CS180 - Graphs

Announcements

- 2 weeks left
  Final is Dec 17th at noon
  (Review last of class or Friday before)

- Check point due tomorrow

- Lab on Thursday

- Instructor evals this week
A graph \( G = (V, E) \) is a set of 2 sets \( V + E \).

- \( V = \text{vertices} \) \( V = \{v_1, v_2, v_3, v_4\} \)
- \( E = \text{edges} \) (which are pairs of vertices)

\( E = \{\{v_1, v_2\}, \{v_2, v_3\}, \ldots\} \)
Why use graphs?

They can model anything!

Examples:
- Airport terminals
- Road networks
- Computer networks
- Social networks
Definitions

- G is undirected if every edge is an unordered pair so \( \{v, u\} = \{u, v\} \).

- G is directed if every edge is an ordered pair

  \[ e = (u, v) \neq (v, u) \]

  ![Diagram of a directed graph with labeled edges]
Dfs

- 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 \( \forall i \in [1, k-1], v_i \neq v_{i+1} \) \( \forall e \in E \).

- A path is simple if all vertices are distinct.

- A path is a cycle if it is simple except \( v_1 = v_k \).

Degree 3

Cycle
Lemma: (Degree-sum formula)

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

Why?

LHS: counting all edges connected to each vertex.

Every edge is connected to 2 vertices.

3 + 3 + 1 + 1 + 2 = 10
Sizes of $|V| + |E|$

We usually let $n = |V|$ and $m = |E|$. How big can $m$ be? $m \leq \frac{n(n-1)}{2}$, $m = O(n^2)$

$\{1, 2, \ldots, n\}$

\[
\binom{n}{2} = \frac{n!}{(n-2)! \cdot 2!} = \frac{n(n-1)}{2}
\]

\[
(n-1) + (n-2) + (n-3) + \ldots + 1 = \frac{n(n-1)}{2}
\]
Tree: \( n \) vertices in a tree
\[
m = n - 1
\]

Sparse graphs have \( m \geq n \)
Graphs on a computer

How can we construct this data structure?

Node - with pointers to other nodes

- city
  - list of pointers (or vector)
Vertex List (or vectors) \( n \) vertices \( m \) edges

\( v_1 : 2, 5 \)
\( v_2 : 1, 3, 5 \)
\( v_3 : 2, 4, 5 \)

\( v_4 : \)
\( v_5 : \)

Size: \( 2m \) (in list) + \( n = O(m+n) \)

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

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

Can store in any auxiliary data structure.

Trade-offs: usual
- insert/delete
- keep sorted & have binary search
Adjacency Matrix

\[ A \]

\[
\begin{array}{ccccc}
  & v_1 & v_2 & v_3 & v_4 & v_5 \\
v_1 & 0 & 1 & 0 & 0 & 0 \\
v_2 & 1 & 0 & 1 & 0 & 0 \\
v_3 & 0 & 1 & 0 & 0 & 0 \\
v_4 & 0 & 0 & 0 & 1 & 0 \\
v_5 & 1 & 0 & 0 & 0 & 0 \\
\end{array}
\]

space: \( O(n^2) \) \( \approx \) always worse

check neighbor: \( A[i][j] = 1 \Rightarrow O(1) \)
Which is best?

Just depends.

Sparse graphs - use lists
Definition:

- $G$ is connected if for all $u, v$, 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.

\[ d(u, v) = 2 \]
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 (u, v) ∈ E
    RecursiveDFS(v)

(depth-first search)

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

At end, if t is marked, return true
DFS "tree":

DFS(1):

1
2
4
6
5
3
8
Another version of DFS

Iterative DFS(\(v\)):
create empty stack \(S\)
\(S.\ push(\(v\))\)

while \(S\) is not empty:
\(v \leftarrow S.\ pop\)
if \(v\) is not marked
mark (\(v\))
for each edge \(vw\)
\(S.\ push(w)\)
Iterative DFS (1):

Stack: