Tutorial Week 2 - CSCB63 -Complexity

Draw a 2D table with column headers and row headers as follows:

	$\ln(n)$	$\lg(n)$	$\lg(n^2)$	$(lgn)^2$	n	$n*lg(n)$	2^n	$2^{(3n)}$
$\ln(n)$								
$\lg(n)$								
$\lg(n^2)$								
$(\lg n)^2$								
n								
$n*\lg(n)$								
2^n								
$2^{(3n)}$								

(Remember that lg means log base 2.)

In each cell, fill in "Y" iff (its row function)$\in O$(its column function).

We haven't talked about transitivity (if $f \in O(g)$ and $g \in O(h)$, then simply deduce $f \in O(h)$ and be done), but you may prove on your own and use it.

Observation. Even though $3n \in O(n)$, we cannot "exponentiate both sides" to infer $2^{3n} \in O(2^n)$.

Lets look 3 of the more interesting cells to show proofs for.

Fill in proofs for selected big-O cells:

1. $n \in O(n\lg(n))$, using the definition of big-O:
2. $n\lg(n) \notin O(n)$ using the definition of big-O: (Note: This can be explained as a proof by contradiction...other ways possible too).
3. $2^{(3n)} \notin O(2^n)$, using a limit theorem from lecture (you may not have seen the limit theorem if your tutorial is before the Wed. class).

   The limit theorem says:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \Rightarrow f(n) \notin O(g(n))$$

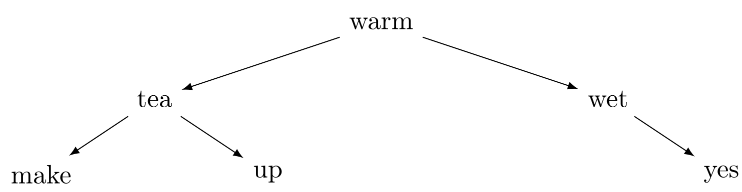
Some practice questions:

1. $6n^5 + n^2 - n^3 \in \Theta(n^5)$
2. $3n^2 - 4n \in \Omega(n^2)$

For these the intentions are to use the **definitions** of Theta and Omega not use the limit laws.

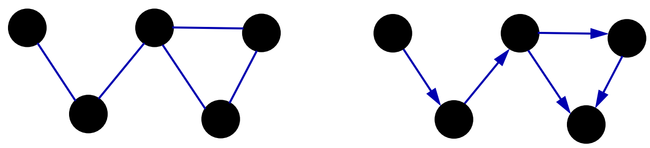
AVL Insert

Starting tree:



Insert honey, milk. No rotation. Insert cake. Insert vanilla.

Graph Theory Definitions

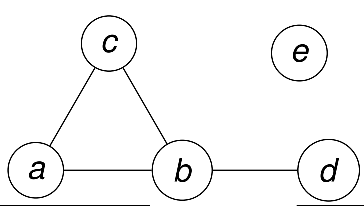


Undirected Graph   Directed Graph

- A *graph* $G = (V, E)$ consists of a set of *vertices* (or *nodes*) V and
- A set of *edges* E.
- Let $n = |V|$, the number of nodes, and $m = |E|$, the number of edges.
- In an *undirected* graph, each edge is a set of two vertices $\{u, v\}$ (so (u, v) and (v, u) are the same), and self-loops are *not allowed*. When it's clear from the context, we will use (u, v) for $\{u, v\}$.
- In a *directed* graph, each edge is an *ordered pair* of nodes (u, v) (so (u, v) is considered *different* from (v, u)); also, self-loops (edges of the form (u, u)) are allowed.

Terminology: Adjacent

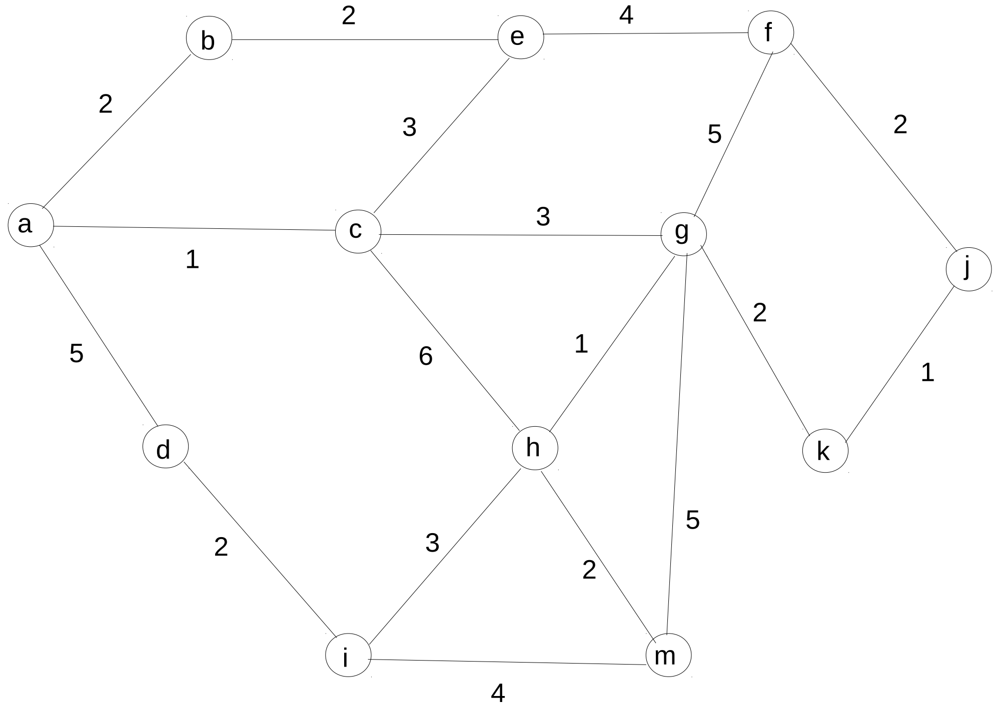
Two vertices are *adjacent* iff there is an edge between them.



	a	b	c	d	e
a		✓	✓		
b	✓		✓	✓	
c	✓	✓			
d		✓			
e					

	is adjacent to
a	b, c
b	a, c, d
c	a, b
d	b
e	

This brings us to two nice ways to store a graph. . .



Adjacency lists:

a: (b,2), (c,1), (d,5)
b: (a,2), (e,2)
c: (a,1), (e,3), (g,3), (h,6)
d: (a,5), (i,2)
e: (b,2), (c,3), (f,4)
f: (e,4), (g,5), (j,2)
g: (f,5), (c,3), (h,1), (m,5), (k,2)
h: (c,6), (g,1), (m,2), (i,3)
i: (d,2), (h,3), (m,4)
j: (k,1), (f,2)
m: (i,4), (h,2), (g,5)

Tutorial 7 – Dijkstra

Find all shortest paths from a start vertex s to every other vertex.

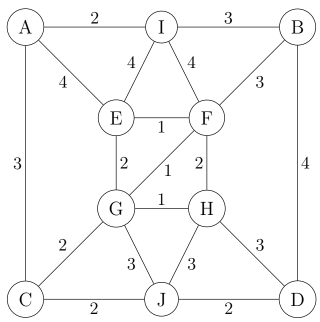
Dijkstra's algorithm finds the shortest paths by something similar to *breadth-first search*, but with a twist:

The *queue* is changed to a *min priority queue*.

The algorithm *grows a distance tree T* one edge at a time.

Priority of vertex v = *least weight path* between v and s so far. (∞ if no such path yet.)

Perform Dijkstra's algorithm on the following graph - choose B as your start vertex:



Make sure you include for each vertex `v` the distance `d[v]` of `v` from B as well as `p[v]` the parent of `v` in the distance tree.

```
dijkstra(G, s)
   PQ := new min-heap()
   PQ.insert(s, 0)
   d[s] := 0
   for each vertex z ≠ s:
      # initialize priority queue
      PQ.insert(z, ∞)
      d[z] := ∞
   while PQ not empty:
      #greedy choice of vertex to grow shortest path tree
      v := Q.extract-min()
      for each u in v's adjacency list:
         #Update priorities of adjacent nodes
         if d[v] + w({v,u}) < d[u]:
            PQ.decrease-priority(u, d[v] + w({v,u}))
            d[u] := d[v] + w({v,u})
            pred[u]:= v
```

Dijkstra Correctness

Fill in the missing steps of the correctness of Dijkstra:

- Let T_s be the *distance tree* constructed by *Dijkstra's Algorithm* starting at s.

Binary Counter Increment

Put a k-bit number in an array C of k bits. LSB at $C[0]$.
Initially all 0's.

increment():
    $i := 0$
    while $i < C$.length and $C[i] = 1$:
        $C[i] := 0$
        $i := i + 1$
    if $i < C$.length:
        $C[i] := 1$

(For this example: modifying a bit takes $\Theta(1)$ time.)

Up to k bits could be already 1. Increment takes $\Theta(k)$ time worst case. What about a sequence of m increments?

Tutorial 9 – Probability

The Assignment Due Date Paradox

You're taking 4 courses. All four Assignment 1's are due in week 4. Each prof independently chooses one weekday (Monday to Friday) for the due date.

1. How many ways are there overall?
2. How many ways are there such that all 4 assignments are due on different days, i.e., no two assignments are due on the same day? And so what is the probability that this happens?
3. How many ways are there such that at least two assignments are due on the same day? And probability?
4. You always have two assignments due on the same day, or two midterms on the same day. Is that conspiracy? Or just probability? :)

The i^{th} Ball

There are 6 red balls and 9 blue balls in a bag; when you draw a ball from the bag, each ball in the bag is equally likely drawn. Randomly draw 3 balls from the bag without replacement—after a ball is drawn, do not put it back into the bag. Find the probability that the i^{th} ball, ($1 \leq i \leq 3$), is one of the red balls.