Graph Isomorphism

When are two graphs that may look different when they're drawn, really the same?

Answer: $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ are *isomorphic* if they have the same number of vertices ($|V_1| = |V_2|$) and we can relabel the vertices in G_2 so that the edge sets are identical.

- Formally, G_1 is isomorphic to G_2 if there is a bijection $f : V_1 \rightarrow V_2$ such that $\{v, v'\} \in E_1$ iff $(\{f(v), f(v')\} \in E_2$.
- Note this means that $|E_1| = |E_2|$

In general, it's very hard to tell if two graphs are isomorphic.

Reachability

Is there a path in graph G from vertex v to v'?

- if the vertices in a graph correspond to towns, and v and v' are connected by an edge if there's a direct road link from v to v', then v is reachable from v' if there's a way of driving from v to v'
- in a communication network, reachability describes who can (ultimately) communicate with whom.

How can we test if one vertex is reachable from another?

A Useful Representation of a Graph

How can we represent a graph in a computer.

- You can't draw pictures ...

One obvious way: list the vertices and edges

Anoter way: represent a graph $G(V, E)$ by its *adjacency matrix.*

If $V = (v_1, \ldots, v_n)$, then the adjacency matrix is an $n \times n$ matrix.

- $A = (a_{ij})$, where $a_{ij} = 1$ if there is an edge from v_i to v_j; otherwise $a_{ij} = 0$.
- in a multigraph, a_{ij} is the number of edges from i to j.

Example:

$$\begin{bmatrix} 0 & 0 & 2 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Note:

- an undirected graph will have a symmetric adjacency matrix: $a_{ij} = a_{ji}$.
- the indegree of v_i = sum of entries in column i
- the outdegree of v_i = sum of entries in row i
- the adjacency matrix is a good way of representing a graph in a computer

Adjacency Matrices and Reachability

What does the adjacency matrix have do with reachability?

Theorem: Suppose A is the adjacency matrix of G and $A^m = (a_{ij}^{(m)})$. Then $a_{ij}^{(m)}$ is the number of paths of length m from v_i to v_j.

Proof: By induction on m. Let $P(m)$ be the statement of the theorem. $P(1)$ is immediate from the definition of the adjacency matrix. Assume $P(m)$. Suppose $A^{m+1} = (a_{ij}^{(m+1)})$. By definition,

$$a_{ij}^{(m+1)} = \sum_{k=1}^{n} a_{ik}^{(m)} a_{kj}$$

- $a_{ik}^{(m)} = \#$ paths of length m from v_i to v_k
- $a_{kj} = \#$ edges (paths of length 1) from v_k to v_j
- Therefore $a_{ik}^{(m)} a_{kj} = \#$ paths from v_i to v_j of length $m+1$ whose second-last vertex (just before v_j) is v_k
- Therefore $a_{ij}^{(m+1)} = \Sigma_{k=1}^{n} a_{ik}^{(m)} a_{kj}$ is the total number of paths of length $m+1$ from v_i to v_j

- v_j is reachable from v_i iff there is a path of length $\leq n-1$ from v_i to v_j iff the ij in at least one of $A, A^2, \ldots, A^{n-1}$ is 1 (where $n = |V|$).
- The ij entry of $A + A^2 + \cdots + A^n$ gives the total number of paths of length $\leq n$ from v_i to v_j.

Example:

$$A = \begin{bmatrix} 0 & 0 & 2 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

$$A^2 = AA = \begin{bmatrix} 0 & 0 & 2 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 2 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 2 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

$$A^3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 2 \\ 0 & 2 & 0 & 0 & 1 \end{bmatrix}$$

A Better Algorithm

Each time we multiply two $n \times n$ matrices, we need n multiplications to compute the ij entry, and thus n^3 multiplications altogether

- There are theoretically better algorithms for matrix multiplication that take roughly $n^{2.5}$ multiplications

Thus, to compute $A^1, \ldots, A^n$, requires roughly n^4 multiplications

- Could cut this down to $n^3 \log(n)$

Warshall's algorithm gives an even better approach to computing reachability.

- I won't cover Warshall's algorithm in class. You can read about it in the text if you want, but it won't be on the prelim/final.
- You can also use Dijkstra's algorithm (which I will cover) to compute reachability efficiently.

Transitive Closure

Recall that the *transitive closure* of a relation R is the least relation R^* such that

1. $R \subset R^*$
2. R^* is transitive (so that if $(u, v), (v, w) \in R^*$, then so is (u, w)).

How are the graphs $G(V, E)$ and $G^*(V, E^*)$ corresponding to R and R^* related?

- G^* is the result of putting an edge between u and v is there's a path from u to v in G

  How do we prove this?

  - Let $G_k(V, E_k)$ be such that there is an edge $(v, v') \in E_k$ iff there is a path of length $\leq k$ in the original graph G.
  - Let R_k be the relation corresponding to G_k.
  - Note that $R_1 = R$. Prove by induction that $R_k \subseteq R^*$ for all k. Then show that R_{n-1} is transitively closed, so $R_{n-1} = R^*$.

Tentative Prelim Coverage

- Chapter 0:
  - Sets
    - Set builder notation
    - Operations: union, intersection, complementation, set difference
  - Relations:
    - reflexive, symmetric, transitive, equivalence relations
  - Functions
    - Injective, surjective, bijective
  - Important functions and how to manipulate them:
    - exponent, logarithms, ceiling, floor, mod, polynomials
  - Summation and product notation
  - Matrices (especially how to multiply them)
  - Proof and logic concepts
    - logical notions ($\Rightarrow$, $\equiv$, $\neg$)
    - Proofs by contradiction

- Chapter 1
  - You do not have to write algorithms in their notation
  - You must be able to *read* algorithms in their notation
  - Procedures, recursion, recursive calls
  - Loop invariants
  - Analysis of algorithms
    - Relative ordering (n^2 vs. $n \log n$)
- Chapter 2
  - induction vs. strong induction
  - guessing the right inductive hypothesis
  - inductive (recursive) definitions
- Chapter 3
  - terminology: bipartite, complete, degree, path, tree, clique (number)
  - adjacency matrix
    - three representations of a relation
  - reachability and transitive closure

Shortest Paths

Suppose you have a graph with weights on the edges. (Think of the weights as driving times.) You want to find the minimum length path.

- if there are no weights on the edges, think of this as the special case where all the weights are 1.
- let $len(u, v)$ be the weight of the edge (u, v) ($len(u, v) = \infty$ if there is no edge from u to v).

Could do it by *brute force*:

- If there are n vertices, find all paths with no repeated vertices, and compute their weight.
- There could be as many as $(n - 2)!$ paths!

Can we do better?

Dijkstra's Algorithm: Key Idea

Suppose we want to find the shortest path from v_0 to v_n.

Generalize: Find the shortest path from v_0 to *every* other vertex.

How?

- First find the closest vertex and the path to it, then the next closest, and so on.
- Sooner or later v_n will be the next vertex added.

Why does this help?

- Can compute the next closest vertex recursively.

How do we find the vertex closest to v_0?

- Easy: just look

If $U = \{u_0, u_1, \ldots, u_k\}$ are the k closest vertices to v_0 (listed in order, with $u_0 = v_0$), how do we find u_{k+1}?

Suppose v is the next-closest vertex:

- The shortest path from v_0 to v must go through $\{u_1, \ldots, u_k\}$
  - If it got to v through some other vertex, that vertex would be closer to v_0 than v!
- That means the minimum length path from v_0 to v must have length

  $$d(v) = \min_{j=0}^{k}(d(u_j) + len(u_j, v)) \qquad (*)$$

  $len(u_j, v)$ is the weight of the edge from u_j to v
- Compute $(*)$ for each vertex not in U, and pick the shortest.

Dijkstra's Algorithm: Outline

At kth step of the algorithm, assume (inductively) we have:

- $u_1, \ldots, u_k$, the k closest vertices to v_0 (not counting v_0 itself)
- $d(u_j)$ (the minimum distance from v_0 to u_j)
- the minimum distance $d_k(v)$ from v_0 to any vertex v, going on path that involve only $u_1, \ldots, u_k$

At the $(k+1)$st step:

- for every vertex v connected to u_k, compute

$$d(u_k) + len(u_k, v)$$

- If this is better than $d_k(v)$, then let this be $d_{k+1}(v)$; otherwise $d_{k+1}(v) = d_k(v)$
- pick the $(k+1)$st closest vertex

Dijkstra's Algorithm: Example

k	$d(v_1)$	$d(v_2)$	$d(v_3)$	$d(v_4)$	$d(v_5)$	$d(v_6)$	$d(v_7)$	New
0	∞	∞	∞	∞	∞	∞	∞	v_0
1	2	4	∞	∞	∞	∞	∞	v_1
2	2	4	5	6	∞	∞	∞	v_2
3	2	4	5	5	∞	∞	∞	v_3
4	2	4	5	5	6	10	10	v_4
5	2	4	5	5	6	8	9	v_5
6	2	4	5	5	6	7	9	v_6
7	2	4	5	5	6	7	8	v_7

Dijsktra's Algorithm

Input $G(V, E)$ [a graph]
v_0, v_n [start and end]

Algorithm Shortest Path

$d(v_0) \leftarrow 0$ [Initialize distance from v_0]
for $i = 1$ **to** n [$n = |V|$]
    $d(v_i) \leftarrow \infty$
endfor
$U \leftarrow \{v_0\}$ [Initialize closest vertices]
$u \leftarrow v_0$ [u is most recent entry into U]
repeat until $u = v_n$
    for $i = 1$ **to** n
        if $(u, v_i) \in E$ and $v_i \notin U$, **then**
            $d(v_i) \leftarrow \min(d(v_i), d(u) + len(u, v_i))$
    endfor
    $mindist \leftarrow \infty$ [find next closest vertex]
    for $i = 1$ **to** n
        if $v_i \notin U$ and $d(v_i) < mindist$ **then**
            $mindist \leftarrow d(v_i);\ u \leftarrow v_i$
    endfor
    $U \leftarrow U \cup \{u\}$
endrepeat

Dijkstra's Algorithm: Correctness

Why is this right? Getting the right loop invariant is hard.

- What does $d(v_i)$ correspond to if $v_i \notin U$?

Loop invariant. For all $k \geq 0$, after the kth interation of the loop:

- U consists of v_0 and the k vertices closest to v_0
- u is the kth closest vertex to v_0
- if $v \in U$, then $d(v)$ is the length of the shortest path from v_0 to v.
- if $v \notin U$, then $d(v)$ is the length of the shortest path from v_0 to v that has only vertices in $U - \{u\}$ as intermediate points.
    - if there is no such path, then $d(v) = \infty$
- u is the most recently added vertex.

Once we get the right loop invariant, it's not hard to prove that it is maintained … by induction (of course):

Basis: $P(0)$ is obvious.

Inductive step: …