CS2100 - Intro to Graphs

Announcements

- Lab tomorrow (already up)
- HW due Wednesday
Decode Huffman

Binary Tree -> myTree

BitStream myFile

myFile.open("banana.my.zip.txt");

int input;

input = myFile.read(1);

if (input == 0) {
  myTree.createRoot();
}

read more bits

// loop to create the tree

while ()

  (move position up & down to

  insert new values)
Graphs

A graph $G = (V, E)$ is a set of two sets $V \cup E$.

$V = \text{vertices} \quad V = \{v_1, v_2, v_3, v_4\}$

$E = \text{edges} \quad (\text{which are pairs of vertices})$

$E = \{e_1, e_2, \ldots \}$

\begin{itemize}
  \item $e_1 : \{v_1, v_2\}$
  \item $e_2 : \{v_3, v_4\}$
  \item $\ldots$
\end{itemize}
Why use graphs?
They can model anything!
Examples:
- maps
- networks (Facebook, pagerank...)
- scheduling
Definitions

- $G$ is undirected if every edge is an unordered pair, so $\overline{e} = \{v, u\}$

- $G$ is directed if every edge is an ordered pair

$e = (u, v) \neq (v, u)$
DMS

- The degree of a vertex, $d(v)$, is the number of adjacent edges.

- A path $P = v_1 \ldots v_k$ is a set of vertices with $\forall i, j, v_i, v_j \in E$.

- A path is simple if all vertices are distinct.

- A path is a cycle if it is simple except $v_i = v_k$. 

$d(v) = 3$
Lemmas: (degree-sum formula)

\[ \sum_{v \in V} d(v) = 2|E| \]

Why? Counting: think of edges in \( G \)

every edge connects 2 vertices

Count via the vertices:

\[ d(v_1) + d(v_2) + d(v_3) + \ldots \]

Either way, counting a vertex-edge incidences
Sizes of $|V| = |E|$

We usually let $n = |V|$ and $m = |E|$. How big can $m$ be?

How many edges can a graph have?

\[
\binom{n}{2} = \frac{n(n-1)}{2} = \sum_{i=1}^{n} i = \sum_{i=1}^{\frac{n(n-1)}{2}} e + n
\]

\[
m = O(n^2)
\]
Tree: A connected graph with no cycles.
(Note: No root in this definition!)

How many edges?
$n - 1$
Graphs on a computer

How can we construct this data structure?

Linked structure

struct Node {
    Vertex value
    vector or list of nbs
}

3
Vertex List $B$ (or Vectors)

$V_1 : V_2, V_5$

$V_2 : V_1, V_5, V_3$

$V_3 : V_2, V_4, V_5$

$V_4 : V_3, V_5$

$V_5 : V_1, V_2, V_3, V_4$

Size: $\sum_{v \in V} d(v) = O(m)$

Check if $v_i$ is neighbor of $v_j$: $O(n)$
Implementation

We call these vertex lists, but don't actually need lists.

Options: vectors, lists, BSTs

Trade-offs: insert vs remove vs find
Value list: \(v_1\) is "Trin"

\[v_2\]

\[\text{Adjacency Matrix} \]

\[
\begin{array}{cccc}
\text{v}_1 & \text{v}_2 & \text{v}_3 & \text{v}_4 & \text{v}_5 \\
\hline
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0
\end{array}
\]

Weight

Only need this half if G is directed

Space: \(O(n^2)\)

Check neighbor: \(O(1)\)
Which is best?

Just depends.
Incidence Matrix

e_1 \cdots e_m

V_1 \ V_2 \ V_3 \ V_4 \ \cdots \ \ V_n
Dfs

- $G$ is connected if for all $u$ and $v$, there is a path from $u$ to $v$.
- The distance from $u$ to $v$, $d(u, v)$, is equal to the length of the minimum $u, v$-path.
Algorithms on Graphs

Basic Question: Given 2 vertices, are they connected?

How to solve?
Suggestion:

- Suppose we're in a maze, searching for a treasure.

What do you do?
Recursive DFS (u):

If u is unmarked:
  mark u
  for each edge \( uv, v \in E \) 
  Recursive DFS (v)

To check if s and t are connected,
Call DFS (s).

At end, if t is marked, return true.
DFS

Diagram of a graph with nodes labeled 1, 2, 3, 4, 5, 6, 7, and 8.
Another version of DFS

Iterative DFS($u$):
create empty stack $S$
$S$. push($u$)
while $S$ is not empty:
    $v$ ← $S$. pop
    if $v$ is not marked
        mark ($v$)
        for each edge $v$ to $w$
            $S$. push($w$)