20 JAN - 26 JAN

Hashing — insert, search, delete $O(1)$. no successor & predecessor.

$\boxed{\text{Keys should be immutable}}$

- hashing with chaining.
  - ↳ insert $O(1)$ worst

m key, n items. → $O(m+n)$ space

$\boxed{\text{load} : \frac{n}{m} = \alpha}$ → $O(1+\frac{n}{m})$

$\frac{n}{m} \approx O(1)$.

Worst case: all hash to one → sorted array ($O(\log n)$)
general case: $O(n)$ worst for search/remove.
~~insert still $O(1)$ too~~

max expected $\frac{\log n}{\log\log n}$ → Poisson dist

Open Addressing.

- search → hash(key, 0)
  h (key, 1)
  h (key, 2)
- delete → $\boxed{\text{tombstone}}$

$\boxed{\text{When full, cannot search \& insert anymore}}$
↳ need resize

$\boxed{\text{WORST CASE still } O(N)}$

If insert sees tombstone → just change to new var.

2 prop of good hash

- $h(key, i)$ $0 \le i \le m$ $\boxed{\text{permutation}}$ of $\{0, \dots, m\}$. $\exists\, i$ s.t. $h(key, i) = j$
- $\forall$ key → Prob slot to key $i = \boxed{\frac{1}{m}}$ $\forall i$

20 JAN - 26 JAN

What about double hashing ??

$h(key) = f(key) + i \cdot g(key) \cdot \bmod m$.

relatively prime

$m = 2^a$.

Worst / Avg

	Search	Insert	Del
Sorted Arr	$\log N / \log N$	$N / N/2$	$N / N/2$
Unsorted	$N / N/2$	N / N	$N / N/2$
LL	$N / 1$	$N / 1$	$N / 1$
lnar probe	$N / 1$	$N / 1$	$N / 1$

double hash: hard to implement DELETE

better cache (array) over LL.

Load $\alpha = \frac{n}{m}$ expected cost of search & insert $\boxed{\frac{1}{1-\alpha}}$

sensitive to α & hash

↳ as $\alpha \to 1$ $\boxed{\text{bad}}$

$\alpha < 0.75$.

m bucket n items inside it.

$\frac{n}{m} \cdot \left(\frac{n-1}{m-1}\right) \cdots$

$\left(1-\frac{n}{m}\right) + \frac{n}{m}\left(1-\frac{n-1}{m-1}\right) +$

$1-\alpha + \alpha(1-\alpha) +$

$1 + \frac{n}{m}\left(1 + \frac{n-1}{m-1}\left(1 + \cdots\right.\right.$

$1 + \alpha(1 + \alpha(1 + \alpha + \cdots$

$\boxed{\frac{1}{1-\alpha}}$

Linear probing usually choose $\boxed{M = 2n}$

To grow table = O(old + new ~~#elements~~)

if $m \le \frac{n}{2}$ → $m \to 2m$.

$m > \frac{n}{2}$ → $m \to \frac{m}{2}$

JANUARY 2020 FINGERPRINT

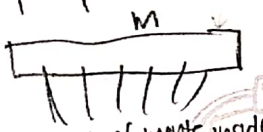
27 JAN - 2 FEB

Key idea : storing big keys — long strings / image } is costly.

so use fingerprints → DO NOT store key in table

↳ maybe false positives
NO FALSE NEGATIVES

Prob of false positives :



$\left(1-\frac{1}{m}\right)^n \approx \left(\frac{1}{e}\right)^{\frac{n}{m}}$ $1-\frac{1}{m}$: P(one bit is not set aft mapping of 1 obj)

$P_{\text{a query return a false}+} = P_{\text{an arb bit is set}} = 1-\left(1-\frac{1}{m}\right)^n$.

$\Leftrightarrow \frac{n}{m} \le \log \frac{1}{1-p}$.

$\Leftrightarrow m \geq 13n$

Trade off

⊕ less space to store key

⊖ bigger table to reduce collisions

⓪ Bloom Filter → 2 hash fn. → map to 2 bits.

search = true if both bit is true.

$P_{\text{a query return false}+} = \left(1-\left(1-\frac{1}{m}\right)^{2n}\right)^2$

- What about k hash fns → there is optimal number

$k = \frac{m}{n} \ln 2 \rightarrow P_{error} = 2^{-k}$

why PQ is natural for heaps!!

PQ & heaps

SEMESTER 2 WEEK 3

↳ insert, extract Min, decrease key (decrease priority) 27 JAN - 2 FEB

$O(\log n)$ $O(\log n)$ $O(\log n)$

need to bubble down

lookup, delete, insert

usually combined with HT <priority val, Node/Array slots>

might be $O(n)$ w/o a HT

! • All leaves are to the left

Complete Binary Tree.

↳ max Height: $\lfloor \log(n) \rfloor$

- no rotations

$\log n$ { remove : swap w/ last, then bubble down
insert : add @ last, then bubble up.
extract Max : swap root w/ last, then bubble down from the root

HeapSort : extract max → put at the back

UNSTABLE safe because at extract max, we have swapped last to idx 0

Worst case $n \log n$

in place : space : $O(1)$

Heap successor : $O(n)$ → searching whole array.

NOTES

Graphs as adjacency list : $O(V+E)$ space → good for sparse graphs.
- ✓ list all neighbor $O(n)$
- ✓ any $O(1)$
- ✗ are v and w neighbor $O(n)$

as matrix : $O(V^2)$ space → good for dense graph → good: are v and w neighbors? random access $O(1)$
- ~~bad~~: list all neighbor $O(n)$
- ✗ find any neighbor $O(n)$

there is some order

Directed Graphs as adj. list. $O(V+E)$ space → only maintains outgoing/ingoing edge.

↳ scheduling

$O(V^2)$ space → $A[v,w]$ = v to w) not symmetric anymore

DAG iff Topo Order

DAG → TOPO SORT $O(V+E)$ → Topo Order

↑ post-order DFS = $O(V+E)$.
- ~~put to the back~~
- process a node iff all outgoing edge have been processed! → put to back
  (i.e process node after calling DFS-visit recursive call)

Algo 1

Repeat:
- S = all node w/ no incoming edge
- add Nodes to topo order
- Remove all edge adj to S
- remove S from graph

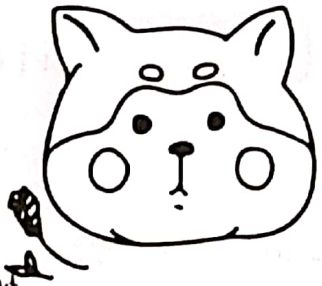
$O(V+E)$

each v only processed once

Algo 2
Kahn

Strongly connected component
∃ path v→w & w→v

Graphs of strongly connected component is acyclic.



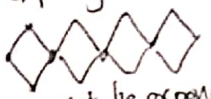
BREAK TIME

Week 4 to Recess Week
AY 19/20 Sem 2

```
class NeighborList extends ArrayList

class Node {
    ~~value~~ int key
    NeighborList nbrs
}
class Graph {
    Node[]
```

exploring all paths



might be exponential.

DO NOT VISIT EVERY PATH POSSIBLE

FEBRUARY 2020

efflorescence

DFS → recursive Iterative: use Stack!

```
For all v ∈ V:
    if (!visited[v])
        DFS-visit(v)

DFS-visit (Node start).
    process (start)
    for all v ∈ Neighbor(start):
        if (!visited[v]):
            visited[v] = true
            DFS-visit(v).
```

BFS → recursive Iterative: use Queue

↗ min number of HOPS
P.S: In unweighted graph = shortest path

```
frontier = new ArrList<>
frontier add (start)
while(frontier not empty)
    nextfrontier add (frontier.children)
    frontier = nextfrontier
```

Only visit each node once.
Need to keep track visited in RECURSIVE

(adj) flowers blooming; growth and development

O(V) space

NOTES

BFS / DFS : O(V+E) → adjacency list } that is only sparse = use list
↑ Store visited[]
O(V²) → adjacency matrix.

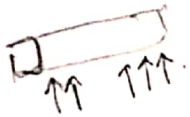
BFS → shortest path → since only visit every node once and visit in order
O(V+E)

- each node only visited once
- each neighbour of node is enumerated.
  - ↳ in adj list ~ length of list
  - ↳ in adj matrix ~ O(V) per node

DFS → in adjacency matrix O(V²)
searching neighbours : O(V).
↳ detect cycle?

BFS & DFS ← doesn't guarantee SP (except tree?).
BFS ↓ Queue
DFS ↳ Stack

→ if not visited



Shortest Path.

NO REWEIGHT

↳ unweighted : BFS + store parent pointer

3 FEB - 9 FEB

BF → relax every edge ⇒ repeat V times.
(SSSP)

V-1 times + 1 times
check for ⊖ / end.

Invariant: at i^{th} iteration, at least i node has been final. (i Hop is correct)

at i^{th} iteration, we have find all SP of at most length i from the SOURCE

SPFA avg O(E)?

use Queue. Conly relax (in Q) aftrelax put back to Q.

Detect ⊖ cycles

BF don't work for Graph w/ ⊖ cycles but can detect

Dijkstra. Relax in correct order, so only relax every edge ONCE

Invariant: at each iteration → add 1 node to final

build Shortest Path Tree

modify relax + decrease Key / insert

Priority Queue.

Binary Heap:
- deleteMin ; swap w/ last, remove, bubble down
- Insert : insert at end, bubble up.

log

decreaseKey: delete (swap, remove) + insert

contains : HT (idx & value).

AVL Tree
- deleteMin() } V times
- insert
- decreaseKey } E times
- contains : HT (keys) (idx in tree)

relax (int u, int v)
if (dist[v] > dist[u] + weight(u,v))
dist[v] = dist[u] + weight(u,v);

3 FEB - 9 FEB

DAG

Why Topo order?
- If P = S - X - D is SP from S-D
  Then P also contains SP from S-X, X-D
- Path relaxed s.t. dist aft relax is ☑

• Longest path on DAGs → ⊖ the edges

Input	
Whatever	BF, O(VE)
unweighted all same	BFS O(V+E)
Tree (undirected, weighted, no cycle)	BFS/DFS O(V)
NonNeg Edges	Dijkstra O(E log V)
DAGs	Topo+Relax O(V+E).

Queue BFS : Take v discovered least recently (FIFO)
Stack DFS (LIFO)
PQ Dijkstra Take v closest to source

Idea: Maintain set of explored vertices.
- add v by following e that go from explored → unexplored

→ at every step we add an edge crossing the cut (if in Mstang)

Repeat:
- Find unfinished vertice w/ smallest estimate
- Relax all outsons
- Mark vertex finished

Basic Idea: maintain dist estimates.

	ins	delMin	decKey	
Array	1	V	1	V^2
AVL	log V	log V	log V	E log V
d-way heap	d log V	d log d V	1	E log V
Fib heap	1	log V	1	E + V log V

undirected? replace (u,v) w/ u→v & v→u.

Dijkstra? → cannot have neg edges.

So if your undirected graph all has ⊖ edge

BF? → no ⊖ cycle. if u,v has neg weight then

Is a ⊖ cycle

So, also not possible. Just use Dijkstra.

LONGEST PATH

↳ DAG → Topo Sort + Relax

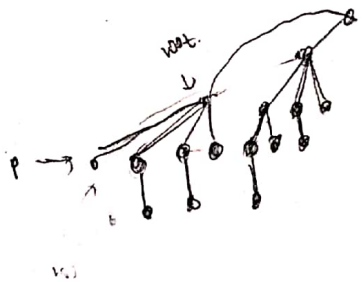
→ ⊖ all edge / modify relax

↳ general cyclic graph ⊗ can have positive weight cycle

↓ ⊖

negative weight cycle

our SP algo's will fail



UF-DS → to support Kruskal & Boruvka.

Data Structure · 1d Array.

Objects → HT → integer + OpenAddressing

key	0	1	2	3	. . .	8
parent, size					. . .	

Quick Find : O(1) find, O(n) union.

union(p,q) : all el in component q → change parent to p.

Quick Union : O(n) find, O(n) union → union(p,q) = parent(parent(p)) = parent(q)

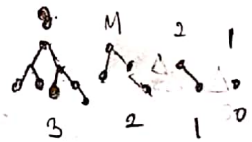
↳ have to search for both parent → O(n) in skewed tree

PROBLEM

Introduce Path Compression || Weighted QU.

→ O(log n) find, O(logn) union

WQU → height only ↑ iff 2 trees of same h is combined.



$h \le \log n$.

$h = \lfloor \log n \rfloor$

Ackermann f.

WQUPC : $O(\alpha(m,n))$ find. , $O(A)$ union

Worst case ~~for~~ QUPC : O(n)

WQU : O(logn)

FEBUARY 2020 MST → one CC.

17 FEB - 23 FEB

$\boxed{\text{MST cannot be used to find SP}}$

but can be used to find minimax.

Network Design

bottleneck

① No cycle → a tree?

② Cut an MST → 2 MSTs → BUT: ∀ cut, can have ≥ 1 edge connecting (lightest edge, no guarantee!) → can be heaviest on other cycle

③ Cycle property: heaviest edge on every cycle → RED

④ Cut property: min edge crossing a cut → BLUE

(min outgoing edge of every node in MST ✓

because it cuts node | everything else

max outgoing edge not in MST ✗.

Cut an MST → 2 MSTs

Generic MST - color R/B for every edge

on termination → ∀ cycle, ∃ 1 red edge

→ blue edge → forest

→ blue edge, MST, if not ∃ cut, no blue edge.

$\boxed{\text{PRIM}}$ → BLUE-only strategy (each added edge is lightest on some cut)

PRIM vs Dijkstra

edge weight	estimate + edge weight (distance)

→ keep extra arr of dist[]

initialization:

- all v → ∞
- start → 0

PQ.

DS used.

HashMap <Node, Node> parent

HashSet <Node> seen.

Dijkstra don't hv this cos each node can only visit once

$O(E\log V)$.

space: $O(V)$

Prim: E insert

E decrease key

V extract Min

← DS-used (UF D-S). Space: $O(V)$. ← UF

SEMESTER 2 WEEK 6

17 FEB - 23 FEB

$\boxed{\text{KRUSKAL}}$ → sort, then use UFDS to determine (find) whether it's connected / not

↓ do nothing

↓ union & save edge.

sort $O(E\log E) \approx (E\log V)$.

+ ↳ use RED & BLUE strategy.

find/union $E \times O(\log V)$

or

$E \times O(A)$ +

$O(E\log V)$

Kruskal: $E\log E$ sort

E union-find

$\boxed{\text{BORUVKA}}$ → parallelizable → use UDFS

① Add all out edges (i.e) cut property corollary

↳ add min edge out of every component / subset

@ the beginning, all nodes are component.

One boruvka "step"

- search for min outgoing edge ∀ component → find parent - store in cheapest[parent] BFS/DFS per component OR Iterate through edgelist and store it in same cheapest
- add to MST
- contract / merge connected components (union)

$O(V+E)$

$O(V)$

Iterate through cheapest & union.

There is log V boruvka step. (each step: $V \to \leq \frac{V}{2}$ CC)

Boruvka: $O(V+E)$ cheapest

$O(V)$ merge components

MST special cases

→ connect all edges. w/ min total weight

24 FEB - 1 MAR

Input	
unweighted/ homogenous	BFS / DFS. → V-1 edges only. (guaranteed). O(V+E)
edges range {1.. k} constant k.	O(E)! or O(V+E)

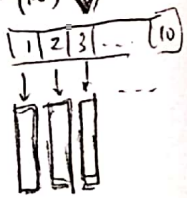
diff w/ SP problem is that S → T might vary in length, DFS doesn't guarantee shortest

① Kruskal Variant!

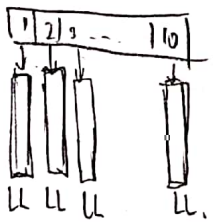
↳ ① sort : O(E). Use k buckets, iterate through array.

② UF : O(αE) → connect/not (find) → union

Total : O(αE)



② Prim Variant



PQ how? → If outdated, next extractMin

O(V) { - extractMin → O(1) just iterate 1..k
- contains → HashSet.

O(E) { - insert → O(1) access + O(1) LL insert.
- decreasekey → Lazy deletion??

O(V+E).

2OPT

$6' \leq 2OPT$

$T \leq 6'$

Directed MST → Arborescence Problem on Rooted Tree

- ∀ node reachable on path from root.
- no cycles

↳ Classic Algo doesn't work.

DAG w/ one Root ∀ node not Root, add min. incoming edge

O(V+E)

Max ST → reweight (+/- constant doesn't matter. WHY? it is +/- c.(V-1) for all results)

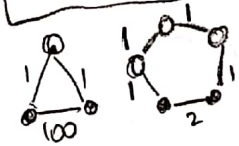
→ just - all edges run MST (find most "negative").

or sort Kruskal descending (?)

or Prim's PQ max-heap instead of min-heap.

Steiner Tree → MST of a subset of graph, but can use other nodes optionally (Steiner nodes)

(required nodes)



→ Approximation **Algo**

① ∀ pair, find shortest path ← SP (weight).

② construct new Graph G(V', E') ← required

③ MST on new Graph G'

④ map back to original graph

Sketch Proof: DFS on Opt.

→ ↳ trace each edge

→ ↳ kill Steiner nodes

↳ nodes all on G' and subset of its edges

↳ T is MST on G'

NOTES

DP. / memo

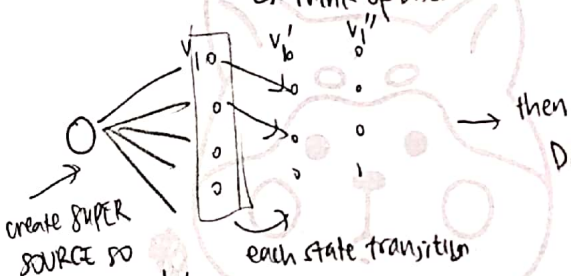
Longest Increasing Subsequence
- → longest ending at _ (fr → back)
- ↳ longest starting at _ (fr ← back)

$O(n^2)$

Prize Collecting

- Detect cycle : BFS / $\boxed{\text{DFS}}$

→ max possible income w/ max k edges travelled.
(longest path problem).
↳ Think of DAG.



→ then run longest path on DAG → $O(|V|+|E|)$

$O(V+E) = O((kV+1)+kE+V) = O(kV+kE)$.

space $O(kV)$

DP approach : $P[v,k]$ starting from v take k steps } max amount can be collected.

$P[v,k] = \max\{P[w_i,k-1] + w(w_i,v)\}$

$w_i \in$ Neighbor of v.

Table View

	v_1	v_2	v_3	...	v_V
0					
1					
2					
⋮					
k					

- kV sub problems @ v.neighbor(v)
  $O(kV^2)$
- total $O(kE)$ ← total row : k
  ← each for each row $= O(E)$.

2
2 ~~5~~ 3.
2 3 7
2 3 7 ~~11~~ 8.
2 3 7 8 9.

Vertex Cover

Input : rooted, unweighted, undirected tree
return size of vertex cover.

Subproblems :
- → included. ← children include / children not included
- → not included. $\approx g_i$
  ↳ all children must be included

2V subproblems $* O(v)$. foreach / $O(E)$.

AND IT CONTINUES

Week 7 to Week 10
AY 19/20 Sem 2

add MINIMAX problem

Subproblem of SP: If $u \to v \to w$ is SP from $u \to w$
It is also SP $u \to v$ & $v \to w$

MARCH 2020

ineffable

(adj) too great to be expressed in words

NOTES

APSP → Run SSSP v times. $O(V(V+E))$ BFS on unweighted
$O(VE\log V)$

Subproblem: $A \to (?) \to B$

$|?| = 0, 1, \ldots, v-2$.

$S[v, w, P]$

$P_0 = \phi$ $P_{i+1} = P_i \cup \{i+1\}$

basecase: $S[v, w, \phi] = E[v, w]$

$S[v, w, P_{i+1}] = \min(S[v, w, P_i], S[v, i+1, P_i] + S[i+1, w, P_i])$

making use of subproblem $u \to v \to w$
$u \to v$, $v \to w$

FW → $O(n^3)$.

Filling in 3D-array start, end, set $1, 2, \ldots, V$

get least significant bit: n & $n-1$

MARCH 2020

MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY/SUNDAY
30	31				
					1
2		4	5	6	7
					8
9		11	12	13	14
					15
16	17	18	19	20	21
					22
23	24	25	26	27	28
					29

DFS detect cycle

```
dfs (int v, int parent) {
    color[v] = 1
    for (int w : neighbors[v])
        if (color[w] == 1) . cycle ?
        else if (color[w] == 0) dfs(w, v)
}
color[v] = 2
```