Announcements

- ok we will be wrapping up with graphs and advanced data structures
- please check your grades/work submission online
- Final Exam : online (same as midterm). Will be available from Thursday 5pm - Friday 11pm
  - should be able to be completed in two and half hours or less
Definition

- A graph $G=(V,E)$ consists of a set of vertices (nodes) $V$ and edges $E$ where each edge is a pair $(v,w)$ such that $v,w \in V$. 
Directed Graph – A graph where vertices have direction

Undirected Graph

Weights – in addition can associate a weight with each edge

Example

think of a map
- cities (nodes)
- roads (edges)
- mileage (weights)

- path between two cities
- total road length
- shortest path between any two points
- capacity
Definition

- Path = a sequence of vertices $v_1, v_2, v_3...$ such that the vertices are adjacent

- Length = number of edges

- Loop = edges that starts and end at the same vertex

More definitions

- Simple path = a path with distinct vertices that doesn’t cross itself

- Cycle = path with the first and last vertex being the same
**definitions**

- **connected graph** = undirected graph where there is a path from any vertex to any other

**Strongly Connected**

- **directed graph** where path from any vertex to any other
more definitions

- weakly connected – directed graph which would have been connected if it was undirected

- Complete graph = every vertex connected to every other

Point

- Very general DS
  - very useful for real world problem representation

- Air transit
  - airports
  - flights
  - weight = cost

- Can answer:
  - what are the cheapest flights?
  - shorted flights?
  - where we should add flights?
  - can we reach every city?
one last definition

- **Degree of a node:**
  - undirected
    - number of connections of the node
  - Directed
    - in degree
    - out degree

- **Sparse Graph**
  - \( E << V \)

- **Dense Graph**
  - \( E \leq V \)

---

questions ?

- so how to code graph so arbitrary node labels ?
- how to grow over time ? (no preset number of nodes ?)
code

```java
DirectedWGraph dirG(20);
dirG.add("a","b",10);
dirG.add("b","c",5);
...
dirG.isDirEdge("a","c");
dirG.getW("a","b");
dirG.getEdgeCount();
dirG.getNodeCount();
```

Arrays implementation

- 2 dimensional n by n array
- weights are in location M,N to show directed edge from M to N
- -1 for no edge (if possible 0 weight)
- diagonal is M to itself

- great for dense graphs
Linked List implementation

- List representation called an adjacency list
- Array of nodes
- Linked list of edges
- How hashtable fits in here?

Resources

- How much space (in terms of V and E) would it take to represent a graph on either implementation?
- What does that tell you about your implementation choice?
further on...

- once you have your information in the graph, how do you save it?
  - serialization
  - file representation
    - compact
    - dump
    - considerations

- Data in graphs can be operated on in a variety of ways
  - Printing
  - Would like to support is searching the graph
    - We want to visit nodes without wasting time or missing any

Expansion

- starting from any node on a graph, we would like to reach others

- List method
- DFS
  - depth first search

- BFS
  - breadth first search
Expansion Algorithm

- have a list of vertices and nodes
- choose one from the list
- put into DS X
- while X is not empty
  - choose y out of X
    - if not visited, visit
      - for each adj of y if not visited put into X

- what is X ??

- STACK
  - DFS
- Queue
  - BFS
Topological sorting

- sometimes it's useful to see an ordered output of a graph, so there is some type of relationship information present in the output

- Topological sorting
  - an ordering of nodes in a DAG such that if exists a path from $v$ to $w$, then $w$ appears after $v$ in ordering

Algorithm

```c
for(i=0; i< num_vertices; i++) {
    v = findnewIndegreeZero();
    if (v==NULL) return;
    cout<<v;
    for(j=0; j<children(v); j++)
        nextchild(v).indegree--;  
}
```
complexity?

- $O(v+e)$

- what do you do about zero degree nodes?

bottom line

- ok so what else can we do with graphs??
  - relationship algorithms
  - path algorithms
  - minimum connectivity
path exploration

- We mentioned finding the shortest path between two nodes
  
  - price line tickets
  
  - so how would you do it ?? if you hadn’t read the chapter ??

goal

- Given a weighted graph G(V,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 t it’s the same computation to every other one
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

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

```java
int pathlen = 0;
foreach(v)
    v.distance = ∞;
s.distance =0;
while(pathlen < numvertices) {
    foreach node.distance == pathlen;
    foreach adj node.distance= ∞
        distance = pathlen +1
    pathlen++;
}
```
- what expansion strategy was this?
- how to get the actual cheapest path?

- now for weighted path
  - we want cheapest weighted path

- Dijkstra’s algorithm
  - greedy algorithm
  - doesn’t deal with negative weighted paths....
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

- 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

- hand out
- will be using extra graph for book keeping
- will review next time

algorithm design

- for solving any problem the algorithm is important
- but so is the data structure
- so how do you invent an algorithm ??
- different classes of algorithms we have seen
Greedy Algorithm

- many examples that we have seen
- “grab what you can and run “ philosophy
- settle for an answer not quite best, hoping we get close

Problem

- how would you program a cash register to spit out correct change for a transaction?
- let’s say we want to minimize the number of coins we are returning
Approaches

- usually largest coin first

- so for 15 cents?

- runtime?

- what would happen if we have a 12 cent coin?

from the past

- huffman was an example of a greedy algorithm

- that is why we needed to keep the tree local to the encryption so we can decrypt the data
packing problem

- say you are writing the software system for UPS/FEDEX how do you load the trucks going to destination x ??

- given N items of size $s_1$, $s_2$, ..$s_n$
- fewest number of bins to fit them in which each bin has a capacity

- example capacity 1,
- .2 .5 .4 .7 .1 .3 .8

constraints

- online vs offline

- real world online
  - what would be an algorithm
  - can you prove its optimal ?
strategies

- next fit
  - fit in last or create new bin
  - runtime ?
  - we can prove that if optimal is M, it will not use more than 2M bins

  - can you do the proof ??

proof

- consider 2 adjacent bins
- the sum must be greater than capacity (else they would be placed in one bin)
- which means we are wasting < ½ the capacity
**first fit**

- scan all current bin and place in first bin which can hold the current item
  - runtime?
  - can you improve it?
- it wont use more than 17/10 M bins 😊

**best fit**

- find where it fits in best
  - runtime?
  - worst case is exactly as first fit
  - easy to code
what can you say about the offline version of the problem?

can we get to optimal?

how?
problem

- given a set of points in 2 dimensional space
- can you find closest pair of points to any point?
- hmmm sounds familiar??
- will talk about it next time
  - do reading, catch up to homeworks please