[TAP: CWJXL] Balanced Trees

• Which of the following are not guaranteed to be “balanced”?
  A. AVL Tree
  B. Red-black Tree
  C. Splay Tree
  D. They are all balanced
  E. Whatever
Today’s Outline

- Graphs
  - Undirected Graph
  - Directed Graph
  - Implementation
Graphs Describe the World

- map of cities
  - cities, roads
  - cities, rivers

- Social network
  - users, following (twitter)
  - users, friend (facebook)
Nodes = subway stops; Edges = track between stops
Nodes = cities; Edges = rail lines connecting cities
Connections in graph matter, not precise locations of nodes
Internet (~1972)
Facebook social network graph
Wire-Frame Models
Today’s Outline

• Graphs
  • Undirected Graph
  • Directed Graph
  • Implementation
An undirected graph is denoted as $G = (V, E)$, where

- $V$: set of vertices
- $E$: set of edges, and each edge is an unordered pair of vertices (we write $e = \{u, v\}$)
Walking Along a Graph

• A walk from $u$ to $v$ in a graph $G = (V, E)$ is an alternating sequence of vertices and edges (often, we just write the vertices)

$$u = v_0, e_1, v_1, e_2, v_2, \ldots, v_{k-1}, e_k, v_k = v$$

such that each $e_i = \{v_i, v_{i+1}\}$ for $i = 1, \ldots, k$

• A path (trail) is a walk where no edge appears more than once

• A simple path (path) is a walk where no vertex appears more than once
Reachability and Connectedness

- A vertex \( v \) in \( G \) is **reachable** from a vertex \( u \) in \( G \) if there is a path from \( u \) to \( v \)
  - Note, \( v \) is reachable from \( u \) *iff* \( u \) is reachable from \( v \)
- An undirected graph \( G \) is **connected** if for every pair of vertices \( u, v \) in \( G \), \( v \) is reachable from \( u \) (and vice versa)
- The set of all vertices reachable from \( v \), along with all edges of \( G \) connecting any two of them, is called the **connected component of** \( v \)

\[ G \] is not connected but has 2 CC.
Little Tiny Theorems

- If there is a walk from u to v, then
  - there is a walk from v to u.
  - there is a path from u to v (and from v to u)
- If there is a path from u to v, then
  - there is a simple path from u to v (and v to u)
Let $\deg(v)$ be the degree of a vertex $v$. Is the following statement true?

For any graph $G = (V,E)$

$$\sum_{v \in V} \deg(v) = 2 |E|$$

where $|E|$ is the number of edges in $G$
Today’s Outline

• Graphs
  • Undirected Graph
  • Directed Graph
  • Implementation
An *undirected graph* is denoted as $G = (V, E)$, where

- $V$: set of vertices
- $E$: set of edges, and each edge is an unordered pair of vertices (we write $e = \{u, v\}$)
Degrees

Out-degree: Σ1 of outgoing edges
In-degree: Σ4 of incoming edges
The concept of a walk and path is still the same, but you can only walk along the direction of the edges.
Reachability and Connectedness

- A vertex \( v \) in \( G \) is \textit{reachable} from a vertex \( u \) in \( G \) if there is a path from \( u \) to \( v \).
- Note, \( v \) is reachable from \( u \) \textit{iiff} \( u \) is reachable from \( v \).
- An undirected graph \( G \) is \textit{connected} if for every pair of vertices \( u, v \) in \( G \), \( v \) is reachable from \( u \) (and vice versa).
- The set of all vertices reachable from \( v \), along with all edges of \( G \) connecting any two of them, is called the \textit{connected component of} \( v \).
Today’s Outline

• Graphs
  • Undirected Graph
  • Directed Graph
• Implementation
Implementing Graphs

- Involves a number of implementation decisions, depending on intended uses
  - What kinds of graphs will be supported?
    - Undirected, directed, mixed
  - What underlying data structures will be used?
  - What functionality will be provided
  - What aspects will be public/protected/private
Graphs in structure

- Interface `Graph<V,E>` extends `Structure<V>`
  - Type `V` holds a `label` for a vertex
  - Type `E` holds a `label` for an edge

Example: city name, distance
Desired Functionality

• What are the basic operations we need to describe algorithms on graphs?
  • Given vertices u and v: are they adjacent?
  • Given vertex v and edge e, are they incident?
  • Given an edge e, get its incident vertices (ends)
  • How many vertices are adjacent to v? (degree of v)
    • The vertices adjacent to v are called its neighbors
  • Get a list of the neighbors of v (or the edges incident with v)
Representing Graphs

• Two standard approaches
  • Option 1: Array-based (directed and undirected)
  • Option 2: List-based (directed and undirected)
Entry (i,j) stores 1 if there is an edge from i to j; 0 otherwise.

E.G.: edges(B,C) = 1 but edges(C,B) = 0
## Adjacency Array: Undirected Graph

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td>1</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>H</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Entry $(i,j)$ store 1 if there is an edge between $i$ and $j$; else 0
E.G.: $\text{edges}(B,C) = 1 = \text{edges}(C,B)$
## Adjacency Array: Undirected Graph

Halving the Space (not in structure 5)

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

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

(i, j) maps to i*7 + j

0 1 2 3 4 5 6 7 8 9 ...
Adjacency List: Directed Graph

The vertices are stored in an array V[].
V[] contains a linked list of edges having a given source.
**Adjacency List : Undirected Graph**

The vertices are stored in an array $V[]$.

$V[]$ contains a linked list of edges incident to a given vertex.