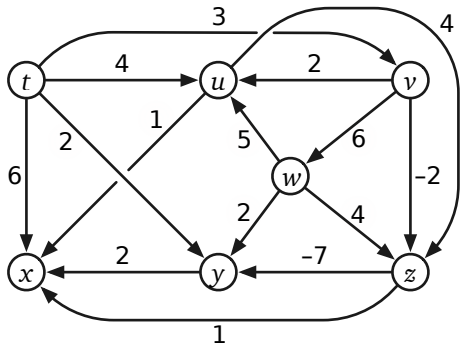
Write your answers in the separate answer booklet.

You have 120 minutes (after you get the answer booklet) to answer five questions.

Please return this question sheet and your cheat sheet with your answers.

1. ***Clearly*** indicate the following structures in the directed graph below. Don't be subtle! To indicate a subset of edges, draw a **HEAVY BLACK LINE** along the entire length of each edge. If the requested structure does not exist, write the word NONE. (The answer booklet contains several copies of this graph.)



   (a) A depth-first search tree rooted at v
   (b) A breadth-first search tree rooted at w
   (c) A shortest-path tree rooted at t
   (d) A list of vertices in topological order

2. Suppose you are given a directed graph $G = (V, E)$, each of whose edges are colored red, green, or blue. Edges in G do not have weights, and G is not necessarily a dag. A *rainbow walk* is a walk in G that does *not* contain two consecutive edges with the same color.

   Describe and analyze an algorithm to find all vertices in G that are reachable from a given vertex s through a rainbow walk.

3. Suppose we are given an n-digit integer X. Repeatedly delete one digit from either end of X (your choice) until no digits are left. The *square-depth* of X is the maximum number of perfect squares that you can see during this process.

   For example, the integer 32492 has square-depth 3, by the following sequence of digit deletions:

$$32492 \xrightarrow{5^2} 3249 \xrightarrow{7^2} 324 \xrightarrow{18^2} 24 \xrightarrow{2^2} 4 \to \varepsilon.$$

   Describe and analyze an algorithm to compute the square-depth of a given integer X, represented as an array $X[1..n]$ of n decimal digits. Assume you have access to a subroutine IsSquare that determines whether a given k-digit number (represented by an array of digits) is a perfect square ***in $O(k^2)$ time***.

4. Suppose you are given k arrays $A_1[1..n]$, $A_2[1..n]$, ..., $A_k[1..n]$, each with the same length n, and each sorted in increasing order. Describe an algorithm to merge the given arrays into a single sorted array. Analyze the running time of your algorithm as a function of n and k.

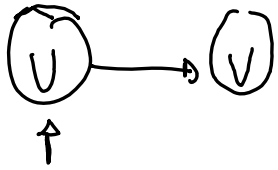
Mergesort → merge linear time ?

5. Sham-Poobanana University has been closed due to damage caused by a nationwide infestation of rabid squirrels. You decide to complete your computer science degree at Sham-Poobanana's long-time rival Burgoo Polytechnic ("Go Boilers!"), which has somehow survived the squirrel plague.

   Burgoo has an extremely rigid CS program. There is a single set of classes that *all* CS majors *must* take, and prerequisites are strictly enforced. Burgoo does not offer proficiency exams or accept transfer credit. On the other hand, Burgoo's classes are extremely easy; they do not enforce class attendance; and they do not limit the number of classes you can take in a single term. You rationally decide to speed-run your degree, but how?

   (a) Suppose you are given a directed acyclic graph $G = (V, E)$, where every vertex represents a required class and each edge $u \to v$ indicates that class u must be completed before you can begin class v. Describe an algorithm that computes a class schedule that uses the minimum number of terms; for each required class, the output of your algorithm should specify the term when you will take that class.

   (b) Burgoo relaxes its degree requirements due to massive overcrowding. Describe an algorithm that computes the minimum number of terms required to take ***at least 75%*** of the classes represented in G, while still obeying all prerequisite constraints. *[Hint: Use the output of your algorithm for part (a).]*



DAG

CS/ECE 374 A ✧ Fall 2026

Midterm 2 Practice 2

November 6, 2026

Name:	Maharsh Jani
NetID:	majani2

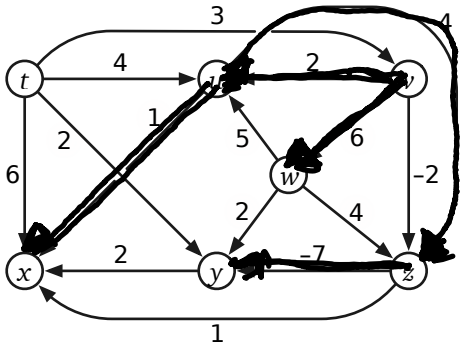
- ***Don't panic!***
- You have 120 minutes to answer five questions. **The questions are described in more detail in a separate handout.**
- If you brought anything except your writing implements, your **hand-written** double-sided 8½" × 11" cheat sheet, and your university ID, please put it away for the duration of the exam. In particular, please turn off and put away *all* medically unnecessary electronic devices.
- Please clearly print your name and your NetID in the boxes above.
- Please also print your name at the top of every page of the answer booklet, except this cover page. We want to make sure that if a staple falls out, we can reassemble your answer booklet. (It doesn't happen often, but it does happen.)
- Greedy algorithms require formal proofs of correctness to receive credit. Otherwise, proofs or other justifications beyond those listed in the standard grading rubrics are required if and only if we explicitly ask for them, using the word ***prove*** or ***justify*** in bold italics.

- **Do not write outside the black boxes on each page**. These indicate the area of the page that our scanners will actually scan. If the scanner can't see your work, we can't grade it.
- If you run out of space for an answer, feel free to use the overflow/scratch pages at the back of the answer booklet, but **clearly indicate where we should look**. If we can't find your work, we can't grade it.
- **Only work that is written into the stapled answer booklet will be graded**. We **do not** recommend detaching pages from the answer booklet, because any work on those detached pages will not be graded (and it disrupts our process for uploading solutions to Gradescope.) Please let us know if you detach a page either on purpose or accidentally so we can reattach it.
- We will provide additional blank scratch paper on request, but we will not grade anything written on it.
- **Please return all paper to Emily or a TA** including your **answer booklet**, your question sheet, your cheat sheet, and all scratch paper. There will be two piles of collected paper, the first for answer booklets and the second for everything else. We **will not** consider the paper in the second pile when grading your exam.

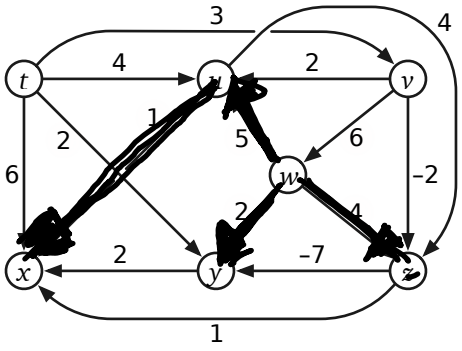
This sentence contains four and three fourths percent a's, five and one half percent c's, three percent d's, eighteen and one fourth percent e's, four and one half percent f's, three fourths percent g's, five percent h's, two and one half percent i's, two percent l's, twelve and one half percent n's, six percent o's, five percent p's, eight percent r's, seven percent s's, nine and three fourths percent t's, two and one fourth percent u's, one and one half percent v's, one and one fourth percent w's and one half percent x's.

CS/ECE 374 A ✧ Fall 2026 Midterm 2 Practice 2 Problem 1	Name: Maharsh Jani

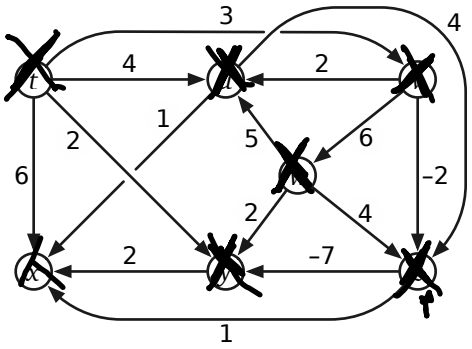
Clearly indicate the following structures in the directed graph below. Don't be subtle! To indicate a subset of edges, draw a **HEAVY BLACK LINE** along the entire length of each edge. If the requested structure does not exist, write the word NONE.



(a) A depth-first search tree rooted at *v*



(b) A breadth-first search tree rooted at *w*



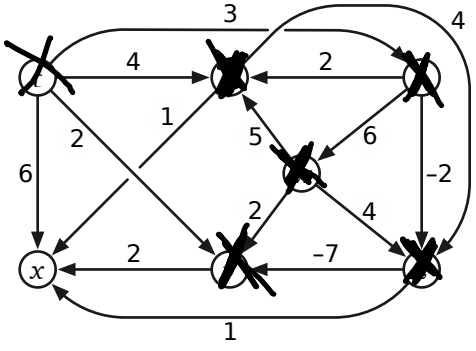
Start t

(c) A shortest-path tree rooted at *t*

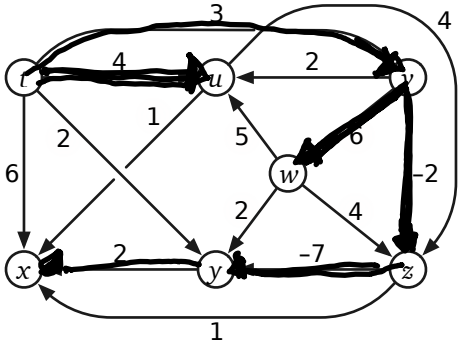
t v w x y z
4 3 9 −4 −6 1

t, v, w, v, z, y, x

(d) A list of vertices in topological order



[scratch]



[scratch]

CS/ECE 374 A ✧ Fall 2026 Midterm 2 Practice 2 Problem 2	Name: Maharsh Jani

Suppose you are given a directed graph $G = (V, E)$, each of whose edges are colored red, green, or blue. Edges in G do not have weights, and G is not necessarily a dag. A *rainbow walk* is a walk in G that does *not* contain two consecutive edges with the same color.

Describe and analyze an algorithm to find all vertices in G that are reachable from a given vertex s through a rainbow walk.

$G' = (V', E')$ Allow the thing we can take in G'

$V' = V \times \{Red, Green, Blue, \phi\}$ ⇒ colour of last edge used to get to that vertex

$E' = \{(U, C') \to (V, C) \mid U \to V \text{ in } E \text{ and } C' \neq C\}$

↳ Call WFS (DFS or BFS) starting on (s, ϕ) and that would help with reachability in this rainbow walk

$|V'| = 4V$ vertices

$|E'| = 3E$ edges

⇒ Total Runtime : Time to Build G' ⇒ $O(|V'| + |E'|)$
Time to run WFS ⇒ $O(|V'| + |E'|)$
$O(|V'| + |E'|)$

So it gets us with $O(4V + 3E)$
$= O(V + E)$ ⇒ $O(n + m)$

CS/ECE 374 A ✧ Fall 2026 Midterm 2 Practice 2 Problem 3	Name: Maharsh Jani

Suppose we are given an n-digit integer X. Repeatedly remove one digit from either end of X (your choice) until no digits are left. The *square-depth* of X is the maximum number of perfect squares that you can see during this process.

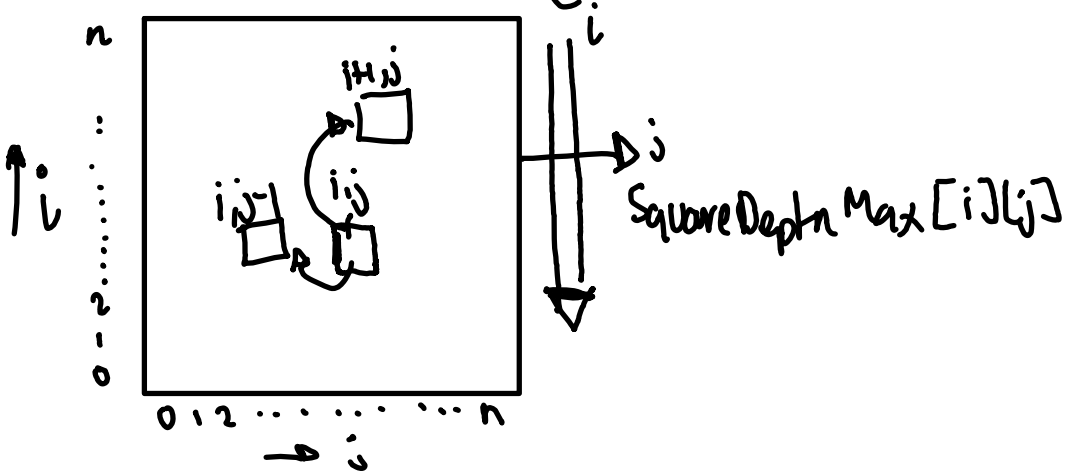
Describe and analyze an algorithm to compute the square-depth of a given integer X, represented as an array $X[1..n]$ of n decimal digits. Assume you have access to a subroutine IsSquare that determines whether a given k-digit number (represented by an array of digits) is a perfect square ***in $O(k^2)$ time***.

SquareDepthMax(i,j): it would return the max square-depth of X[i...j]

↳ Need to call SquareDepthMax(1,n)

$$\text{SquareDepthMax}(i,j) = \begin{cases} 0, & \text{if } i > j \\ \text{else,} & \end{cases}$$

$$\text{else,} \quad [\text{isSquared}(X[i..j])] + \max \begin{cases} \text{SquareDepthMax}(i+1, j) \\ \text{SquareDepthMax}(i, j-1) \end{cases}$$



Runtime: $O(n^2)$ subproblems → $O(n^2)$ work per subproblem

$\Rightarrow O(n^4)$

CS/ECE 374 A ✧ Fall 2026 Midterm 2 Practice 2 Problem 4	Name: Maharsh Jani

Suppose you are given k sorted arrays $A_1[1..n], A_2[1..n], \ldots, A_k[1..n]$, all with the same length n. Describe an algorithm to merge the given arrays into a single sorted array. Analyze the running time of your algorithm as a function of n and k.

MergeAll $(A_1 \ldots A_k)$: It takes all the arrays from $A_1 \ldots A_k$ and returns the final merged sorted array

```
MergeAll(A_1 ... A_k):
  if K < 374:
    ret Brute force to get answer
  m ← ⌈k/2⌉
  A_x ← mergeAll(A_1 ... A_m)
  A_y ← mergeAll(A_m+1 ... A_k)
  A_z ← merge(A_x, A_y)
  return A_z
```

K=0: return [ ]

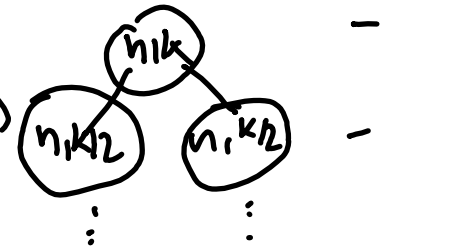
K=1: $A_1[1 \ldots n]$

else:

$nk/2$ $nk/2$ $= O(nk)$

$\Rightarrow T(n,k) = 2T(n, k/2) + O(nk)$

$\hookrightarrow O(nk \log k)$ $\log(k)$ $O(nk)$



$(u) \to (v) \Rightarrow \text{term}(u) < \text{term}(v)$

CS/ECE 374 A ✧ Fall 2026 Midterm 2 Practice 2 Problem 5	Name: Maharsh Jani

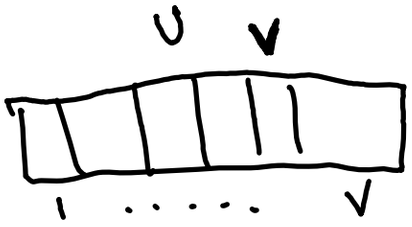
Suppose you are given a directed acyclic graph $G = (V, E)$, where every vertex represents a required class and each edge $u \to v$ indicates that class u must be completed before you can begin class v.

(a) Describe an algorithm that computes a class schedule that uses the minimum number of terms. For each class, the output of your algorithm should specify the term when you will take that class.

(b) Describe an algorithm that computes the minimum number of terms required to take *at least 75%* of the classes represented in G, while still obeying all prerequisite constraints.

(a) for any of class v, term(v) number of classes on the longest path in G ends at v

$$\text{term}(v) = \begin{cases} 1 & \text{if we are source vertex (no incoming edges)} \\ 1 + \max_{u \to v} (\text{term}(u)) & \text{otherwise} \end{cases}$$



evaluate this in topological order

term[1...v] ⇒ Runtime: we are doing topo sort: $O(|V|+|E|) = O(n+m)$

Get terms(G):
- toposort of G
- for each v in this topo ordering:
  - term[v] = 1
  - for edges u→v
    - term[v] = max{term[v], term[u]+1}

returning term[1...v]

⇒ [redacted]

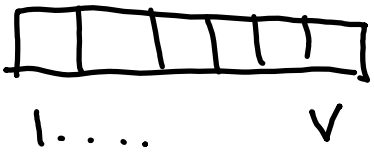
for any vertex v, what term that vertex is in

(scratch paper)

Term1 Term2 Term3
124 128 273
101
103

(b)

term t	1	2	3
classes in term t	3	1	1
running total	3	4	5



if I meet 75%. then I can simply return

75%. of 5
↳ 3.75>

speedrun(y): gives us the the min terms to do to get atleast 75%. completion in classes

↳ speedrun(G):

Terms = GetTerms(G) // from part(a)

count[1...V] ← 0

for each v:
  count[Term[v]]++

total ← 0

for t ← 1 to V
  total += count[t]
  if total ≥ 3/4(V):
    return t

⇒ Total Runtime:
O(|V|+|E|)
O(n+m)

(scratch paper)

(scratch paper)

(scratch paper)

(scratch paper)