Announcements

- Hope you enjoyed the programming

- Next homework only theory

- Will post later tonight
Today

- We are talking about path finding algorithms and other graph related DSs....
path exploration

- We mentioned finding the shortest path between two nodes
  - price line tickets
  - Driving a delivery route from point a to point b
goal

- Given a weighted graph \( G(\mathcal{V}, \mathcal{E}) \) and a distinguished vertex \( s \), find shortest path from \( s \) to any other node in the graph

- although usually want \( s \) to a specific end point \( t \)
  - Once we find a short path, there might be an even shorter one....
  - So need to explore all paths to make sure we have the shortest
simplest

- Easiest is if only look at segments (unweighted edges) and want the least number of edges between s and every other node
un weighted shortest path

- $O(|V| + |E|)$

```c
int pathlen = 0;
foreach(v)
    v.distance = \infty;

s.distance = 0;
while(pathlen < numvertices) {
    foreach node.distance == pathlen;
        foreach adj node.distance = \infty
            distance = pathlen + 1
        pathlen++;
}
```
Question

- what expansion strategy was this?
- how to get the actual cheapest path?
Next

- now for weighted path
  - we want cheapest weighted path

- Dijkstra’s algorithm
  - greedy algorithm
  - Can’t deal with negative weighted paths....
Greedy algorithms

- Sometimes to solve a problem, a greedy approach works best
  - Tries best solution at each step
  - Sometimes can get stuck 😊
class Vertex
{
    public List adj;   // Adjacency list
    public boolean known;
    public DistType dist;   // DistType is probably int
    public Vertex path;
        // Other fields and methods as needed
}
/
* Print shortest path to v after dijkstra has run.
* Assume that the path exists.
*/
void printPath( Vertex v )
{
    if( v.path != null )
    {
        printPath( v.path );
        System.out.print( " to " );
    }
    System.out.print( v );
}
void dijkstra( Vertex s )
{
    for each Vertex v
    {
        v.dist = INFINITY;
        v.known = false;
    }

    s.dist = 0;

    for( ; ; )
    {
        Vertex v = smallest unknown distance vertex;
        if( v == NOT_A_VERTEX )
            break;
        v.known = true;

        for each Vertex w adjacent to v
            if( !w.known )
                if( v.dist + cvw < w.dist )
                {
                    // Update w
                    decrease( w.dist to v.dist + cvw );
                    w.path = v;
                }
    }
}
Next application

- Suppose you are hired to wire up a building
- Need to connect specific set of offices
  - Getting paid a flat fee
  - Looking to maximize your profit by laying minim amount of wiring
definition

- spanning tree
  - set of edges in the graph which will connect all nodes
  - let us use a directed graph as example

- what is the simplest example?
harder

- most of the time will have weight associated with the graph
- will want the minimum cost spanning tree connecting all nodes
- Example: laying down electric lines or paving roads
any ideas on generating this?
solutions

- prim’s algorithm
  - greedy solution
  - $O(V^2)$ but can be done in $O(E \log V)$ with heap

- kruskals algorithm
  - bottom up solution
  - $O(E \log V)$
prim’s algorithm

- V is your list of nodes
- choose random node to start MST
- while V notEmpty()
  - choose cheapest edge outgoing from your MST
  - remove from V, add to MST
  - avoid cycles
Kruskal’s algorithm

- look at all edges
- for use the cheapest edge to connect two nodes
  - as long as no cycles created
graph issues

- make sure you are clear on:
  - how would you represent a graph from the code point of view
  - how would you save a graph?
  - load a graph?
flow networks

- sometimes we are interested in graph as a capacity problem:
  - Represent graph of capacity between two points
  - Want to see what is the max flow the graph can carry between the points

- application: oil pipeline, electricity grid, etc
Given a graph with a flow capacity, what strategies can you think will allow you

- any flow
- max flow
- min flow
max flow algorithm

- will be using extra graph for book keeping

- will do next time
Another DS

- To solve this we need another data structure
- Disjointed sets
Disjointed sets

- Given a bunch of elements I want to be able to organize them into sets (collections) and be able to add and find specific elements

- Union
- Find
Equivalence Class ADT

- Union(a,b)
  - merges 2 equivalence classes

- Find(a)
  - retrieves equivalence class containing a
Implementation?

- Any ideas on implementing this DS?
- How long to do UNION??
- How long to do FIND??