
\text{dist}[]_{v=a, \ldots z} &= \infty \\
\text{prev}[]_{v=a, \ldots z} &\leftarrow \text{undefined} \\
Q &= \{a, b, c, d, e, z\}
\end{align*}
\]
Application: The Traveling Salesman Problem

- The **traveling salesman problem** is one of the classical problems in computer science.

A traveling salesman wants to visit all major cities and then return to his starting point. Of course he wants to save time and energy, so he wants to determine the **shortest path** for his trip.

We can represent the cities and the distances between them by a **weighted, complete, undirected graph**.

The problem then is to find a **Hamiltonian cycle of minimum total weight** that visits each vertex exactly once.
• **Example:** What path would the traveling salesman take to visit the following cities?

![Diagram of cities with distances](image)

**Solution:** The shortest path is Boston, New York, Chicago, Toronto, Boston (2,000 miles).

• **Question:** Given \( n \) vertices, how many different cycles (with all \( n \) vertices) can we form by connecting these vertices with edges?

**Solution:** We first choose a starting point. Then we have \((n - 1)\) choices for the second vertex in the cycle, \((n - 2)\) for the third one, and so on, so there are \((n - 1)!\) choices for the whole cycle.

However, this number includes identical cycles that were constructed in opposite directions. Therefore, the actual number of different cycles is \((n - 1)!/2\).

• Unfortunately, no algorithm solving the traveling salesman problem with polynomial worst-case time complexity has been devised yet.

• This means that for large numbers of vertices, solving the traveling salesman problem is **not tractable (impractical)**.

• In these cases, we can use efficient **approximation algorithms** that determine a path whose length may be slightly larger than the traveling salesman’s path.