17 Mar 2025 - Algorithms - Week 11

Longest increasing subsequence

Array A

3	7	2	0	9	12	5	27	4

→ Size : 5

① Dynamic Programming

3	7	2	0	9	12	5	27	4

↑ ↑ ↑ ↑ ↑ . . . ↑

possible ending

→ Size : 5

$S[i][1]$ ⟶ length of the longest increasing subsequence of $A[1..i]$ which contain element $A[i]$

To backtrack

$S[i][2]$ ⟶ index of the previous element in the longest increasing subsequence of $A[1..i]$ which is ending at $A[i]$.

$S[1][1]$, $S[2][1]$, ... $S[i-1][1]$

$S[1][2]$, ... $S[i-1][2]$

$$S[i][1] = 1 + \max_{1 \le j < i} (S[j], 1)$$ ✗

$$\begin{cases} S[i][1] = 1 + \underbrace{\max_{j<i} \{ S[j][1] : A[j] < A[i] \}}_{O(n)} \\ S[i][2] = j \text{ such that } A[j] < A[i] \text{ and } S[j][1] + 1 = S[i][1] \end{cases}$$

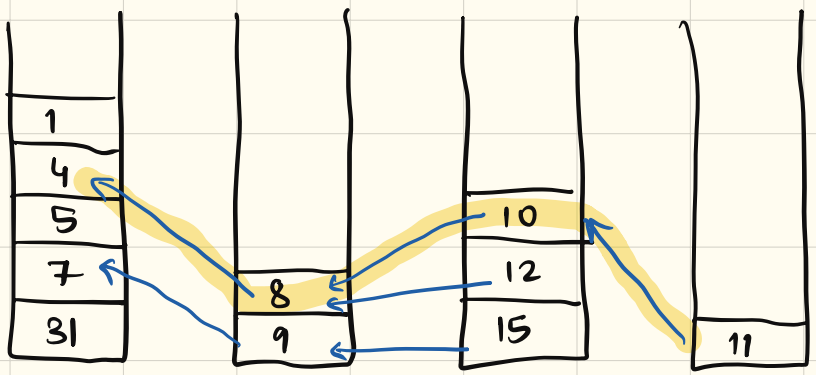
(2n)

$$\text{length of LIS} = \max_{1 \le i \le n} S[i][1]$$

$$\text{Time complexity} = O(n^2)$$

② Greedy algorithm

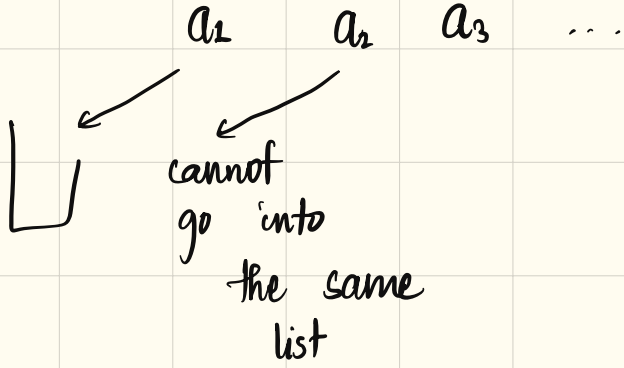
31 7 9 5 15 4 8 1 12 10 11



every time a new stack is made, add a pointer to the top element of stack that created it at that time

Q: Show that length of a longest increasing subsequence is <u>at most</u> the number of stacks / lists.

Simple argument: Pigeonhole principle



Q: Show that the length of a LIS is equal to the number of stacks / lists.

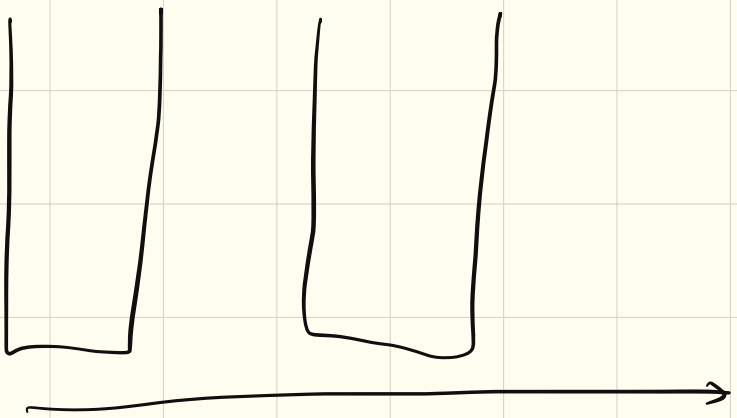
Time complexity ⟶ time to search which stack to enter

$O(n^2)$ ⟶ Better : Binary search

$O(n \log n)$ time.

Erdös - Szekeres Theorem

Given a sequence of $n^2 + 1$ distinct numbers, there is either an increasing subsequence of length $n+1$ or a decreasing subsequence of length $n+1$ or both.



either $n+1$ stacks or more than $n+1$ elements in a stack.

comedy

What is the use of all this in real world?

Erdös ⟶ Ramsay number

Exercises

Q. 0 − 1 knapsack.

n items, every item is a tuple

value ↑ , weight ↗

$\underbrace{(v_1, w_1)}_{\text{item 1}}$, $\underbrace{(v_2, w_2)}_{\text{item 2}}$, ... $\underbrace{(v_n, w_n)}_{\text{item } n}$

Thief who has a bag of capacity W. What is the maximum value of goods possible.

also an input

DP algo with running time $O(n\,W)$.

Input size = $\log v_2 + \dots + \log W_1$

Sorting: n numbers, largest k $\}$ → $n \log k$ input size

↘ Time $O(n^2 \dots)$ → $\log k$ time to compare two numbers

W can be really large

n tuples ⇝ largest k.

$O(n\ W)$ ⟶ pseudo-polynomial.

$O(n \log W)$ ⟶ polynomial

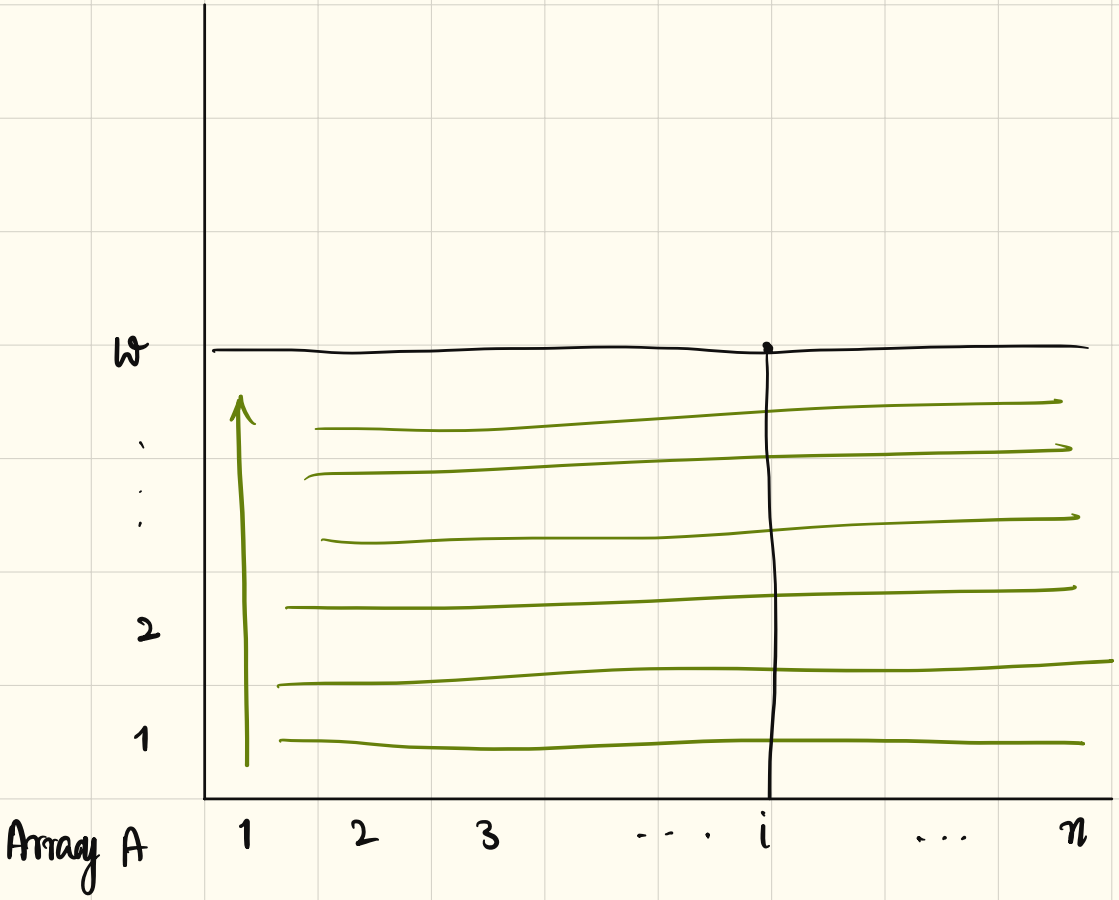
gcd (a, b) ⟶ 1 to a

$a < b$

$\log b$ bits ⟶ exponential

"Dynamic Programming is just drawing tables"

A[n][W]



A[i][W]

The max value that the thief can take home if

(i) capacity of his bag is W

(ii) list of items are

item 1, ... ,

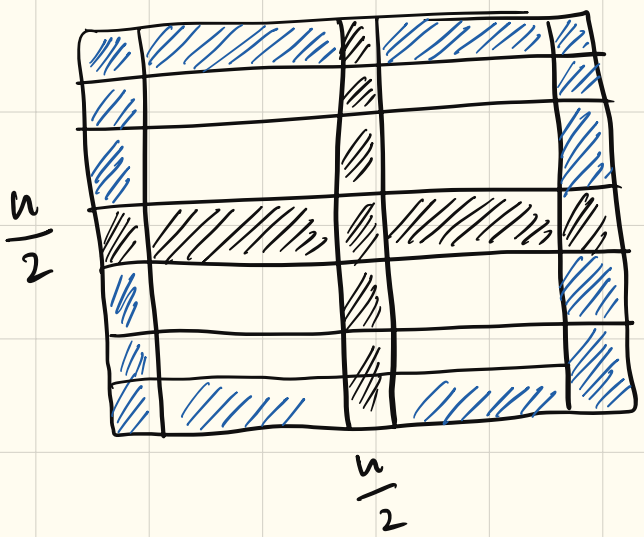
item i

$$A[i][W] = \max \{ \underbrace{A[i-1][w-w_i]}_{\text{when ith item is in the soln}} + v_i \ , \ A[i-1][W] \}$$

20 Mar 2025

Basic Graph Algorithms

Quiz 1 . Q6



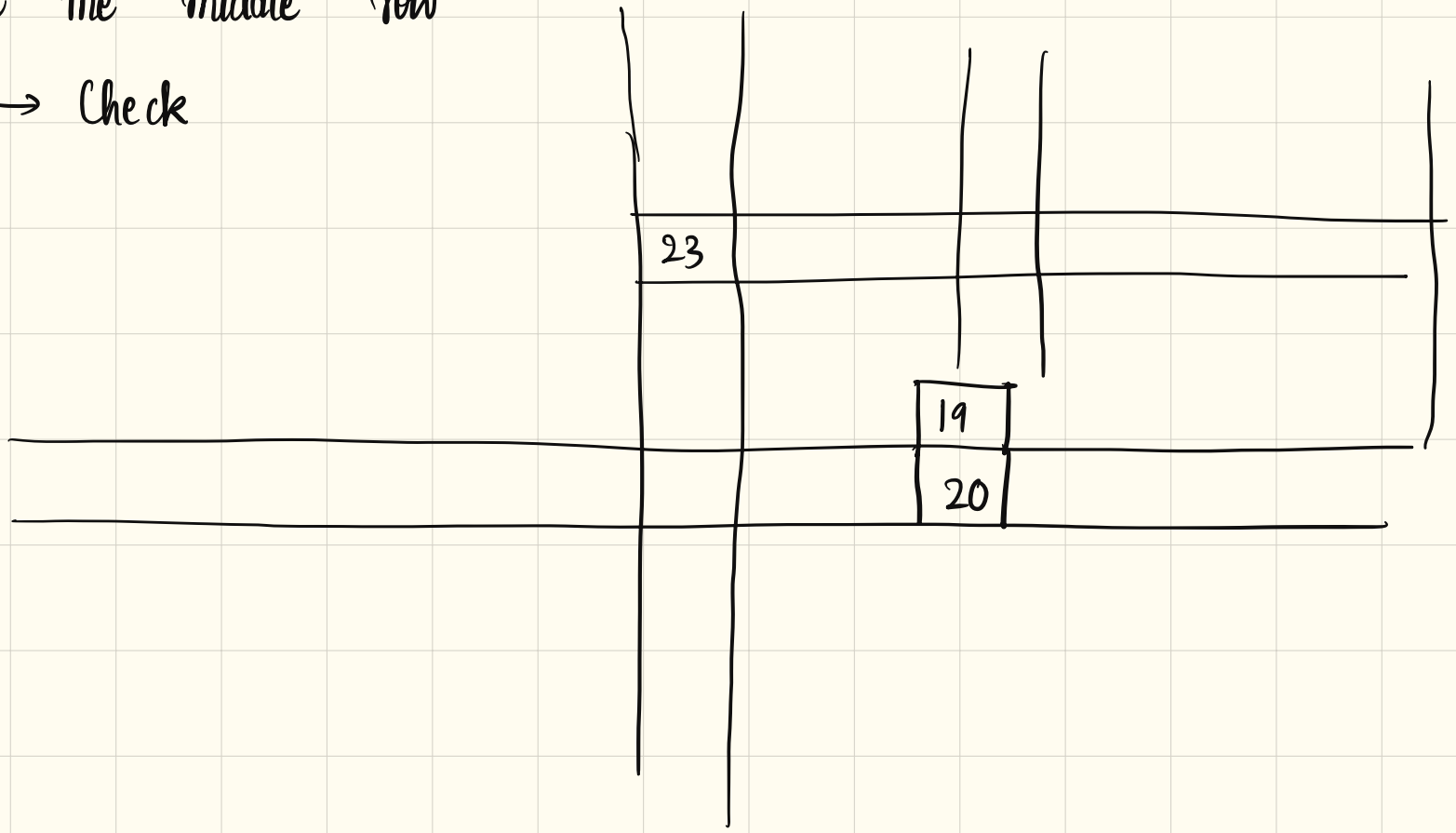
Input : $n \times n$ matrix

Output : local minimum

$T(n) = O(n) + T(n/2)$

Find middle row and column and find the least element in the middle row

→ Check

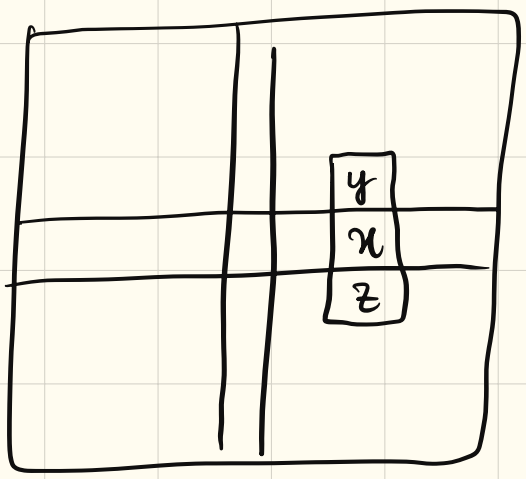


→ Check boundaries

1. Find smallest element x of column / row (first, middle, last)

2. If x is a local minimum, return it.

3. Else let $y < x$

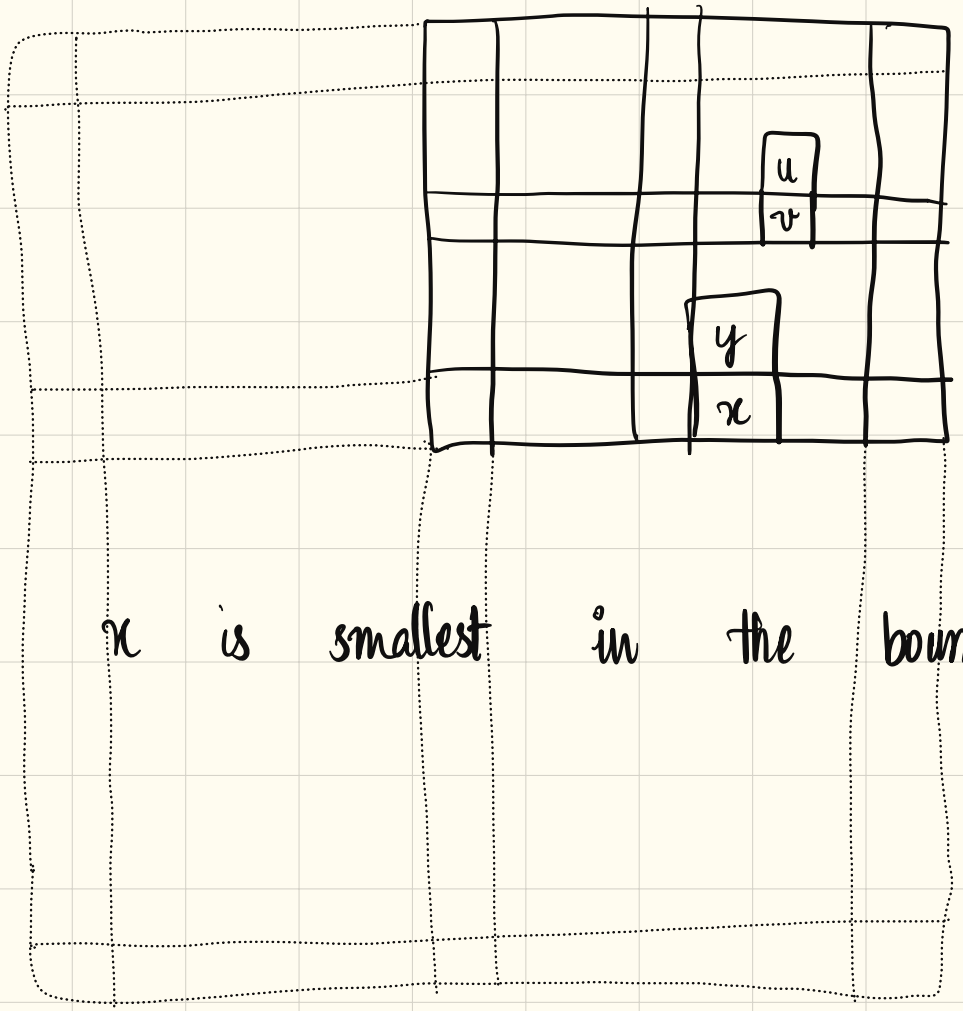


$y < x$

$z < x$

Find internal local minima recursively subject to smallest element of boundary is not a local minimum.

Claim:



$u < v$

$y < x$

x is smallest in the boundary

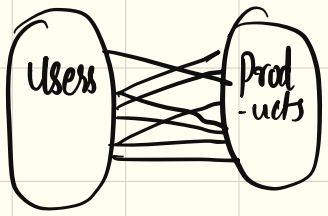
Graph Algorithms

Real-world graph datasets

- Kaggle (See slides)
- Snap

Social networks

Bipartite graphs: amazon



Useful definitions: (see slides)

* path: no repitition of vertices
* walk: repitition allowed.

* eccentricity
* diameter

(??) How much time to convert b/w adjacency list and adjacency matrix ?

Adjacency List

- $|list(v)| = deg(v)$

$$\sum_{v \in V} deg(v) = 2|E|$$

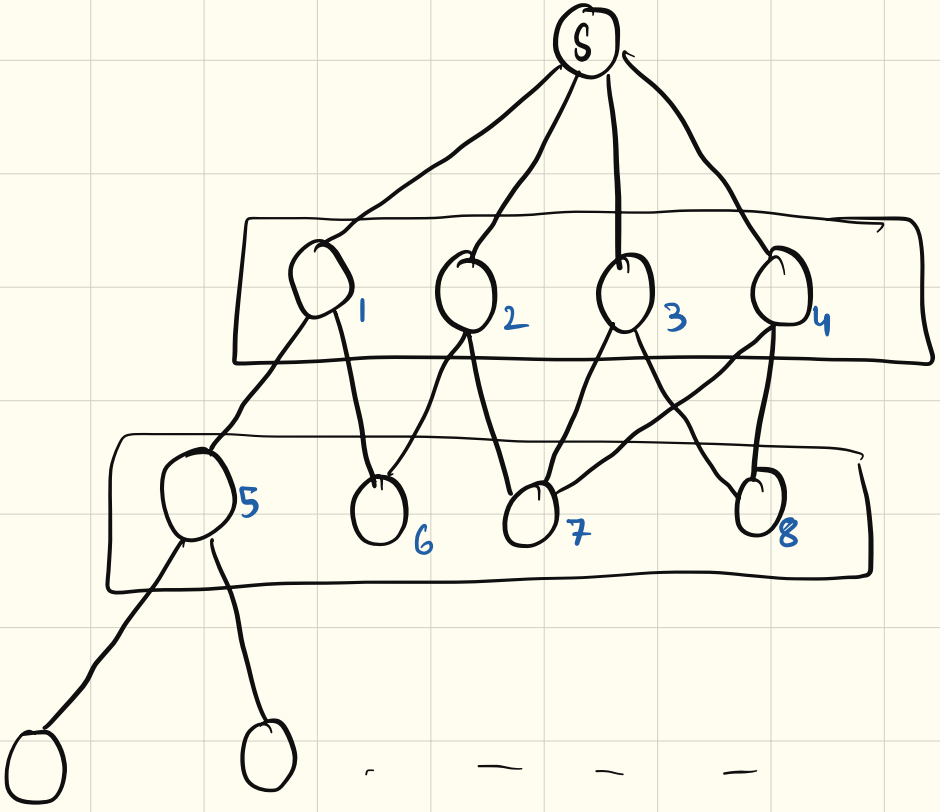
$$O(|V| + |E|)$$

- Accessing the neighbours ✓

Adjacency Matrix

- $O(n^2)$
- Checking for adjacencies ✓

Breadth-first search



$d[v] = 1$

$d[v] = 2$

$d[v] = 3$

Q : {s}

↘ explore s

Q : {1, 2, 3, 4}

explore 1, 2, 3, 4

↘ explore 1

Q : {2, 3, 4, 5, 6}

See slides

undiscovered (white)

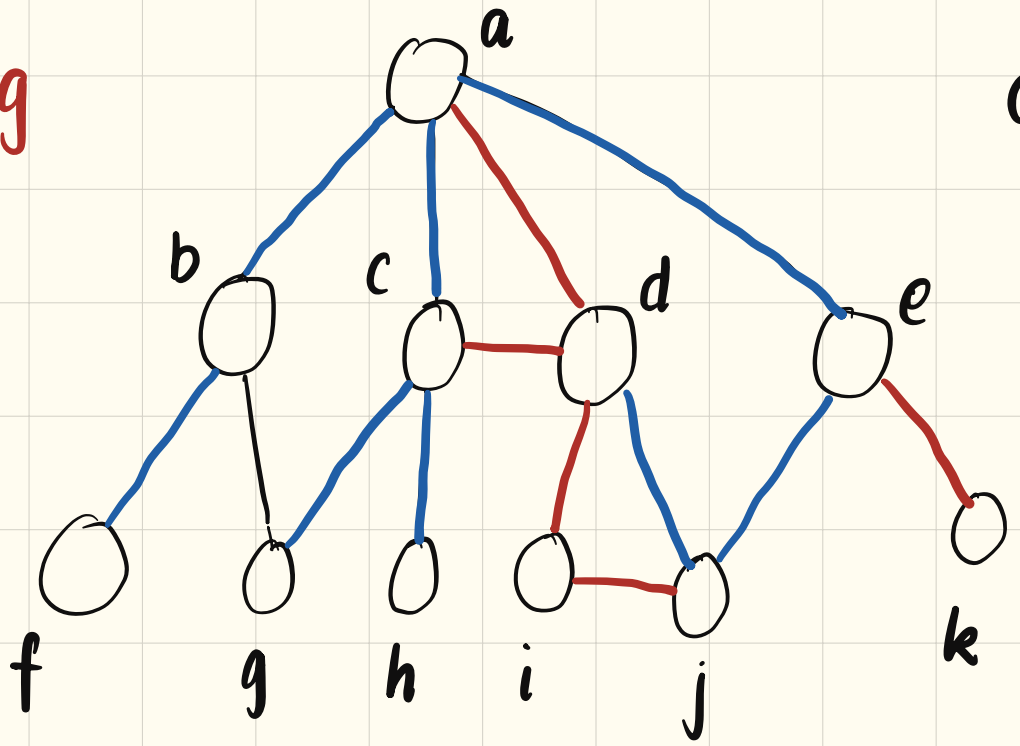
discovered (gray)

explored (black / yellow in slides)

- Time Complexity : $O(|V| + |E|)$
- Algorithm
- Printing the path

Alternating Paths (shortest)

Path may not exist



a → d → j

BFS modification: when exploring u

* store ⟨colour, vertex⟩ pair in queue ($\rightarrow^{(u,v)}$)
* explore only the opposite colour after dequeueing.