Hash Functions cont.

Last time: proved that expected search time for a uniform hash $h$ was $\Theta(1 + \alpha)$ where $\alpha = \frac{\#\text{ of elem.}}{\text{size of table}}$.

Division Method

$\text{hash fn: } h : U \to \{0, 1, \ldots, m-1\}$

$h(k) = k \mod m$

key

pos. integer

table size
m should not be a power of 2.

pick m away from powers of 2. (primes can be good)

**Multiplication Method**

Choose $0 < A < 1$.

$$h(k) = \sum_{m} (kA \mod 1)$$

$$= kA - \lfloor kA \rfloor$$

advantage: $m$ can be anything.
- can be implemented
just using 'bit' operations

\[ m = 2^5 \]

\[ T \times A \]

\[ f = h(k) \]

(remember for CS 350!)
Graphs

A graph $G = (V, E)$ consists of a finite set of vertices $V$ and a finite set of edges $E$.

Edges can be either directed or undirected.

$$E \subseteq V \times V = \{(u, v) \mid u, v \in V\}$$

"cross product" $A \times B = \{(x, y) \mid x \in A, y \in B\}$

Note: does not rule out "self-edges" $(v, v)$.
Directed

\[ u \rightarrow v \]

\((u, v) \in E\]

"an edge from u to v"

Undirected

\[ u \quad \longrightarrow \quad v \]

\(E\) is symmetric

\((u, v) \in E \Rightarrow (v, u) \in E\)

Weighted Graphs

A graph is weighted if there is a wt. function
w : E → R.

Example

```
12  → v3  → -6
  ↑   ↓   ↓   ↓
  2   v4   o v5
```

"weighted directed disconnected " graph

**Representations of Graphs**

1. adjacency matrix
2. adjacency lists
Example

adj. matrix $M$

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

$M$ is a $n \times n$ matrix where $n = |V|$. 
### Adjacency Matrix

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

*Note: V4 is marked as a sink.*

- Mainly 0's
- "Sparse" matrix

### Adjacency List

- $V_1 \rightarrow V_2 \rightarrow V_4$
- $V_2 \rightarrow V_3$
- $V_3 \rightarrow V_2 \rightarrow V_4$
- $V_4 \rightarrow \text{null}$
- $V_5 \rightarrow V_2$
- $V_6 \rightarrow V_5$

*Use when graph is sparse*
Observe that if $M$ is symmetric (G is undirected) then we only need to store the upper diagonal of $M$.

Graph Algorithms

- Search it!

- BFS / DFS
  - breadth first search
  - depth first search
Colored Scheme for Vertices

Colors

- white: undiscovered vertices
- grey: visited vertex but can still find new nodes
- black: finished vertex, visited and visited neighbors
Breadth-First Search

- uses a queue data structure

\textbf{BFS} \left( G, s \right)

\textit{Source vertex}

\textbf{Initialization}

for \( u \in V - \left\{ s \right\} \)

\textbf{color} \( [u] = \text{white} \);

\textbf{d} \( [u] = \infty \); \( j \leftarrow \text{shortest path distance} \);

\textbf{π} \( [u] = \text{nil} \); \( i \leftarrow \text{parent of } u \);

\textbf{color} \( [s] = \text{gray} \);

\textbf{d} \( [s] = 0 \);

\textbf{π} \( [s] = \text{nil} \);

\textbf{Q} = \emptyset ; j

\textbf{ENQUEUE} \left( Q, s \right) ;

\textbf{while} \ Q \neq \emptyset

\textbf{u} = \text{DEQUEUE} \left( Q \right) ;

\textbf{for} \ v \ in \ \text{adjacent} \ (u) \n
\textbf{if} \ \text{color} \ [v] = \text{white} \n
\textbf{then} \ \text{color} \ [v] = \text{gray} \;

\textbf{d} \ [v] = \text{d} \ [u] + 1 ;

\textbf{π} \ [v] = u ;

\textbf{ENQUEUE} \left( Q, v \right) ;

\textbf{color} \ [u] = \text{BLACK} ;