Kruskal: MST algorithm
- initialize empty tree
- U-F datastructure
- L = sorted by cost
- for (u,v) in L
  - if find(u) != find(v)
    - → add (u,v) to tree
    - → union (u,v)
- return T

Time it takes:
- O(n) if done w/o rank/path compression
- O(1) if done w/ rank
- O(α(n)) if done w/ both

UNION FIND (DISJOINT SET) Data Structure

```
add_set(v)
  • parent[v] = v
  • rank[v] = 0  // rank
```

```
find(v)
  • if parent[v] = v
    → return v
  • else:
    → parent[v] = find(parent[v]) // pcomp
    → return parent[v]
```

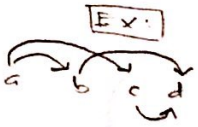
O(log n) if unioning by rank

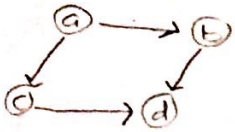
```
union_sets(u,v)
  • xRoot = find(x)
  • yRoot = find(y)
if rank [ • if rank(x) > rank(y) then parent[y] = x
        • else if rank(x) < rank(y) parent[x] = y
        • else parent[x] = y and rank[y] += 1
```

TOPOLOGICAL SORTING

- An ordering that all vertices go "forward" in ordering

Ex:






```
def topo_sort(G)
  • L = list()
  • for v in G:
    → if v not visited
      - topo_dfs(G,v)
  • return L
```

```
def topo_dfs(G,v)
  • mark v as visited
  • mark v as in recurs call
  • for u in neighbors of v {
    → if u not visited
      - topo_dfs(G,u)
    → if u is marked in rec call
      - throw exception (NOT A DAG)
  }
  • add v to front of L
  • unmark v as recurs call
```

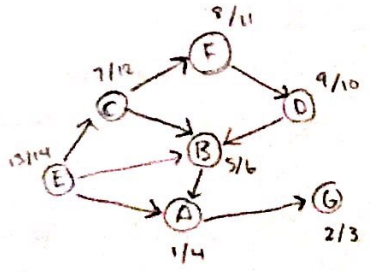
Time it takes

$O(V + E)$

↑ vertices ↑ Edges

asymptotically

Ex:



E C F D B A G

DIRECTED ACYCLIC GRAPHS (DAGS)

- No cycles or, a way to get back to vertex "v"

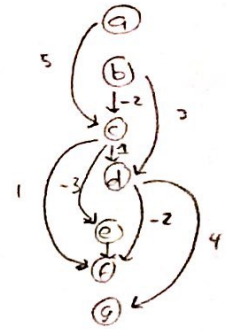
```
def dag_shortest_path(G,start)
  • L = topo_sort(G)
  • d[start] = 0, d[u] = ∞ // every other vertice
  • // q = queue pushed w/ L in order
  • for v in L {  // or while q is not empty
      → v = q.pop() //
      → for u in neighbors of v
          if d[v] + cost(u,v) < d[u]
              d[u] = d[v] + cost(u,v)
  }
  • return d
```

Time it takes

$O(n+m)$

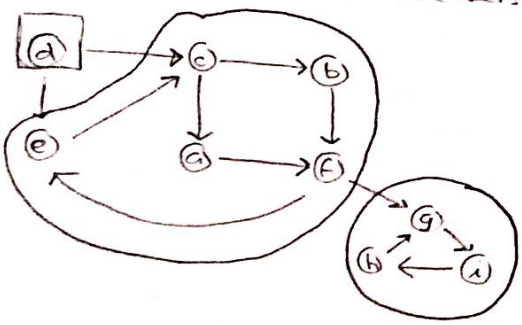
Ex:

d:
- a → 0
- b → ∞
- c → 5
- d → 6
- e → 7
- f → 4
- g → 16



TARJAN'S ALGORITHM for Strongly connected components

- two vertices are strongly connected if there is a path there and back
  "u and v are strongly connected if and only if there's a path from u to v and vise versa"




```
def tdfs(G,v)
  ...
```

Time it takes

$O(n+m)$

TRAVELING SALESPERSON PROBLEM (TSP) "An NP-problem"

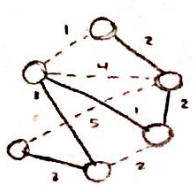
- Problem to find minimum cost to visit every vertex and ending where it started

"minimum cost tour"

Minimum Spanning Tree (MST)

- Problem of finding tree that touches n vertices and minimum cost among those trees

Ex:



Bold = MST
Dotted = possible paths

Whole thing is TSP
BOLD is MST

FINAL REVIEW ICS 46

pg. 1 of 2

P vs. NP

P - polynomial decision problems that can be solved in polynomial time

NP - Nondeterministic Polynomial decision problems that can be verified in polynomial time

"All problems that can be solved in polynomial time can be verified in polynomial time."

essentially $(P \subseteq NP)$ set-wise

Dijkstra

- finding shortest paths between nodes

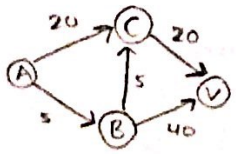
```
def dijkstra:
    distance = hash()
    PQ = priority_queue()
    for v in G:                          # initialize values v
        if v not start
            distance[v] = ∞
            PQ.insert(∞, v)
        else
            distance[v] = 0
            PQ.insert(0, v)

while PQ not empty
    d, v = PQ.remove_min()
    if v == end
        return d
    for u neighbors of v
        if distance[u] > d + cost(v,u)
            PQ.update(u, d + cost(v,u))
            distance[u] = d + cost(v,u)
"raise exception"
```

Time it takes

PQ	remove min	update	dijkstra runtime
Binary Heap	O(log n)	O(log n)	O(n log n + m log n)
unsorted array	O(n)	O(1)	$O(n^2 + m)$
Fibonacci heap	O(log n)	O(1)	O(n log n + m)

Ex:



write in order distances assigned to each starting from A

start	a	b	c
a→0	a→0	a→0	a→0
b→∞	b→5	b→5	b→5
c→∞	c→20	c→10	c→10
v→∞	v→∞	v→40	v→30

Bellmans-Ford

- Same as dijkstra but can run negative edges but SLOWER

- Has an exception thrown

BFS / DFS

Breadth-first-search

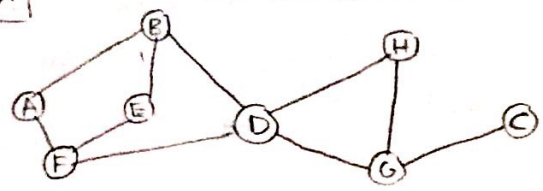
- It starts at tree root, explores neighbor nodes first before moving onto next level neighbors
- uses priority queue
- Time : O(n+m)

Depth-first search

- It starts at tree root, explores as far as possible before backtracking
- uses stack?
- Time : O(n+m)

Ex.

starting from A



BFS

PQ: ~~A~~ ~~B~~ ~~F~~ ~~E~~ ~~D~~ G H

order: A B F E D G H C

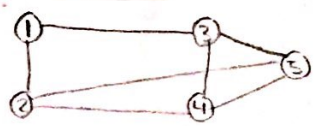
DFS

stack: A B D ~~G~~ ~~H~~

order: A B D F E G C H

GRAPH

Undirected



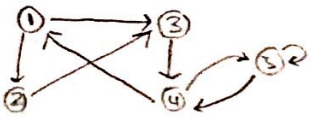
(2,5) is an edge

(2,3) is not

degree(5) = 3

degrees are how many so in and out of vertex node

directed



indegree(1) = 1

outdegree(1) = 2

indegree = edges going into node

outdegree = edges going out of node

- Max # of edges for undirected is $\binom{n}{2} = \frac{n^2-n}{2}$
- Max # of edges for directed is n^2

ADJACENCY LIST / MATRIX

LIST

(i)

	0	1	2	3
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15

(A)

4i + j ← (i,j)

vertex	neighbors
1	2,3
2	1,4,5
3	1,4,5
4	2,3,5
5	2,3,4

MATRIX

Undirected

	1	2	3	4	5
1		1	1		
2	1			1	1
3	1			1	1
4		1	1		1
5		1	1	1	

Directed

	1	2	3	4	5
1		1	1		
2			1		
3				1	
4	1				1
5				1	1

FINAL REVIEW ICS 46

PS. #13

MULTIDIMENSIONAL ARRAY (in single array)

- vector<vector<int>> v
- int [n²] m
- m[n * i + j]
  - n: total # in each array
  - i: col
  - j: row

n=4

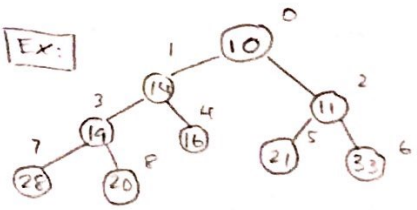
PRIORITY QUEUE & BINARY HEAPS

- Binary heap is O(log n) for insert, remove min, update
- unsorted array is O(1) for insert, O(n) for remove min / update ✓ O(1) if can locate it

BINARY HEAP

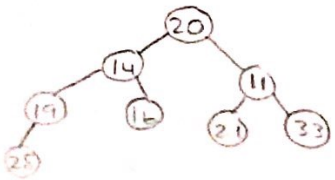
- Parents smaller than children
- complete : left to right

Ex:

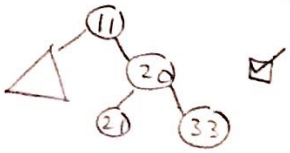


remove_min()

- swaps w/ bottom right most or "n-1"



- keep swapping w/ smaller of two values

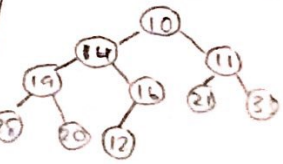


☑

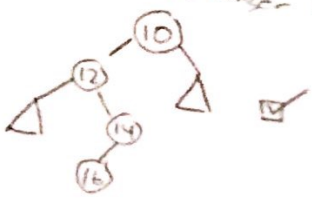
percolates down

insert (10)

- add onto "n"



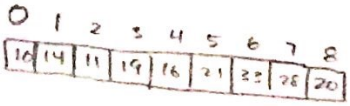
- swap until new node's parent is no longer bigger



☑

percolates up

Storing complete binary tree using array



node at position i

- Left child => 2i+1
- Right child => 2i+2

Parent of i => $\frac{i-1}{2}$

There can be an issue w/ heap sort if two same values on same level but we do not know how to promote which one

Things before/on Midterm

Traversal

in-order: DATASTRUCTURES

pre-order: UAADTTSRRTCUES

post-order: DTASRTACUTSERU

Full: All nodes have 0 or 2 children

complete: left filled

perfect: All filled

BIG O-NOTATION

- $f(n) = O(g(n))$

$O(n)$ ↓ $\frac{f(n)}{g(n)} < \infty$ — upperbound

$\Omega(n)$ $\frac{g(n)}{f(n)} < \infty$ — lowerbound

$\Theta(n)$

ICS 46 Review for Midterm pg. #1

INVERSION

- A pair of indices i, j s.t. $i < j$ but $A[j] < A[i]$

Ex. 3, 2, 15, 32, 7, 5 (indices 1, 2, 3, 4, 5, 6)

(1,2), (3,5), (3,6), (4,5), (4,6), (5,6) ☑

6 inversions

Insertion Sort

WATCH VIDEO

- $O(n^2)$ ← worst case ← completely inverted
- $O(n)$ ← Best case ← already sorted
- $O(n+k)$ ← Array of length n w/ k inversions

Merge Sort

WATCH VIDEO

- $O(n \log n)$
- divides list into smaller units (one) then eventually compare each element until it's a complete adjacent list

BIG-O NOTATION

WATCH VIDEO

- $f(n) = O(g(n))$
- $\lim_{n \to \infty} \frac{f(n)}{g(n)} < \infty$

$g(n)$ is an asymptotic upperbound on $f(n)$

Ex. Is $\sqrt{n} = O(n)$?

$\lim_{n \to \infty} \frac{\sqrt{n}}{n} = \frac{1}{n^{1/2}} = 0 < \infty$ ☑

Ω - NOTATION

WATCH VIDEO

- Asymptotic lower-bound growth
- $\lim_{n \to \infty} \frac{g(n)}{f(n)} < \infty$

Is $\sqrt{n} = \Omega(n)$?

$\lim_{n \to \infty} \frac{n}{\sqrt{n}} = \sqrt{n} = \infty$

NO.

COMPARISON SORT

- A sorting algorithm that compares elements

Θ - NOTATION

- same asymptotic growth
- both O-Notation and Ω-notation

LINKED LISTS

- Array based (dynamic) list
- $O(1)$ ← INSERT AT BEGINNING
- $\Theta(n)$ ← " " END
- $\Theta(1)$ " " POSITION i

Amortized

WATCH VIDEO

- average cost of a sequence of operations
- how much time it takes to execute

HASH MAP

- Collision Resolution
  - Separate chaining
  - Linear Probing

SEPARATE CHAINING

Worst case	Avg. case
$O(1)$ ← insert ?	$O(1)$ ← insert
$O(n)$ ← delete	$O(1 + \frac{n}{m})$ ← delete
$O(n)$ ← lookup	$O(1 + \frac{n}{m})$ ← lookup

m ↑ load factor

Is it $O(n)$ since we could traverse through a linked list?

LINEAR PROBING

Avg:

insert → $O(1)$

delete →

lookup →

WATCH VIDEO

BUCKET SORT

- A stable sort w/ equal keys w/ diff values appear in the output in same order as they did in input
- It takes $\Theta(n+b)$

Ex: $2_a\ 1_b\ 2_c\ 1_d\ 3_e\ 3_f$

↓

$1_b, 1_d, 2_a, 2_c, 3_e, 3_f$

WATCH VIDEO

- Worst case: $O(n^2)$ ← Falls in same bucket
- Best case $\Omega(n+k)$ ← # of buckets

RADIX SORT

- $\Theta(d(n+b))$
  - d: # of digits
  - n: # of elements
  - b: base of b
- "Interesting" case

$\Theta\left(n \cdot \left\lceil \frac{\log N}{\log n} \right\rceil\right)$ - where N is largest #

WATCH VIDEO

- store lowest "column" first then biggest

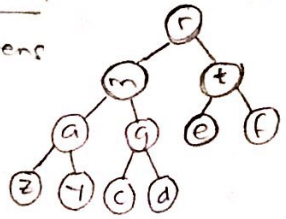
Ex: [359, 911, 456]

#1	#2	#3
911	911	359
456	⇒ 456	⇒ 456
359	359	911

Binary Rooted Trees

- Trees w/ 2 childrens
- Insert ← $\Theta(h)$
- lookup ← $\Theta(h)$
- delete ← $\Theta(h)$

- Study in-order, pre-order and post-order



in-order: zaymcgdretf

pre-order: rmazygcdtef

post-order: zyacdgmeftr

visit left, visit root, visit right

WATCH VIDEO

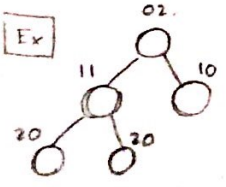
FULL - All nodes have zero or two children

COMPLETE - Left is filled, right isn't

Perfect - Leaf have zero children

- depth is distance from the root
- height is distance from bottom

Ex



Depth / height

MIDTERM REVIEW pg. #2

Quicksort

- choose random "median"
- smaller or equal to value to left list
- bigger values to right list
- Work of appending lists
- $O(n \log n)$ time avg case / best
  $O(n^2)$ worst case

WATCH VIDEO

- divide and conquer method

AVL trees

- All nodes' height balance factor is -1, 0 or +1

WATCH VIDEO

- Rotations? Insertion / Deletion?

STACK AND QUEUE

- Stack: LIFO → Last in First out
- Queue: FIFO → First In First out

Ex: 1 6 1 8 * * 2 9 * 3 * * 8 * * *

Stack
8 1 9 3 2 8 6 1

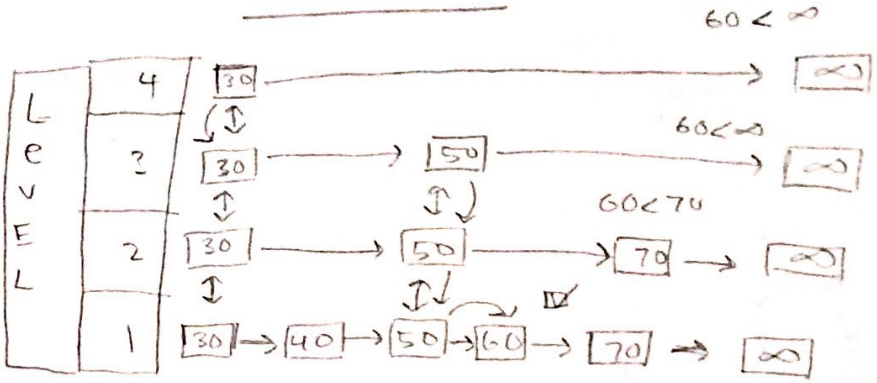
Queue
1 6 1 8 2 9 3 8

SKIP LISTS

	worst case	avg.
- Inserting	O(n)	O(log n)
lookup	O(n)	O(log n)
delete	O(n)	O(log n)

- A data structure that allows fast search through maintains a hierarchy of subsequences
- "coin flip" to add "columns"

WATCH VIDEO



Finding 60 ↑