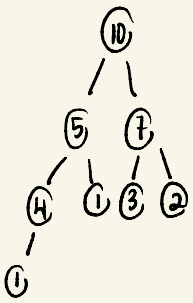
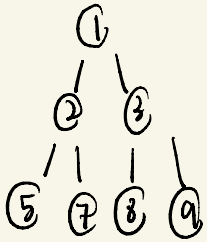
FIT2004
WK5
TUTORIAL

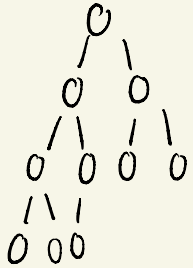
max heap



min heap



complete heap

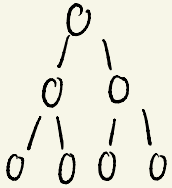


fill top → bottom,

✓ left → right

✓ not null / empty

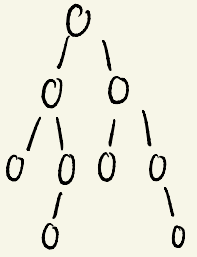
full heap



fill top → bottom,

✓ left → right

incomplete heap



fill top → bottom,

X left → right

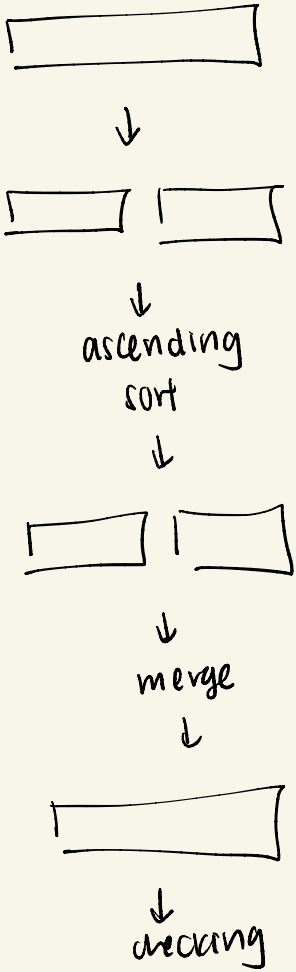
✓ null / empty

merge sort

↳ divide to 2 parts
  ↳ or more smaller parts

↳ solve and/or sort the smaller parts
  ↳ eg. ascending / descending

↳ once done, merge 2 array tgt



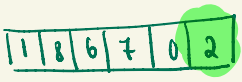
quicksort

↳ divide & conquer

① choose any elem as pivot

② divide main array to smaller arrays

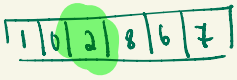
③ compare and sort all elem with pivot
- ↳ left : smaller
- ↳ right : bigger

eg. 

← original array

choose pivot
- ↳ eg. rightmost elem : 2

① compare all elem with pivot & place pivot within array so that
- right elem > pivot
- left elem < pivot



② choose pivot for left & right and continue to arrange

③ choose pivot for left & right and continue to arrange

★ SORTED

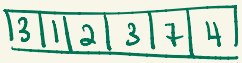
counting sort

① find max value in array

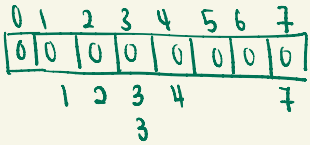
② initialize and create new array of size max+1 with all 0

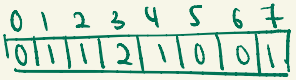
③ iterate through original array while increment the count of each element in count array
- ↳ original array no change yet

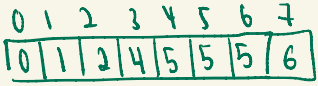
④ find index of each elem of original array in count array
- ↳ place array elem in correct position
- ↳ decrement count in count array

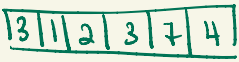
eg. 

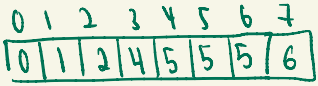
max : 7

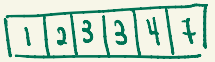






original 

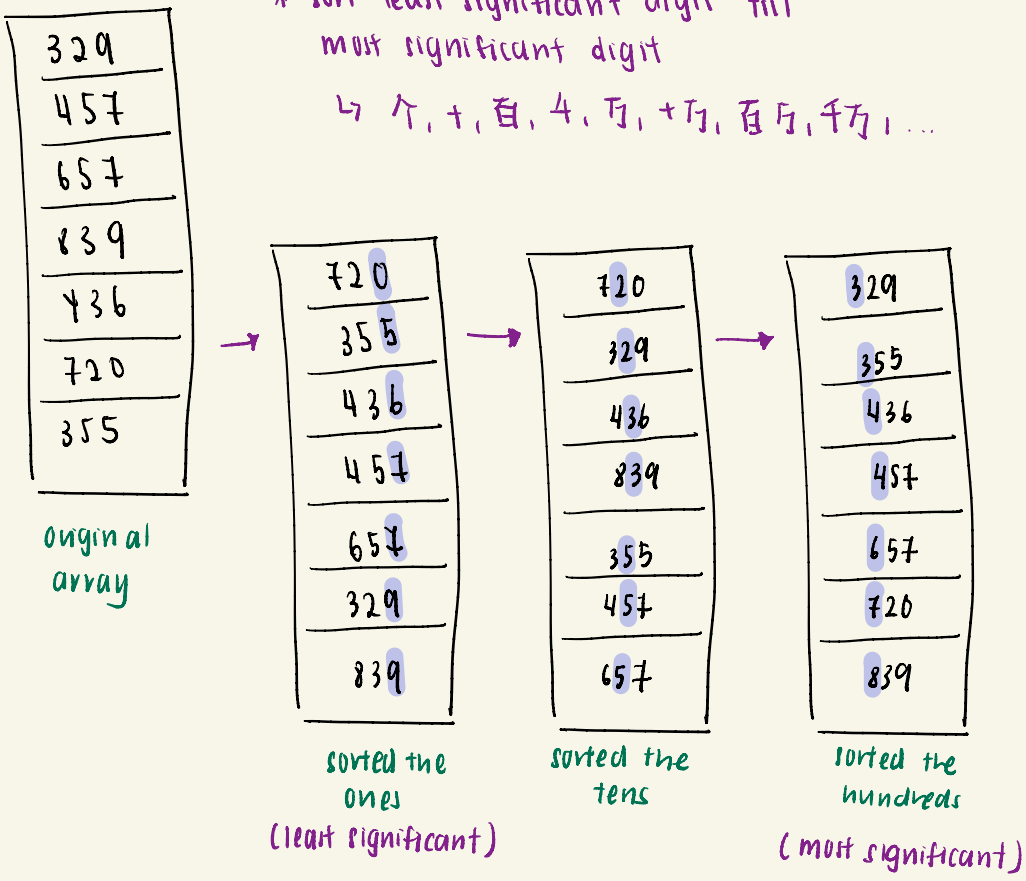
count (cumulative) 

output 

radix sort

- numbers in array : 329, 457, 657, 839, 436, 720, 355

* sort least significant digit till most significant digit
  ↳ 个, 十, 百, 千, 万, 十万, 百万, 千万, ...



quickselect

↳ selection algorithm

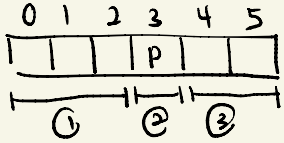
↳ find k-th smallest element from unordered list

↳ @ see quick sort (sorting algorithm)

① select a pivot
   ↳ random / first / last / median

② sort elements to left right
   ↳ compare each elem with pivot

③ k value if > pivot index, search right
   k value if < pivot index, search left
   k value if = pivot index, found, return



① if k = 0/1/2
② if k = 3
③ if k = 4/5

eg. array = | 7 | 10 | 1 | 4 | 20 | 15 |

① k = 3
   ↳ output = 7

② k = 4
   ↳ output = 10

② k = 6
   ↳ output = 20

median of medians

↳ approximate median selection algorithm

↳ divide and conquer

↳ find kth smallest num in unsorted array

① divide array to smaller array

② find median in the smaller array

③ put all median to a list, find median of the list

↳ median of the list = pivot

** median of list = pivot

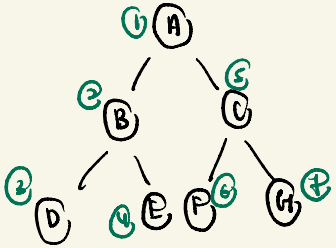
④ compare array elem with pivot } quickselect

↳ < pivot = left

↳ > pivot = right

Depth - First Search (DFS)

- start : root node
- movement : adjacent unmarked node
- aim : visit and mark unmarked nodes untill no unmarked nodes exist
- output : print nodes in path

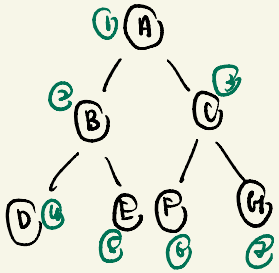


DFS path : ABDECFG

" left to right "

category → visited / discovered
category → not visited / discovered

breadth - first search (BFS)



"layer by layer, up to down"

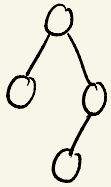
BFS : ABCDEFG

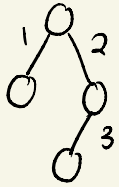
category → visited / discovered
→ not visited / discovered

- start : root node
- movement : unmarked node layered
- aim : visit and mark unmarked nodes untill no unmarked nodes exist
- output : print nodes in path

graph

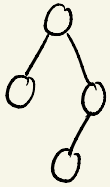
unweighted vs weighted

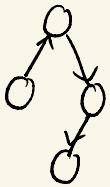


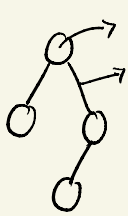


* cycle in graphs:
  begins and ends in the same vertex

undirected vs directed







* with distance & node, can construct the sequence of shortest path from start → end

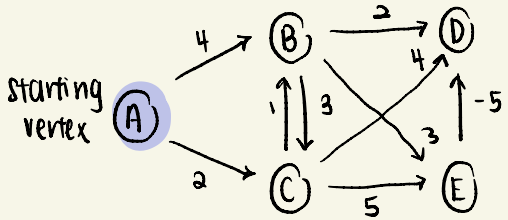
shortest path

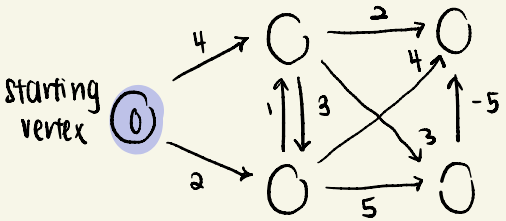
2 array
- → distance
- → vertex / node that proceeds to another vertex / node of a specific distance saved in the other distance array

Bellman-Ford algorithm

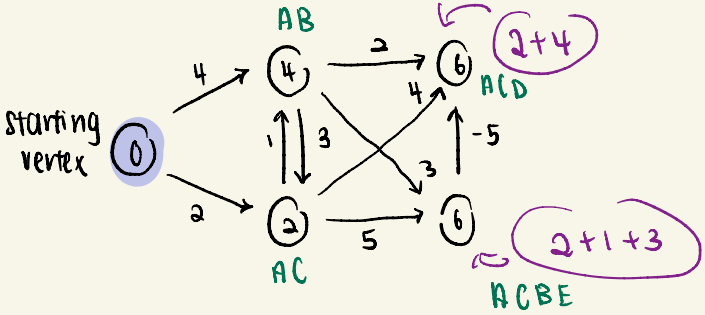
- solve single-source shortest path in weighted graph
- calculate shortest path in bottom-up manner
- ✓ negative weights, check negative weight cycles
- goes through each edge in every iteration

① weighted graph, choose a starting vertex

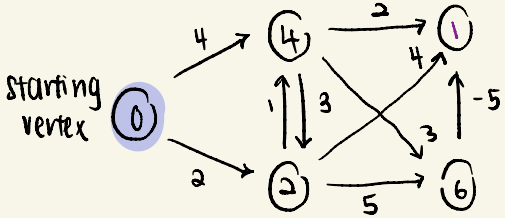




② visit each edge



③ adjustments



ACBED

$= 2+1+3+(-5)$

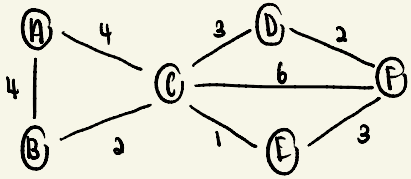
$= 6-5$

$= 1$ #

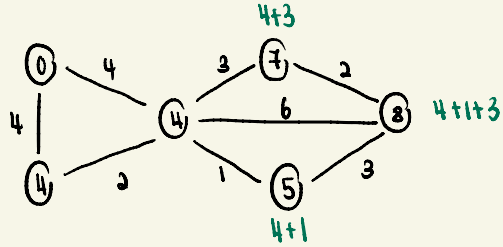
④ check negative cycle

Dijkstra's algorithm

- X negative weight graph
- shortest path in weighted graph from source to target
- √ use weight of edges
- √ greedy algorithm



starting vertex = A



* repeatedly visit vertex until all vertex are visited

* complexity
  1. time → O(E log(V))
  2. space → O(V)

Floyd-Warshall algorithm

X negative weighted, negative cycle graph

V both directed, undirected

- shortest path in weighted graph
- dynamic programming approach

complexity

① time : $O(n^3)$

② space : $O(n^2)$

matrix method: empty matrix with V x V size

* V = vertex

fill in current version of graph to matrix

iterate through v_1 to v_n with other v in the mid to find paths

↳ all possible paths with nth number of v in the mid for v_1 to regin v_n

kahn's algorithm & topological sorting

- ordering vertices of directed graph
- directed edge (u,v), u vertex comes before v vertex
- helps determine the order
- possible for directed acyclic graphs (DAG)
  - ↳ x directed cycles
    - ↳ if √ cycle, x topological ordering
      - ↳ circular dependency

↳ iteration

① empty ordering array

② calculate incoming edges to specific vertex

③ queue all vertex with 0 incoming edges (start vertex)

④ while queue != 0,

```
deque vertex v
emptyarray.append(v)
for i in v.neighbour:
    i.incomingEdge -= 1
    if i.incomingEdge == 0:
        queue.enqueue(i)
```

* ordering array = valid topological order

** if algorithm not finish, means there is a cycle

*** time complexity = O(V+E)

DFS

① empty ordering array

② x incoming edges = start
  - ↳ mark 0, visited

③ adjacent vertex visited + add current vertex to array

④ continue until all vertex is visited

* ordering list / array = valid topological ordering

** if still have unvisited vertex = cycle present

graph connectivity

pg 149

① incremental connectivity problem

↳ connect(u,v) & link(u,v)

↳ check connections

↳ add edge

prim's algorithm

↳ greedy algorithm

↳ find minimum spanning tree (MST) of weighted, undirected graph
  ↳ connect all vertices with min weight

① choose a starting point

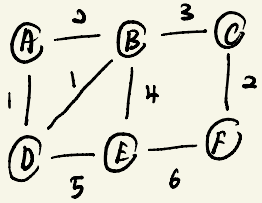
② branch out to find unvisited & unconnected vertex
  ↳ connect to MST

③ until all vertices is connected, it is not complete

④ once completed, the MST is min weight edge chosen to connect unvisited node to visited node in MST

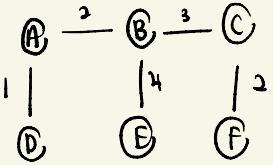
** 100% min spanning tree
  ↳ lowest weight always chosen if can
  ↳ connect all vertex to MST

eg.



starting vertex = A

MST:



weight: 2 + 3 + 1 + 4 + 2
= 12 #

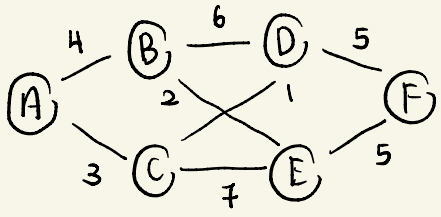
kruskal's algorithm

↳ greedy algorithm

↳ find minimum spanning tree (MST) of weighted, undirected graph
  ↳ connect all vertices with min weight

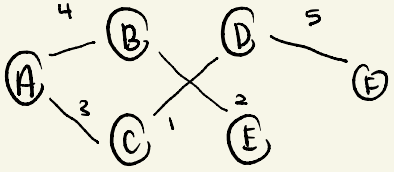
1. sort all edges in non-decreasing order of weight
2. empty array (store MST)
3. iterate through sorted edges smallest → largest
4. if add to MST = ✓ cycle, don't add
   else add
5. repeat until all vertex is visited and have (V-1) edges

KRUSKAL VS PRIM

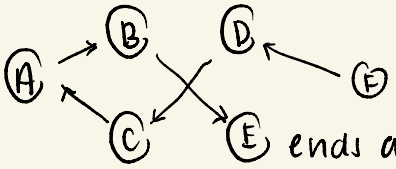


sort by order :

CD – 1
BE – 2
AC – 3
AB – 4
DF – 5
EF – 5
BD – 6



check for cycle

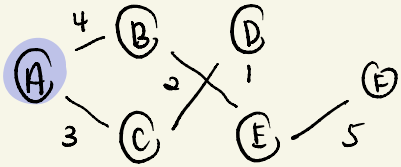


ends at E or F, no cycle exist

total weight : 1 + 2 + 3 + 4 + 5

= 15 #

choose starting point (A)



✓ all vertices visited

✓ shortest edge chosen

weight

= 1 + 2 + 3 + 4 + 5

= 15 #

same !

DYNAMIC PROGRAMMING

√ overlapping subproblems
- ↳ solution to subproblems needed multiple times
  - ↳ store in memorization table
    - ↳ access & use immediately
    - ↳ avoid repeated / redundant calculation
    - ↳ efficiently solve larger problems

* find base case of subproblem
- ↳ solve & store solution in matrix/table
- ↳ eg. fibonacci
  - ↳ solution = sum of 2 previous numbers
  - ↳ computed, stored in table
    - ↳ retrieve for next calculation with index
    - ↳ X count / calculate again
- ↳ eg. knapsack
  - ↳ select item with certain values & weight to maximise total value within limit
    - ↳ matrix to store info (item x weight)
      - ↳ filled iteratively on possible weight
      - ↳ get max value within limit from possible values
        - ↳ previous item & weight

SORTING ALGORITHM

selection sort	$O(1)$	$O(n^2)$	$O(n^2)$
insertion sort	$O(1)$	$O(n)$	$O(n^2)$
merge sort	$O(n)$	$O(n \log n)$	$O(n \log n)$
heap sort	$O(1)$	$O(n \log n)$	$O(n \log n)$
quick sort	$O(\log n)$	$O(n \log n)$	$O(n^2)$
counting sort	$O(k)$	$O(n+k)$	$O(n+k)$
radix sort	$O(n+k)$	$O(nk)$	$O(nk)$

time complexity : Best, WORST space complexity : AUX

SELECTION ALGORITHM

quick select

median of medians

} time and space complexity depends on pivot choice and implementation.

SORTING ALGORITHM

selection sort ----- not stable ----- in-place
insertion sort ---------- stable ----- in-place
merge sort ---------- stable ---- not in-place
heap sort ----- not stable ----- in-place
quick sort ------ not stable --- in-place
counting sort ------ stable --- not in-place
radix sort ------ stable --- not in-place

IMPORTANT POINTS

↳ heap sort's sorting is in-place
↳ quick sort's sorting is in-place
BUT pivot choice will affect stability

TOP-DOWN & BOTTOM-UP DYNAMIC PROGRAMMING

TOP-down dynamic programming

- "memorization"
  - break big problems to small problems to be solved recursively
  - check if the problem is solved b4 recomputation
    - solved = obtain solution from memory
    - not solved = computation to solve and store in memory
  - avoid redundant computation

BOTTOM-up dynamic programming

- "tabulation"
  - solve small problems then big ones, solution store in memory
    - check if the problem is solved b4 recomputation
      - solved = obtain solution from memory
      - not solved = computation to solve and store in memory
    - iteration

HASHING & HASHTABLES

Hashtables (Hash map)
- efficient storage
- efficient retrieval of key-value pairs

Hashing
- mapping key to specific array using hash function
- input : key
  output : hash code
  ↳ hashtable index
- control hashtable size
  = control memory usage

Open addressing (probing)
- collision resolution strategy
- search alternate empty space present in hashtable to store key
- methods :
  - linear probing
  - quadratic probing
  - double hashing

Chaining
- collision resolution strategy
- maintain link list at each position in the hashtable
  ↳ same hash code
  = store in link list in same hash table position

hash function

- good ? minimise collision probability
- uniformly distribute keys across a range of hash values
  - ↳ bad if keys concentrated within small range of hash values
- efficiently handles collision if collisions has occurred

hashing integers

- convert interger keys to indices for storage in hashtable using hash function
- method : ① divisional
  ② multiplicative

① divisional
- modulo of key with table size

② multiplicative
- constant 'A' to compute fraitional part of 'k*A' while scaling by 'm'

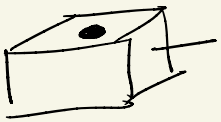
hashing strings

- convert string keys to indices for storage in hashtable using hash function
- method : ① polynomial

① polynomial
- utilize Horner's method to evaluate efficiency of polynomial hashing
- constant time calculation of adjacent substring hashing

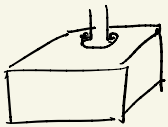
Universal hashing

- approximate totally random hashing while still being computationally efficient
- select random hash functions from universal family of hash functions
- introduce randomness into hash function while being efficient and practical
  - ↳ ensure lower probability that 2 distinct keys hashing will be hashed to the same slot in the hashtable
- helps distribute keys evenly across hashtable



a lot of hash functions

randomly choose 1 hash function



random hash function to be used for hashing 'A'

BALANCED BINARY SEARCH TREES

AVL Trees

- height of left & right subtrees of any node differ by at most one
- remain balanced after insertion & deletion
  - ↳ maintain efficient search, insertion, deletion operation

Definitions

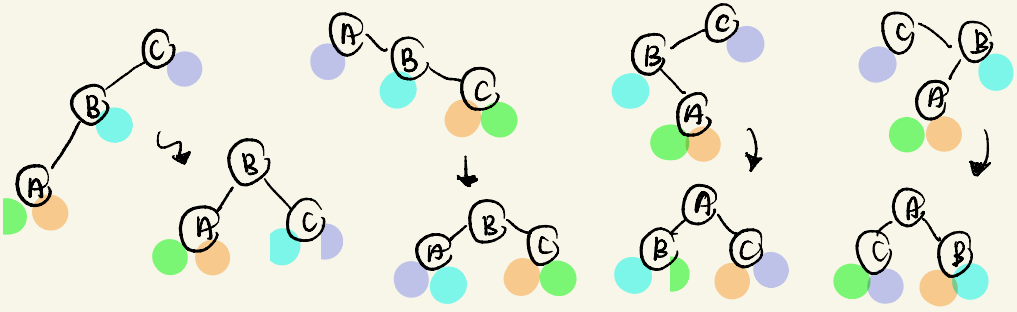
- height : length of longest path
- empty tree height = -1
- left subtree − right subtree = balance factor

absolute balance factor of AVL <= 1

rebalancing

- rotation when insertion / deletion causes difference in node / leaf

① L-L ② R-R ③ R-L ④ L-R



NETWORK FLOW

Finding optimal flow of certain quantity from source node to sink node while taking into consideration on edge's capacity constraint

Ford-fulkerson algorithm

- find maximum flow in a flow network
- repeatedly find paths from source to target while increasing flow

Residual network

- remaining capacity of edges in flow network after flow was pushed through
- check if need increase or decrease flow

augmenting path

↳ path from source to target in residual network where all edges' remaining capacity > 0

↳ gradually increase overall flow

dfs

↳ find augmenting path

↳ explore residual network until reach sink

min-cut max-flow theorum

↳ min flow = min cut capacity

PREFIX TRIES

- retrieval tree
  - ↳ store strings
  - ↳ organised like graph
- 1 root, n child node
  - ↳ weight stored in node
  - ↳ end with $ terminal for each string
- methods:
  1. array
  2. AVL
  3. hash tables

SUFFIX TREES

reduce space complexity

- efficiently store all suffix of given string
- pattern matching problem
  - ↳ find pattern occurrences
  - ↳ eg. text string, T
    pattern, P
    Q: find all occurrence of P as a substring of T

solution

- insert all suffix of text string into prefix tree.
- find / check pattern is a prefix in the tree

SUFFIX ARRAY

- efficiently store and process suffixes of a given string

↓

store starting positions of all the suffix of a given string in order → lexicographical order

The binary search on the suffix array is used to locate the range where the pattern might exist. At each iteration, the middle index "mid" is calculated and the suffix starting at SA[mid] is compared to the pattern P. The position of the first occurrence is stored in the variable "begin". The last occurrence is found at positions SA[begin] through SA[end].

binary search is used to find first and last occurrance of pattern in text

time complexity : $O(m \log(n))$

2022 Semester One (June 2022) Examination Period

Faculty of Information Technology

EXAM CODES:	FIT2004
TITLE OF PAPER:	Algorithms and data structures
EXAM DURATION:	2 hours 10 mins

Rules

During your eExam, you must not have in your possession any item/material that has not been authorised for your exam. This includes books, notes, paper, electronic device/s, smart watch/device, or writing on any part of your body. Authorised items are listed above. Items/materials on your device, desk, chair, in your clothing or otherwise on your person will be deemed to be in your possession. Mobile phones must be switched off and placed face-down on your desk during your exam attempt.

You must not retain, copy, memorise or note down any exam content for personal use or to share with any other person by any means during or following your exam. You are not allowed to copy/paste text to or from external sources unless this has been authorised by your Chief Examiner.

You must comply with any instructions given to you by Monash exam staff.

As a student, and under Monash University's Student Academic Integrity procedure, you must undertake all your assessments with honesty and integrity. You must not allow anyone else to do work for you and you must not do any work for others. You must not contact, or attempt to contact, another person in an attempt to gain unfair advantage during your assessment. Assessors may take reasonable steps to check that your work displays the expected standards of academic integrity.

Failure to comply with the above instructions, or attempting to cheat or cheating in an assessment may constitute a breach of instructions under regulation 23 of the Monash University (Academic Board) Regulations or may constitute an act of academic misconduct under Part 7 of the Monash University (Council) Regulations.

Authorised Materials

CALCULATORS	☐ YES	☑ NO
DICTIONARIES	☐ YES	☑ NO
NOTES	☐ YES	☑ NO
WORKING SHEETS	☑ YES	☐ NO
PERMITTED ITEM	☐ YES	☑ NO

if yes, items permitted are:

Instructions

- This is a **closed book** exam with Specifically permitted items.
- Please **attempt as many questions as possible**.
- **Read each question carefully**. The marks associated with a question is not an indicator of the question difficulty or solution length.
- For algorithms and pseudocode questions, you are allowed to use either:

1. Structured indented text structure.
2. Numbered list.

Best wishes and all the best!

Instructions

Information

You can review your exam instructions by clicking the 'Show Instructions' button above.

Correctness and Complexity

Question 1

2 Marks

For constants *b* and *c*, consider the recurrence relation given by:

- T(n) = b, if n=1
- T(n) = 2 * T(n/2) + c * n^3, if n>1

Which of the following statements is true?

$T(n) = 2T(n/2) + cn^3$

$a = 2$, $b = 2$, $k = 3$, $p = 0$

$\log_b(a) = \log_2(2) = 1$

$1 < 3$

$p = 0 > -1$

$O(n^3 \times \log_0(n)) = O(n^3)$ #

Select one:

- a. T(n) = Θ(n^3 * log n)
- b. T(n) = Θ(n^4)
- c. T(n) = Θ(n^3) ✓
- d. T(n) = Θ(n^6 * log n)
- e. T(n) = Θ(n^3 * log n * log n * log n)

Question 2

4 Marks

Consider the following algorithm, which returns `True` if and only if `m` is a factor of `sum(L)`, where `L` is a list of integers.

```
def sum_factor(L, m):
    """
    Input : A list L of integers, and an integer m
    Output: True if m is a factor of sum(L), and False otherwise.

    For example:
    >>> sum_factor([3, 1, 2, 6], 3)
    True
    >>> sum_factor([4, 1, 2, 6], 3)
    False
    """

    n = len(L)
    s = 0

    for i in range(n):
        s = (s + L[i]) % m

    return s == 0
```

m is a factor of sum(L) where L is a list of integers. in L[0...i-1] and the reason why it is it instead of i is because ith element is not yet processed as the invariant is at the start of the loop, not the end.

(a) Write down a useful invariant for this algorithm.

(b) Show that the invariant you wrote is true on loop entry, and each time the loop runs. skip

(c) Now use your invariant to argue that the algorithm is correct.

the loop terminates when i is equal to the size of list L where i = len(L). m is a factor of sum(L) where L is a list of integers in L[0...(len(L)-1)] which is equivalent to L[0...n-1] where n is len(L). This proves that the invariant is held true for the entire list L.

Sorting

Question 3

Consider an input with ***N*** **elements.**

Please select **all correct statements** regarding **comparison-based** sorting algorithms.

2 Marks

Select one or more:

- [x] a. The best-case complexity of insertion sort is O(N).
- [ ] b. The best-case complexity of merge sort is O(N). O(n log(n))
- [ ] c. Heap sort is stable. not stable
- [ ] d. Selection sort is stable. not stable
- [x] e. The average-case complexity of selection sort is $\theta(N^2)$.

Quickselect and Median of Medians

Question 4

The **Quickselect** algorithm can be run in *O(N)*, provided we have an *O(N)* algorithm to find a **median pivot**.

2 Marks

In your own words, explain why the **median of medians** algorithm can be used for this purpose, even though it only gives an approximation of the median value.

Question 5

You are interviewing a large number of applicants for a job. There are three rounds of interviews, and for the first round, each applicant you interview is assigned a **unique suitability ranking**. These rankings are represented as an **unsorted list of *N* items.** Each item in the list is of the form **(name, rank)**, where **name** is a string representing the applicant's name, and **rank** is a unique floating point value ranging from 0 to 100. Once you've completed all the interviews in the first round, you want a quick way of calculating who should move on to the second and final rounds of interviews.

3 Marks

- The 50% of applicants who received the lowest suitability rankings will not progress any further with the interviews - they will be told they are unsuccessful in their application.
- The 20% of applicants who received the highest suitability rankings can skip the second round of interviews and instead go straight to the third round.
- The remaining 30% of applicants should progress to the second round of interviews.

Describe an efficient algorithm using Quickselect to determine the names of applicants that you know will be proceeding to the second and third round of interviews, after completing the first round. Your algorithm should **run in *O(N)* time**, and you can assume that you have access to a quickselect algorithm which runs in *O(N)* time.

4) median of median improves the worst case complexity of quickselect algorithm by selecting an approximate median element as pivot so the list is partitioned to be more equal and fair whereby the groups smaller than and bigger than have equal size. Resulting in more balanced partitions hence the complexity improves from $O(n^2)$ to $O(n)$. Even if the median selected is not the true median, the median selected is close to the true median value

5) first, calculate the indices for the 50th, 80th and 100th percentile. these will help to identify the participants. quickselect is used to find the kth smallest element for each round. when k=20, the applicant selected will directly proceed to the 3rd round median pivot selected. if the value is less than the pivot, the applicants index more than pivot value partitioned on the right which are the 50% lowest ranking will be eliminated. the leftover applicants which are the 30% will proceed to the 2nd round.

Graph Structure and Traversal

Question 6

adjacency list
↳ space O(V+E)
↳ time → edge exist O(x)
↘ neighbour vertex O(x)

For each of the following operations, determine its **worst-case big-Θ complexity**.

2 Marks

In this question,

adjacency matrix
↳ space O(V²)
↳ time → edge exist O(1)
↘ neighbour vertex O(V)

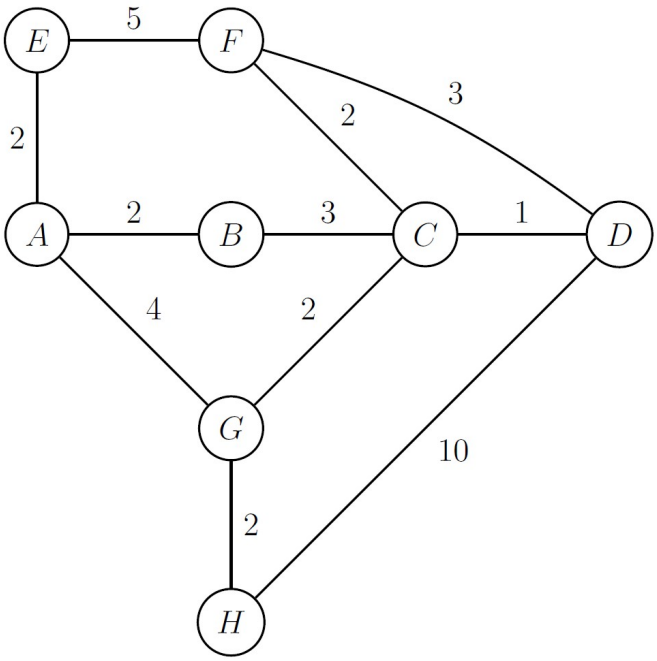
- The graph *G* is a directed weighted graph.
- *V* refers to the number of vertices in the graph.
- *E* refers to the number of edges in the graph.
- *N(A)* refers to the number of neighbors of vertex *A*.

Assume that in the adjacency list representation, the **interior lists are unsorted.**

Time complexity to obtain all incoming edges for vertex *A* in an adjacency matrix representation.	· Θ(log V) · Θ(V) · Θ(V^2) · Θ(N(A)) · Θ(V+E) · Θ(log E) · Θ(1) · Θ(E)
Time complexity to determine if an edge from vertex *A* to vertex *B* exists in an adjacency list representation.	· Θ(log V) · Θ(V) · Θ(V^2) · Θ(N(A)) · Θ(V+E) · Θ(log E) · Θ(1) · Θ(E)
Time complexity to determine if an edge from vertex *A* to vertex *B* exists in an adjacency matrix representation.	· Θ(log V) · Θ(V) · Θ(V^2) · Θ(N(A)) · Θ(V+E) · Θ(log E) · Θ(1) · Θ(E)
Time complexity to obtain all incoming edges for vertex *A* in an adjacency list representation.	· Θ(log V) · Θ(V) · Θ(V^2) · Θ(N(A)) · Θ(V+E) · Θ(log E) · Θ(1) · Θ(E)

Information

Consider the **weighted undirected graph** below and answer the following questions.



Question 7

1 Mark

Perform a **depth-first search** on the graph given above **starting from node A**.

Whenever you have a choice between two nodes, **break ties in ascending alphabetical order.**

Associate each node with the **number for its position in the visiting order**, ie. node A should be associated with *i* if A is *i*-th visited node.

For this question, you can ignore the edge weights.

1st visited node	· D · E · A · H · B · F · G · C
2nd visited node	· D · E · A · H · B · F · G · C
3rd visited node	· D · E · A · H · B · F · G · C
4th visited node	· D · E · A · H · B · F · G · C
5th visited node	· D · E · A · H · B · F · G · C
6th visited node	· D · E · A · H · B · F · G · C
7th visited node	· D · E · A · H · B · F · G · C
8th visited node	· D · E · A · H · B · F · G · C

Question 8

1 Mark

Perform a **breadth-first search** on the graph given above **starting from node A**.

Whenever you have a choice between two nodes, **break ties in ascending alphabetical order.**

What is the **maximum height** of the resulting tree from the breadth-first-search you have performed? **Just type the numerical answer.**

For this question, you can ignore the edge weights.

3 reason: A → B → C → D (have 3 edge)

Dynamic Programming

Question 9

skip !

2 Marks

Recall the following problem from the Dynamic Programming studio:

"You are trying to sell to a row of houses. You know the profits which you will earn from selling to each house *1..n*. If you sell to house i, you cannot sell to houses *i-1* or *i+1*. What is the maximum profit you can obtain?".

Consider instead the **variant of the problem** where **if you sell to house *i*, you cannot sell to houses *i-2*, *i-1*, *i+1* or *i+2***.

Suppose that you have the following DP array, where **cell *i* of the DP array contains the maximum profit you can obtain by selling to a valid subset of the first *i* houses (i.e., a valid subset of {1, 2, ..., *i*})**.

i	1	2	3	4	5	6	7	8	9	10	11	12
DP[i]	12	12	35	35	35	65	65	110	112	120	120	120

Using **backtracking**, determine which houses you should sell to in order to **maximise your profit** (i.e determine the optimal solution to the problem that is **valid for any possible house-profit array** that would result in the DP array above).

For this question you must answer in the following format:

Write the indices of the houses that you have chosen, in ascending order, separated only by a single comma with no spaces. For example if the answer were houses 7, 8 and 9, you would write 7,8,9

Shortest Path

Question 10

tracing question

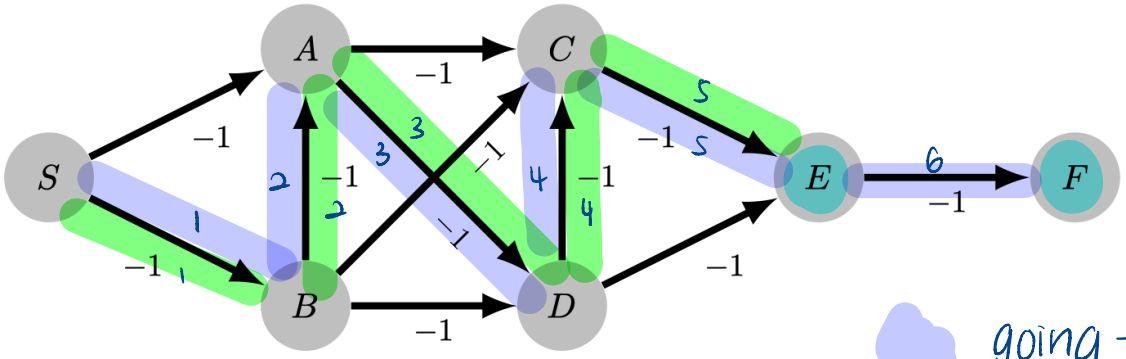
3 Marks

Consider the following version of the **Bellman-Ford algorithm**

Algorithm 54 Bellman-Ford

```
1: function BELLMAN_FORD(G = (V, E), s)
2:     dist[1..n] = ∞
3:     pred[1..n] = null
4:     dist[s] = 0
5:     for k = 1 to n − 1 do
6:         for each edge e in E do
7:             RELAX(e)
8:     return dist[1..n], pred[1..n]
```

and the following directed graph



going to F

going to E

Let S be the source node for the execution of the Bellman-Ford algorithm.

If the edges are relaxed in the following order (S,A), (S,B), (B,A), (B,D), (D,E), (A,D), (A,C), (D,C), (E,F), (B,C), (C,E), what is the **value of dist[E]+dist[F] after the first iteration of the outer loop is finished?**

$5(-1) + 6(-1) = -5 - 6$

$= -11$

Select one:

- a. dist[E]+dist[F]=-7
- b. dist[E]+dist[F]=-9
- c. dist[E]+dist[F]=-11 ✗
- d. dist[E]+dist[F]=-8
- e. dist[E]+dist[F]=∞
- f. dist[E]+dist[F]=-10

E -5
F -4
-9

slowly go through one by one until (GE) then dist[E] + dist[F]

Question 11

Consider a run of the **Floyd-Warshall algorithm** on a **Directed Weighted Graph** *G*.

2 Marks

Algorithm 56 Floyd-Warshall

```
1: function FLOYD_WARSHALL(G = (V, E))
2:     dist[1..n][1..n] = ∞
3:     dist[v][v] = 0 for all vertices v
4:     dist[u][v] = w(u, v) for all edges e = (u, v) in E
5:     for each vertex k = 1 to n do
6:         for each vertex u = 1 to n do
7:             for each vertex v = 1 to n do
8:                 dist[u][v] = min(dist[u][v], dist[u][k] + dist[k][v])
9:     return dist[1..n][1..n]
```

The run produced the following matrix:

	A	B	C	D	E	F	G	H
A	0	14	None	None	20	-9	20	-8
B	13	0	None	None	49	20	None	44
C	16	49	0	20	12	-6	-3	40
D	25	49	None	0	30	47	None	None
E	None	39	None	None	0	40	45	12
F	None	11	31	18	25	0	49	39
G	32	None	None	None	None	47	0	34
H	-10	7	22	None	None	15	21	0

Select the correct observation(s) that can be made from the matrix above.

Note: You **do not need to validate the correctness of the matrix** but just make observations based on the given matrix.

Select one or more:

a. ☐ if the diagonal have less than 0 means negative cycle exist

There is a negative cycle in the graph.

b. ■ look from both vertices if its a number and not none means there is a path exist from vertex 1 to vertex 2

There is a cycle including vertices *A* and *G*.

c. ☐ wording, if they say path instead of edge then may be correct it could be B→A→G→H→C→E ... to get 49, its inconclusive

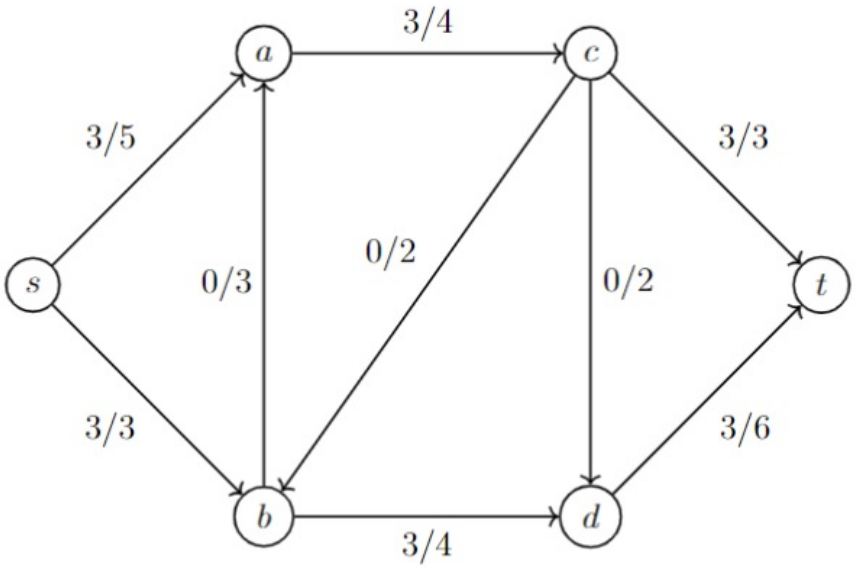
There is an edge from vertex *B* to vertex *E* with a distance of 49.

d. ■ refer to notes in c. 'path' wording is correct.

There is a path from vertex *C* to vertex *F* with a distance of -6.

Information

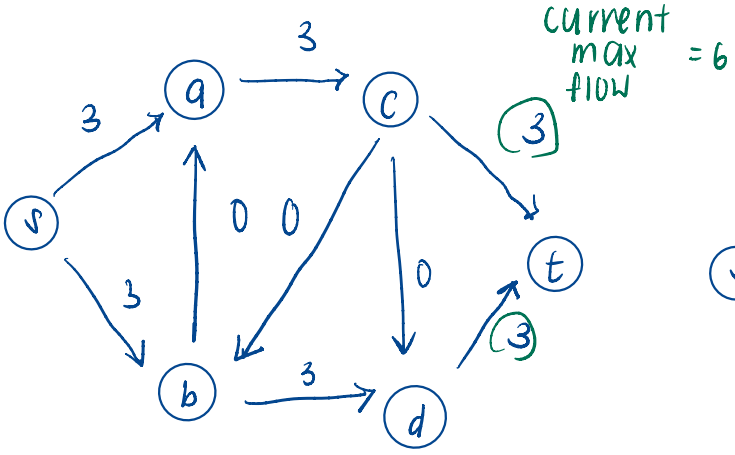
Consider the **flow network** below and answer the following questions.

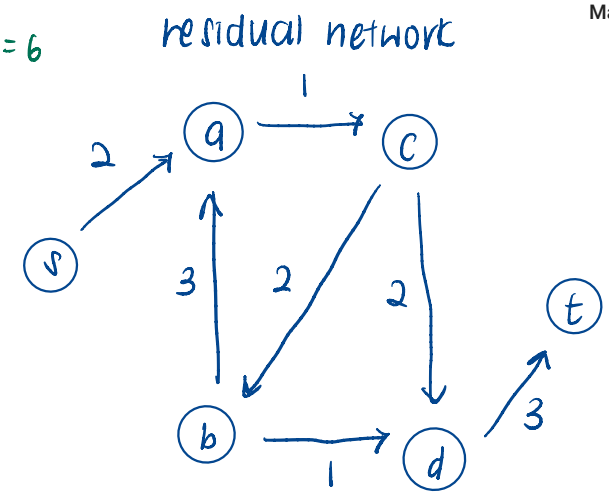


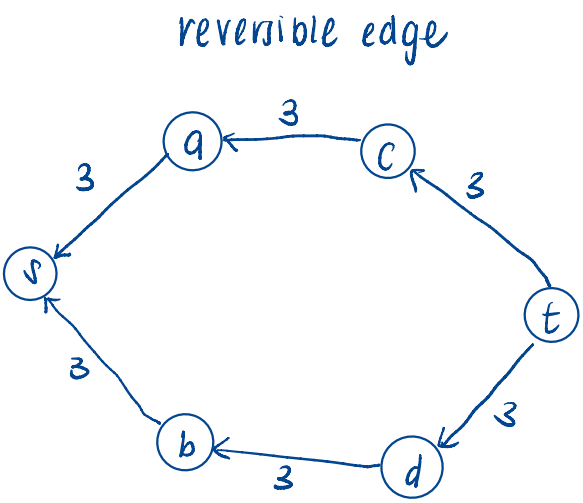
Question 12

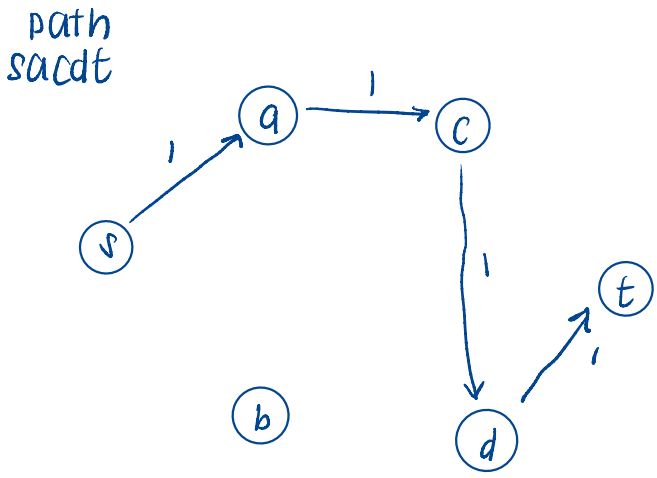
What is the **maximum possible flow** for the given flow network above? Just type the numerical answer.

1 Mark









maximum possible flow = 7 #

Question 13

A cut partitions the vertices into two disjoint sets, S and T, where S contains all the vertices on the source side of the cut, and T contains all the vertices on the sink side of the cut.

1 Mark

Consider the **minimum cut** of the above flow network. Select the **vertices which are in S** from the list of vertices below.

Select one or more:

① use finalised residual network

② get all reachable vertices
↳ have edge from source to another vertex

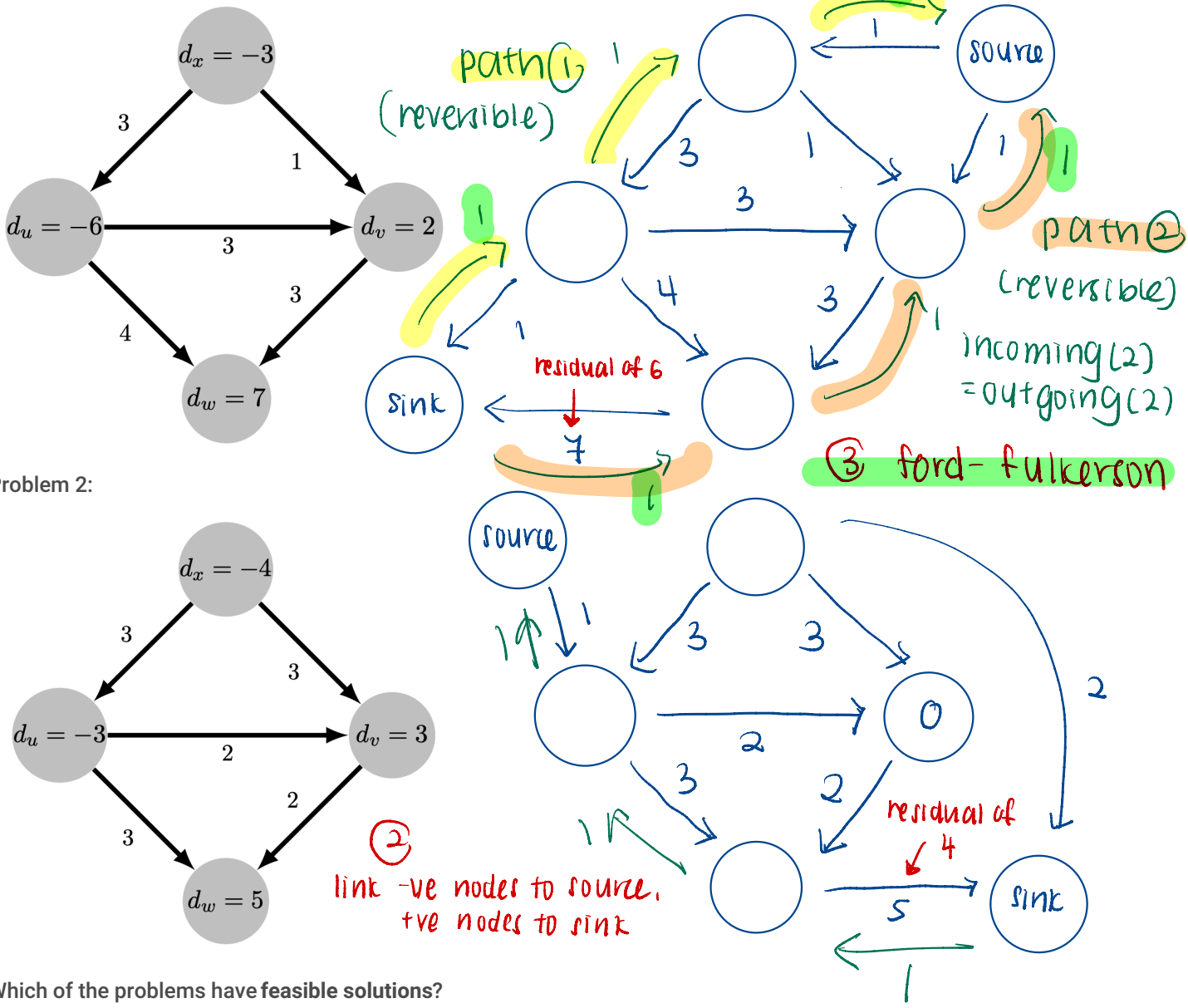
- [x] a. *s*
- [x] b. *a*
- [ ] c. *b*
- [ ] d. *c*
- [ ] e. *d*
- [ ] f. *t*

Question 14

Consider the following two problems of **circulation with demands,** in which the demands are indicated in each vertex, and the capacity in each edge.

3 Marks

Problem 1:



Problem 2:

Which of the problems have **feasible solutions**?

Select one:

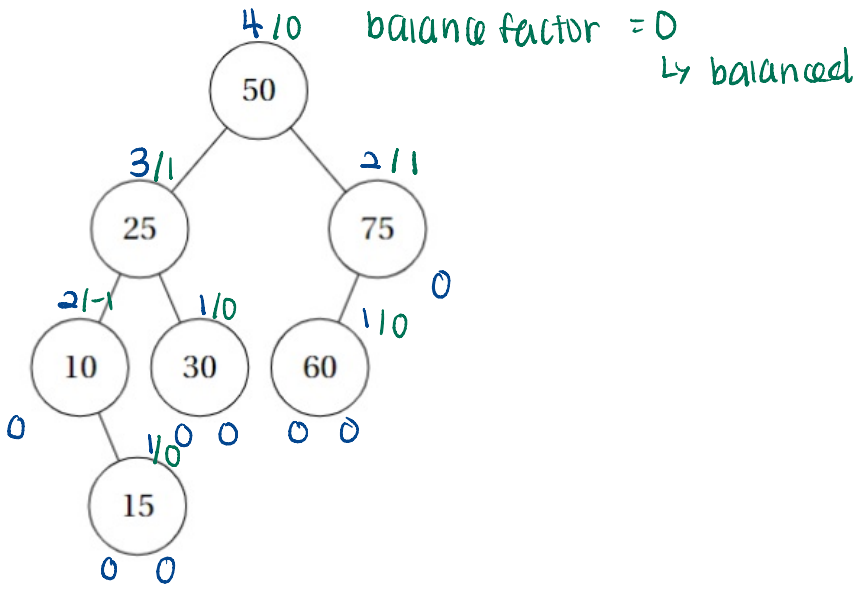
- a. Neither Problem 1 nor Problem 2 has a feasible solution.
- b. Both Problem 1 and Problem 2 have feasible solutions.
- c. Only Problem 1 has a feasible solution.
- d. Only Problem 2 has a feasible solution.

Efficient Lookup Structures

Question 15

2 Marks

Consider the following **AVL tree** below:



Which of the following statements are **true**?

Select one or more:

a.
Just deleting 10 without performing rotations would keep the tree balanced.

b.
Just deleting 30 without performing rotations would keep the tree balanced. X

c.
The AVL tree is unbalanced. X

d.
Just inserting 12 without performing rotations would make the tree unbalanced.

e.
Just inserting 88 without performing rotations would make the tree unbalanced.

Question 16

Consider a **hash table implemented with separate chaining** for collision resolution.

For a hash table with ***N*** items, which of the following data structures would cause the worst-case time complexity of an **insert operation** to be **Θ*(N)*** if the data structure is used to keep the separate chains.

2 Marks

Select one or more:

a. ☐ O(log(n))

Binary search tree

b. ■ need to traverse entire array to check for collision while finding appropriate place to insert new item.

Unsorted array

c. ☐ O(log(n))

AVL tree

d. ■ finding appropriate place O(log(n)) using binary search insert operation require shifting existing elements hence O(n)

Sorted array

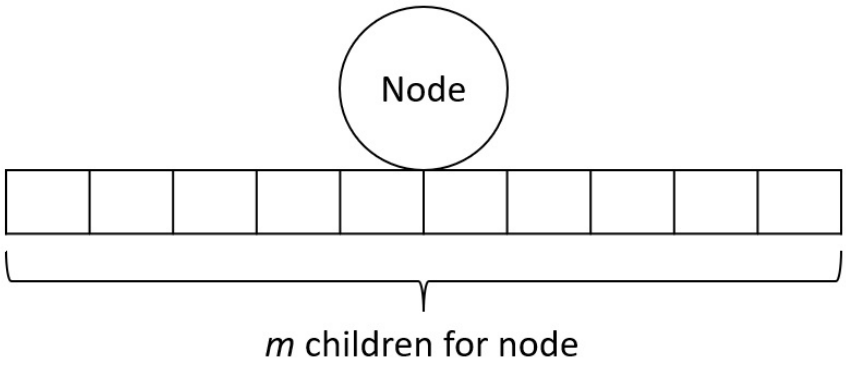
Retrieval Data Structures for Strings

Question 17

Assume that you have an alphabet size of ***M*** **unique characters** and let ***S*** **be a string of length** ***N.***

Thus, you **cannot assume the alphabet size to be** ***O(1)*** as discussed in the lecture. A node implemented with this condition, using an **array of size** ***M*** **for the children** is illustrated below.

2 Marks



For nodes implemented in this way, what is the **worst-case space complexity** in terms of *M* and *N* for string *S* of:

A suffix trie.

- Θ(N log M) · Θ(N M^2) · Θ(NM) · Θ(N)
- Θ(N log N) · Θ(N+M) · Θ(N^2 M) · Θ(M)

A suffix tree, with edges using the **[start,end] or [start,length]** representation.

- Θ(N log M) · Θ(N M^2) · Θ(NM) · Θ(N)
- Θ(N log N) · Θ(N+M) · Θ(N^2 M) · Θ(M)

Question 18

Assume that we are constructing the **suffix array for a string** ***S*** using the **prefix doubling approach.**

We have already sorted the suffixes for string *S* according to their **first 2 characters**, with the **corresponding rank array** shown below:

3 Marks

ID	1	2	3	4	5	6	7	8	9	10	11
Rank	11	10	2	7	2	7	2	9	5	6	1

rank[i+k]
↖ k=2

We are now **sorting on the first 4 characters.**

For the suffixes with **ID5** and **ID7**, **describe in detail** (i.e. all the steps) how will you **compare them on their first 4 characters in** ***O(1)*** and what **the result (order)** from the comparison would be.

When ID5, rank=2 while when ID7, rank=2
since both rank=2, add 2 to ID
ID5+2 = ID7 while ID7+2=ID9
when ID7, rank=2, when ID9, rank=5
since 2<5 so ID7 is before ID9
HENCE, ID5 is before ID7 #

Applications

Question 19

19) minimum spanning tree with using Prim's algorithm.
tree grows on roads based on the connections between key location.
if 2 roads have the same importance, choose to connect road with lower cost first.

Consider the following application problem.

3 Marks

You are the transport minister of your country. The recent disaster devastated the road network connecting all of the key locations. Thus you are to plan out the optimal roads to connect the key locations together, based on the following criteria:

- You want to connect all key locations together; but they do not need to be directly connected
- You are given the importance of directly connecting 2 key locations together, for all key location pairs. The importance is represented by a number for each pair of locations, and the highest this number is, the most important it is to directly connect the 2 key locations.
- You are given the cost of directly connecting 2 key locations together, for all key location pairs.
- For cost savings reasons, you should not build redundant roads: between any pair of locations, there should be only one possible path.
- You want to maximise the importance of connecting all key locations together.
- If there are multiple ways of achieving the maximum value of the importance, then you should pick the one that minimises the cost of the selected roads.

Describe how you would model this problem as a graph and algorithmically solve this problem efficiently using an algorithm you have learnt in the unit.

Question 20

① sort based on starting time in ascending order
② iterate and check if request has been added in new initialised empty array
③ if not added, add it. if added, skip and ignore it

You are the manager of a super computer and receive a set of requests for allocating time frames in that computer.

3 Marks

- The i^{th} request specifies the desired starting time s_i and the desired finishing time f_i.
- A subset of requests is compatible if there is no time overlap between any requests in that subset.

For a set of N requests, give a high-level description of an algorithm with time complexity $O(N * log\ N)$ to choose a compatible subset of maximal size (i.e., you want to accept as many requests as feasible, but you cannot select incompatible requests).

① directed empty graph initialisation where topics and students are nodes
② add source and sink nodes
↳ student node connect to source, topic node connect to sink
↳ max capacity = 5
↳ student node connect to topic node (max capacity 4)

Question 21

③ Ford-Fulkerson method to find maximum flow
↳ maximum flow = max student per unit

You are coordinating a final year project unit.

2 Marks

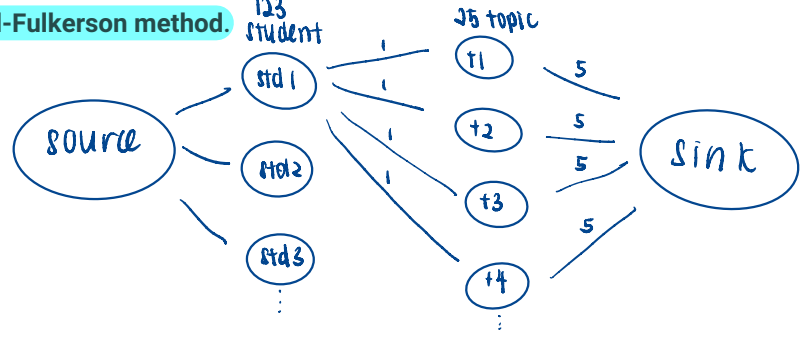
There are a total of 123 students that are enrolled into the unit, and there are 25 topics to choose from:

- Each student is allowed to select up to 4 preferred topics, but they would only be assigned to only 1 topic at the end.
- Each topic can be assigned to at most 5 students.

④ assign student to topic based on maximum flow

You realised that it is not possible for all students to be doing their preferred topics, as some topics are more popular than others. You would however prioritize to satisfy the preferences of as many students as possible.

Describe how you would model this problem as a maximum flow problem, which is then solved using the Ford-Fulkerson method.



Question 22 Bad question - Ian

4 Marks

The engineers in your company believe they have created a phone that is ultra-resistant against falls (perhaps even falls from as high as 150m). To test that hypothesis, your company built two prototypes and asked you to perform the test to determine the maximum height in meters (without considering fractions) that the phone can be dropped without breaking.

There is an unknown integer value $0 \le x \le 150$ such that if the prototype is dropped from heights up to x meters it will not break, while if it is dropped from heights $x+1$ meters or higher it will break. **Your job is to determine x.** If one prototype is broken, it cannot be used in the tests anymore.

As a computer science student, you want to optimize your work. For doing so, you develop an algorithm for determining the heights you should drop the prototype at each iteration (it can take into account the result of the previous iterations). No matter what is the value of $0 \le x \le 150$, your algorithm should be correct and determine the value of x. For an algorithm A and an unknown integer $0 \le x \le 150$, let **Drop(A, x) denote the number of times you need to drop a prototype** if you use algorithm A and x is the threshold. Let Drop(A) denote the worst-case performance of A, which is given by the maximum of Drop(A, x) over all possible $0 \le x \le 150$.

For an optimal deterministic algorithm A that has Drop(A) as small as possible, what is the value of Drop(A)? Just type the numerical answer.

Hint: Since you have two prototypes, it is possible to do better than linear search. But as you only have two prototypes, you need to be very careful if only one prototype remains intact (and thus the search cannot be as efficient as binary search).

ANSWER : 8

The value of Drop(A) for an optimal deterministic algorithm A in this scenario is 8.

To understand why, let's analyze the problem:

We have two prototypes and need to determine the maximum height (x) from which the phone can be dropped without breaking. We want to minimize the number of drops required to find this maximum height.

One way to optimize the process is by using a binary search approach. We start by dropping one prototype from a height of 75 meters. If it breaks, we know the maximum height is below 75 meters, so we drop the second prototype at heights 1 to 74 meters one by one until it breaks. This gives us a total of 75 drops.

If the first prototype doesn't break when dropped from 75 meters, we can conclude that the maximum height is above 75 meters. In this case, we drop the second prototype at heights 76 to 149 meters one by one until it breaks. This gives us a total of 74 drops.

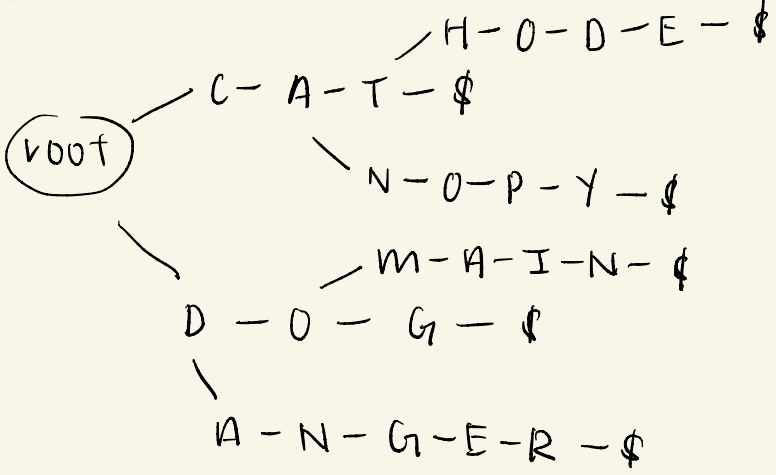
Therefore, in the worst-case scenario, we would need a total of 75 drops (maximum of the two cases). Hence, Drop(A) is 75.

However, it's worth noting that this is a deterministic algorithm, assuming that the phone's resistance remains consistent across prototypes and test drops. In practice, additional factors such as variations in the phone's construction or the testing environment could influence the actual number of drops required.

FIT2004 Tutorial Worksheet

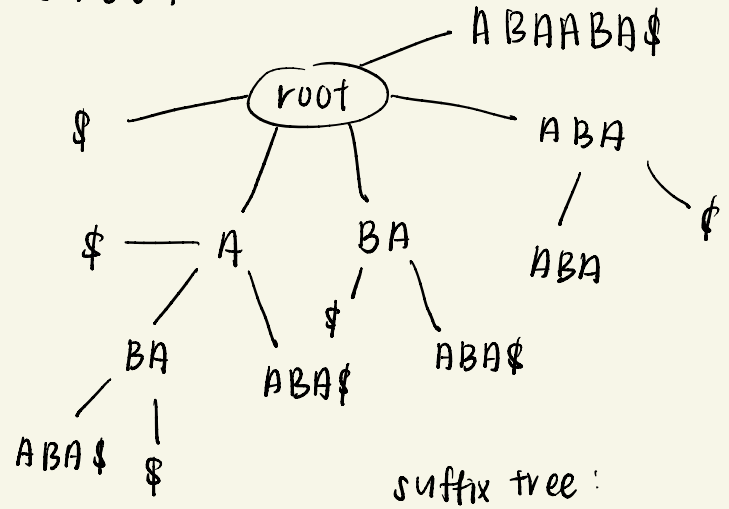
- [ ] week 1
- [ ] week 2
- [ ] week 3
- [ ] week 4
- [ ] week 5
- [ ] week 6
- [ ] week 7
- [ ] week 8
- [ ] week 9
- [ ] week 10
- [ ] week 11
- [ ] week 12
- [ ]
- [ ]

Week 12

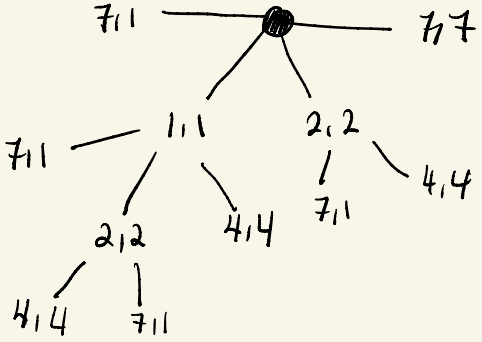
① 

② ABAABA$
1 2 3 4 5 6 7

ABAABA
BAABA
AABA
ABA
BA
A



suffix tree:



(3) sequence of string over constant-size alphabet

initialize count 0
append $ to end of each string
insert string to trie
tranverse trie to count number of $ node

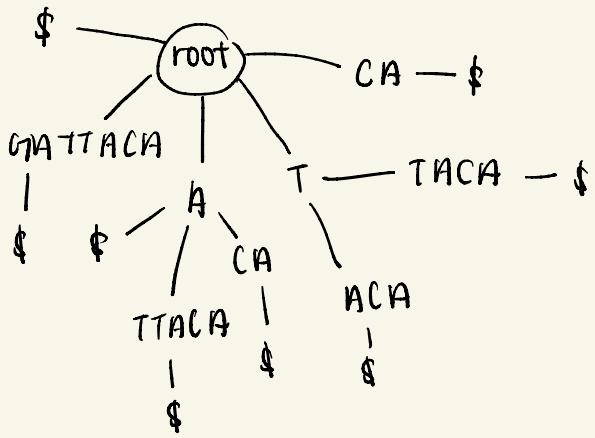
* O(T) to build trie
  O(T) to tranverse trie

** same string = terminate same node
  ↳ distinct only

(4) suffix tree

GATTACA$
1 2 3 4 5 6 7 8

GATTACA
ATTACA
TTACA
TACA
ACA
CA
A



⑤ string s length n as input

count number of distinct substring in O(n)

- initialise count = 0
- initialise an empty tree
- add terminal, $, to end of string s
  - ↳ don't count $ node (else will take longer)
  - ↳ just count nodes
- count total character in the string
- look up parent node indices at current node
  - ↳ determine length

* even one character can be a substring
HENCE, string s length = no. of substring

⑥ longest common substring problem

input : s1 & s2 (2 strings)

output : longest string that's substring of s1 & s2

O(n+m) time complexity using suffix tree

- initialise count = 0
- initialise an empty tree
- count total character in both of the string, s1 & s2

- modify insertion to have node counter
  - ↳ initialise counter = 0
  - ↳ insert each node => counter += 1
  - ↳ track substring length

* find prefix in substring, obtain longest / largest counter

p is the parity of the sum of a list where $L[0 \ldots i-1]$.
The reason why it starts with 0 instead of 1 for i is because the ith element is not processed yet at the start of the loop so it will remain so 0.

the loop terminates when i equals the size of the input list where i = len(L) where L is the input list.
p is the parity of the sum of a list in $L[0 \ldots (len(L)-1)]$ which is equivalent to $L[0 \ldots n-1]$ where $n = len(L)$.
This proves that the invariant is held true for the entire list L.

$2T(n/2) + cn^2$
$\log_2(2) = 1$
$k = 2$
$O(n^2)$ #

$O(n \log(n))$ time complexity
possible aux space:
- $O(1)$
- $O(n)$
- $O(n \log(n))$

quickselect run on 0.3N

right list = faster one

↳ run 0.5N

run through to find
above 20seconds

---

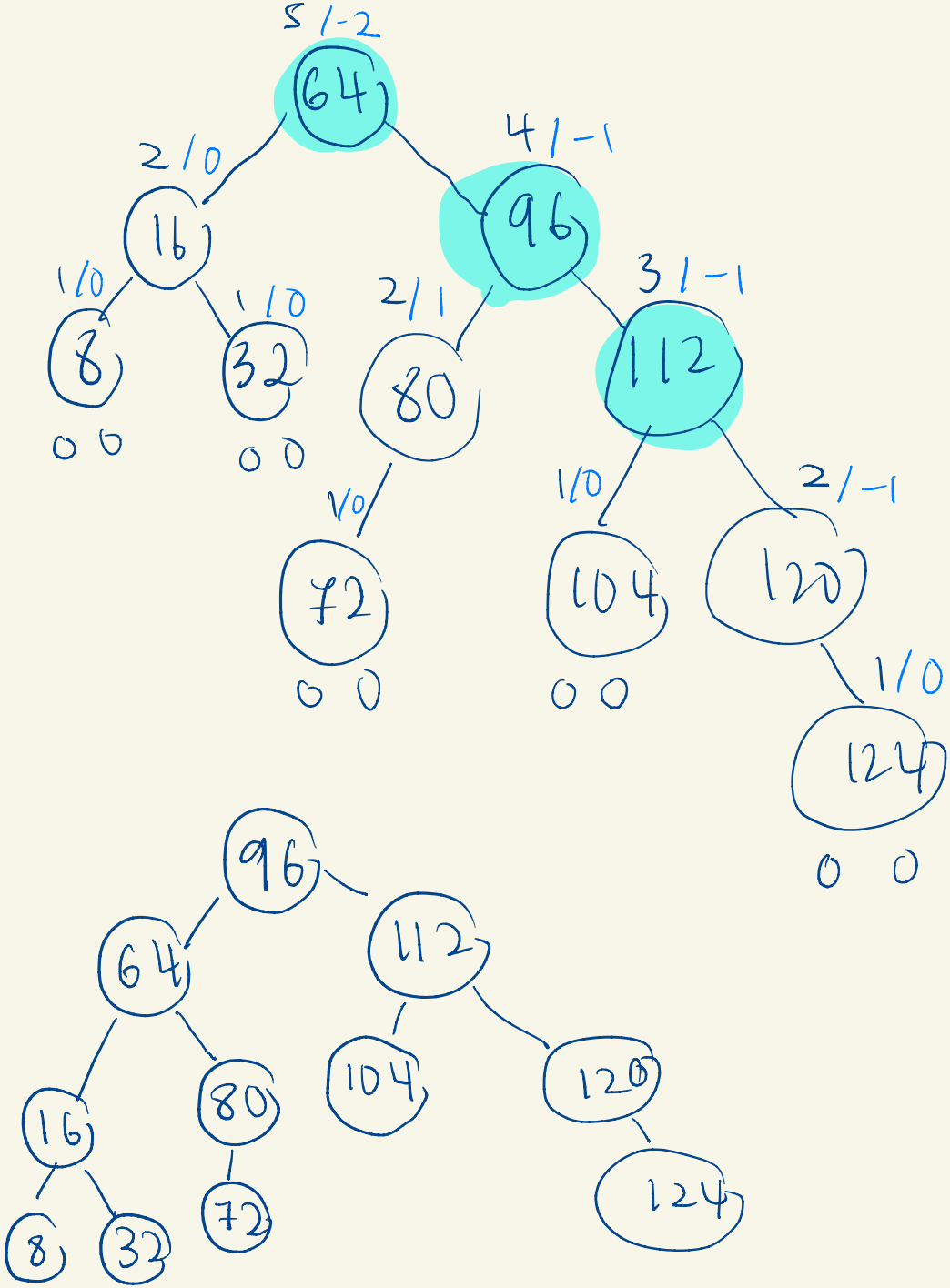
separate chaining hash table
implemented using BST

m = hash table size

n = no. item in table

best = O(1)

worst = O(n)


5 / -2
64
2 / 0
16
4 / -1
96
1 / 0
8
0 0
1 / 0
32
0 0
2 / 1
80
3 / -1
112
1 / 0
72
0 0
1 / 0
104
0 0
2 / -1
120
1 / 0
124
0 0

96
64
112
16
80
104
120
8
32
72
124

K = 2

(a) ID1 and ID5 have the same rank

ID1 + 1 = ID2 → rank 6

ID5 + 1 = ID6 → rank 6

hence not determined

(b) ID1 + 2 = ID3

ID3 rank = 5

ID5 + 2 = ID7

ID7 rank = 3

3 is lesser than 5

so in conclusion

ID7 comes by ID3

hence

ID5 comes by ID1.

(c) ID5 comes by ID1.