CSS324: Artificial Intelligence
Midterm Mock Exam

curated by The Peanuts

Name.....ID.....Section.....Seat No.....

Conditions: Semi-closed Book

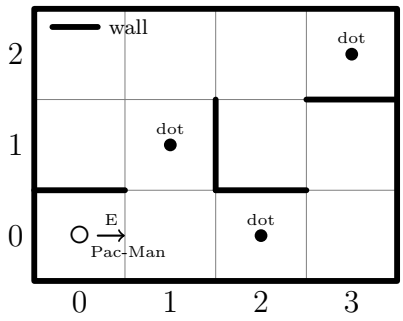
Directions:

1. This exam has 11 pages (including this page).
2. You may bring **one A4 cheat sheet**, written on both sides. If your handwriting is smaller than 2pt font, congratulations, you've basically invented microchips.
3. Calculators are NOT allowed.
4. Cheating is strictly prohibited. We have invisible drones watching (probably)
5. Draw search trees and diagrams where requested. Stick figures are acceptable only if they're funny.
6. Have fun!

May not be 100% correct, but I've checked that it's quite okay.
Please double-check before relying on it.

Question 1

Your goal is to navigate Pac-Man through the following maze to collect all the dots. Each location in the maze is denoted by a coordinate (x, y). Pac-Man is initially placed at $(0, 0)$ and is facing east (right direction). The maze contains dots at coordinates $(1, 1)$, $(2, 0)$, and $(3, 2)$. Pac-Man must collect all dots to win the game.



Pac-Man accepts only four commands: (1) move forward 1 block, (2) rotate 90° clockwise, (3) rotate 180° clockwise, and (4) rotate 270° clockwise. Pac-Man cannot move through walls or outside the maze boundaries.
Formulate this problem so that it can work with the search algorithms studied in class without any modification.

(a) Define how to represent a state and write the initial state.

State representation: A state can be represented as `(x, y, direction, collected_dots)` where:

- `(x, y)` is Pac-Man's position in the maze
- `direction` is Pac-Man's facing direction (N, S, E, W)
- `collected_dots` is a set/list of dots that have been collected so far

Initial state: $(0, 0, E, \{\})$

Where Pac-Man is at position $(0, 0)$, facing East, and no dots have been collected yet.

(b) Explain how to generate a set of successors from a given state (using pseudo-code).

Successor function:

```
function getSuccessors(state):
    successors = []

    // Action 1: Move forward 1 block
    if direction == N and not WallAt(x, y+1):
        new_collected = updateDots(collected_dots, x, y+1)
        successors.append(State(x, y+1, N, new_collected))
    if direction == E and not WallAt(x+1, y):
        new_collected = updateDots(collected_dots, x+1, y)
        successors.append(State(x+1, y, E, new_collected))
    if direction == S and not WallAt(x, y-1):
        new_collected = updateDots(collected_dots, x, y-1)
        successors.append(State(x, y-1, S, new_collected))
    if direction == W and not WallAt(x-1, y):
        new_collected = updateDots(collected_dots, x-1, y)
        successors.append(State(x-1, y, W, new_collected))

    // Action 2: Rotate 90° clockwise
    successors.append(State(x, y, Rotate(direction, 90), collected_dots))

    // Action 3: Rotate 180° clockwise
    successors.append(State(x, y, Rotate(direction, 180), collected_dots))

    // Action 4: Rotate 270° clockwise
    successors.append(State(x, y, Rotate(direction, 270), collected_dots))

    return successors
```

(c) What is the depth of the shallowest goal of this problem? Explain your reasoning for how you obtained this value.

Answer: The depth of the shallowest goal is **9 steps** (using Manhattan distance).

Reasoning using Manhattan Distance:

- Starting point: $(0, 0)$
- Goal dots at: $(1, 1)$, $(2, 0)$, and $(3, 2)$
- Manhattan distances:

$$d((0,0),(1,1)) = |0-1| + |0-1| = 1+1 = 2 \quad (1)$$
$$d((0,0),(2,0)) = |0-2| + |0-0| = 2+0 = 2 \quad (2)$$
$$d((0,0),(3,2)) = |0-3| + |0-2| = 3+2 = 5 \quad (3)$$

- Minimum spanning path connecting all dots:

$$d((1,1),(2,0)) = |1-2| + |1-0| = 1+1 = 2 \quad (4)$$
$$d((2,0),(3,2)) = |2-3| + |0-2| = 1+2 = 3 \quad (5)$$

- Optimal route: $(0, 0) \to (2, 0) \to (1, 1) \to (3, 2)$
- Total Manhattan distance: $2 + 2 + 5 = 9$ steps

(d) Is this search problem more suitable for breadth-first search or depth-first search? Justify your answer.

Answer: This problem is more suitable for **Breadth-First Search (BFS)**.

Justification:

- **Optimal solution guarantee:** BFS finds the solution with minimum number of steps, which is important for finding the shortest path to collect all dots
- **Systematic exploration:** BFS explores all states at depth d before exploring states at depth $d+1$, ensuring we find the shortest solution first

Question 2

Taylor Swift is planning her Eras Tour and needs to visit cities $\{A, B, C, D, E, F\}$ exactly once, starting and ending at city A. However, she has special requirements: she must perform at least 2 consecutive concerts in cities that have populations over 1 million (cities B, D, and F). The travel costs between cities are given in the following matrix:

	A	B	C	D	E	F
A	0	15	20	25	30	35
B	15	0	10	18	22	28
C	20	10	0	12	15	25
D	25	18	12	0	8	16
E	30	22	15	8	0	20
F	35	28	25	16	20	0

(a) Formulate this as a search problem by defining:

(1) **State representation:** `(current_city, visited_cities, consecutive_large_count)` where:

- `current_city`: the city where Taylor is currently located
- `visited_cities`: set/list of cities already visited
- `consecutive_large_count`: number of consecutive large cities (B, D, F) visited in current sequence

(2) **Initial state:** $(A, \{A\}, 0)$

(3) **Goal test:** A state `(current_city, visited_cities, consecutive_large_count)` is a goal if:

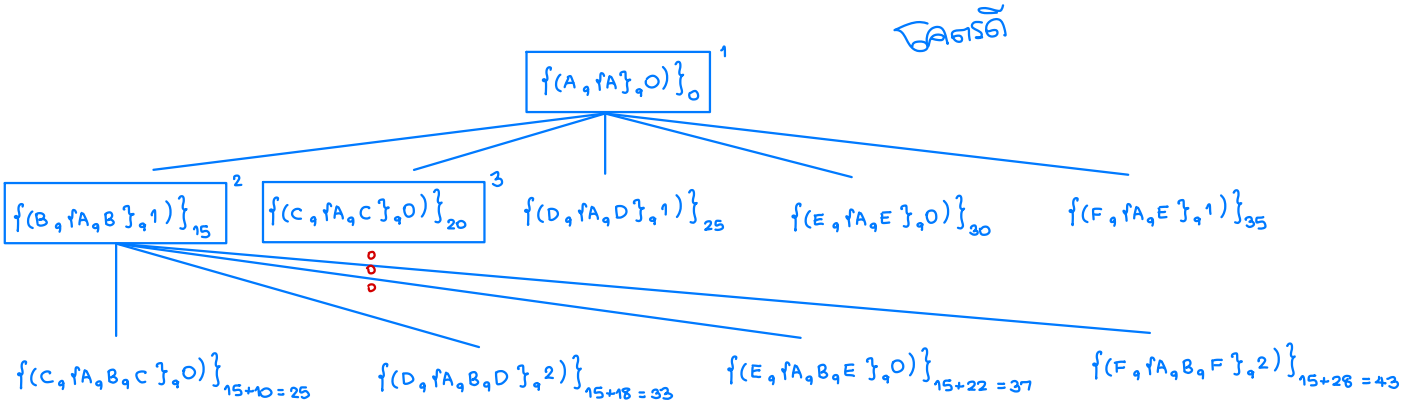
- `current_city` = A (returned to starting city)
- `|visited_cities|` = 6 (all cities visited exactly once)
- The constraint of at least 2 consecutive large city concerts has been satisfied during the tour

(4) **Actions and successor function:** From state `(current_city, visited_cities, count)`:

- Available actions: travel to any unvisited city (if not all visited) or return to A (if all others visited)
- Generate successor by moving to next city and updating:
  - New current city
  - Add city to visited set
  - Update consecutive count based on whether new city is large or not!!

(5) **Step cost:** The travel cost between cities as given in the cost matrix.

(b) Draw the search tree when conducting Uniform-Cost Graph Search on this problem. Expand at least 8 nodes and show the frontier contents after each expansion.



(c) Design an admissible heuristic function for this problem that takes into account the constraint about consecutive large city concerts.

Admissible Heuristic Function: $h(state) = h_1(state) + h_2(state)$

Where:

- $h_1(state)$: Minimum cost to visit remaining unvisited cities and return to A
  - Use minimum spanning tree cost of unvisited cities plus minimum cost to connect to current city and to A
  - This ensures we don't overestimate the travel cost
- $h_2(state)$: Penalty for constraint satisfaction
  - If the consecutive large city constraint is not yet satisfied and no sequence of 2+ consecutive large cities exists in the remaining unvisited cities, return ∞ (inadmissible state)
  - Otherwise, return 0 (no additional cost penalty)

Why it's admissible:

- h_1 never overestimates the actual minimum travel cost needed
- h_2 is either 0 or correctly identifies impossible states
- The sum provides a lower bound on the actual cost to reach a goal state

Question 3

The following Python code attempts to implement Breadth-First Graph Search algorithm, but it contains several bugs that prevent it from working correctly. Identify and correct all the mistakes to make it work properly.

```
from collections import deque
from state import State
from node import Node

def breadth_first_graph_search(initial_state: State):
    initial_node = Node(initial_state, None, 0, 0)
    frontier = [initial_node]
    explored = set()
    n_visits = 0

    while frontier:
        n_visits += 1
        node = frontier.pop()

        if node.state.is_goal():
            return node, n_visits

        explored.add(node.state)

        for child_state, step_cost in
            node.state.successors():
            if child_state not in explored:
                child_node = Node(child_state, node,
                                  node.path_cost + step_cost,
                                  node.depth + 1)
                frontier.append(child_node)

    return None, n_visits
```

(a) List all the errors in the above code and provide the corrected version.

Errors identified:

(a) **Wrong data structure:** Using list with `pop()` makes it LIFO (stack/DFS), not FIFO (queue/BFS)

(b) **Missing goal test at insertion:** Should check if child is goal when adding to frontier

(c) **Missing frontier check:** Should check if child state is already in frontier before adding

(d) **Inefficient frontier management:** No way to check if state is already in frontier

Corrected code:

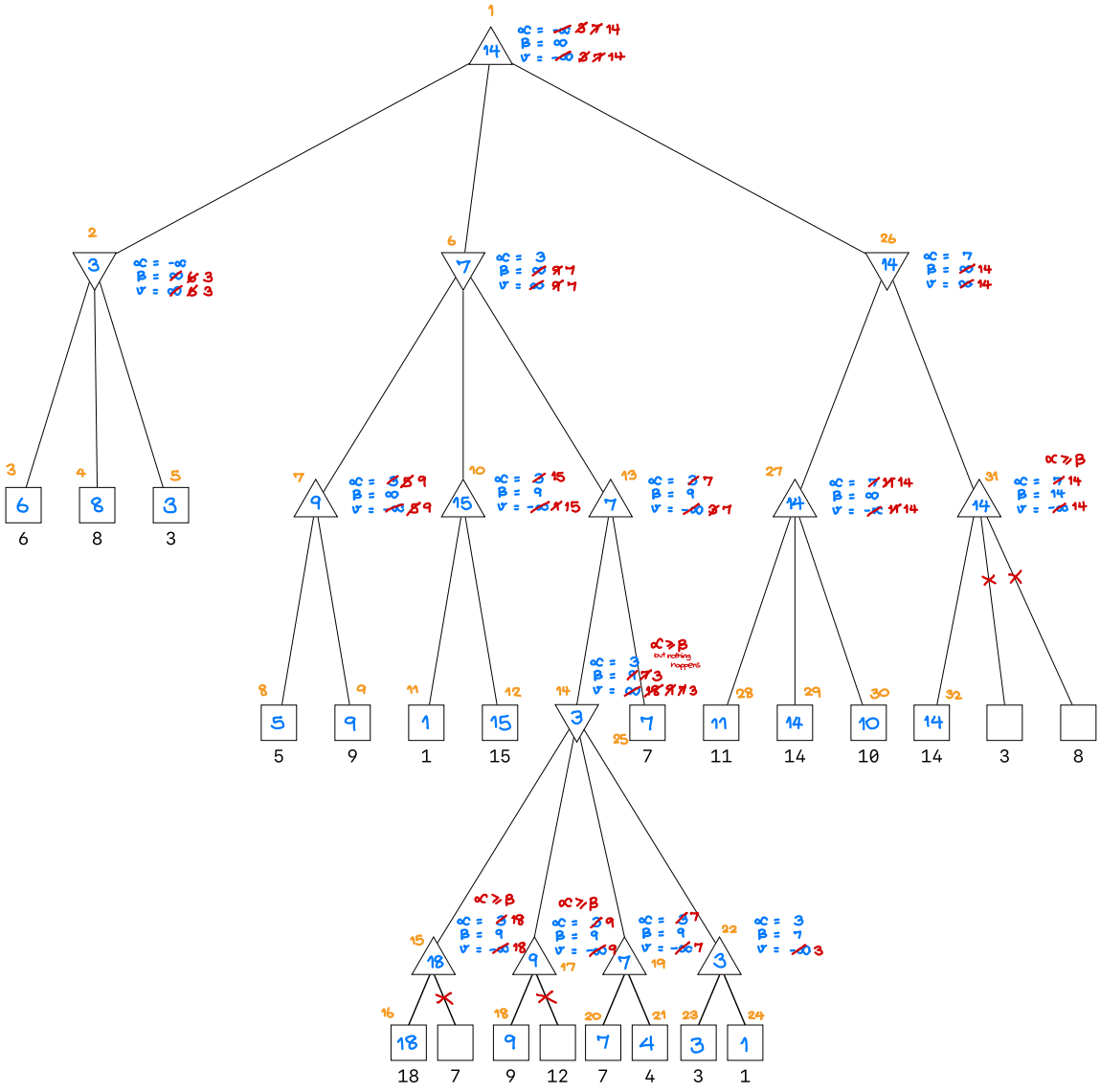
```python
from collections import deque
from state import State
from node import Node

def breadth_first_graph_search(initial_state: State):
    initial_node = Node(initial_state, None, 0, 0)
    frontier = deque([initial_node])  # Use deque for FIFO
    frontier_states = {initial_state}  # Track states in frontier
    explored = set()
    n_visits = 0

    while frontier:
        n_visits += 1
        node = frontier.popleft()  # FIFO: remove from left
        frontier_states.remove(node.state)

        if node.state.is_goal():
            return node, n_visits

        explored.add(node.state)

        for child_state, step_cost in node.state.successors():
            if (child_state not in explored and
                child_state not in frontier_states):
                child_node = Node(child_state, node,
                                  node.path_cost + step_cost,
                                  node.depth + 1)
                frontier.append(child_node)  # Add to right
                frontier_states.add(child_state)

    return None, n_visits
```

(b) Explain the key difference between Breadth-First Search and Depth-First Search in terms of:

- **Data structure used:**
  - BFS: FIFO queue (deque with popleft/append)
  - DFS: LIFO stack (list with pop/append)
- **Order of node exploration:**
  - BFS: Level by level (explores all nodes at depth d before depth $d+1$)
  - DFS: One path at a time (explores deeply along one branch before trace back)
- **Memory requirements:**
  - BFS: Higher memory usage - must store all nodes at current level (exponential in breadth)
  - DFS: Lower memory usage - only stores nodes along current path (linear in depth)

Question 4

Consider the following game tree, use alpha-beta pruning algorithm to cross out unnecessary subtrees. Assume that the tree is evaluated by depth-first search algorithm from left to right.



Question 5

At S2T University, there's a computer science department where students, professors, and courses interact in various ways. Use the following predicates:

- $Student(x)$ - "x is a student"
- $Professor(x)$ - "x is a professor"
- $Course(x)$ - "x is a course"
- $Enrolled(x, y)$ - "student x is enrolled in course y"
- $Teaches(x, y)$ - "professor x teaches course y"
- $Smart(x)$ - "x is smart"
- $Difficult(x)$ - "course x is difficult"
- $Likes(x, y)$ - "x likes y"

(a) Translate the following English sentences into First-Order Logic:

(1) "Every student is enrolled in at least one course"

$$\forall x(Student(x) \rightarrow \exists y(Course(y) \land Enrolled(x, y)))$$

(2) "Some professors teach only difficult courses"

$$\exists x(Professor(x) \land \forall y(Teaches(x, y) \rightarrow Difficult(y)))$$

(3) "No student likes all difficult courses"

$$\forall x(Student(x) \rightarrow \exists y(Course(y) \land Difficult(y) \land \neg Likes(x, y)))$$

(4) "There exists a professor who is liked by all students in at least one course they teach"

$$\exists x(Professor(x) \land \exists y(Course(y) \land Teaches(x, y) \land \forall z((Student(z) \land Enrolled(z, y)) \rightarrow Likes(z, x))))$$

(b) Translate the following FOL sentences into plain English:

(1) $\forall x \exists y (Student(x) \rightarrow (Course(y) \wedge Enrolled(x, y) \wedge Difficult(y)))$

"Every student is enrolled in at least one difficult course"

(2) $\exists x \forall y \forall z (Professor(x) \wedge Course(y) \wedge Student(z) \wedge Teaches(x, y) \wedge Enrolled(z, y) \rightarrow Smart(z))$

"There exists a professor such that all students enrolled in any course that this professor teaches are smart"

(3) $\forall x \exists y \exists z (Student(x) \wedge Professor(y) \wedge Course(z) \wedge Enrolled(x, z) \wedge Teaches(y, z) \wedge \neg Likes(x, y))$

"Every student is enrolled in at least one course taught by a professor whom the student does not like"

(c) Consider the statement: "Every smart student is enrolled in exactly two courses." Write this in FOL using appropriate quantifiers and equality. Then, write the ***negation*** of this statement in both FOL and plain English.

Original statement in FOL:

$$\forall x(Student(x) \land Smart(x) \rightarrow \exists y \exists z(Course(y) \land Course(z) \land y \neq z \land Enrolled(x,y) \land Enrolled(x,z) \land \forall w(Course(w) \land Enrolled(x,w) \rightarrow (w = y \lor w = z))))$$

Negation in FOL:

$$\neg\forall x(Student(x) \land Smart(x) \rightarrow \exists y \exists z(Course(y) \land Course(z) \land y \neq z \land Enrolled(x,y) \land Enrolled(x,z) \land \forall w(Course(w) \land Enrolled(x,w) \rightarrow (w = y \lor w = z))))$$

Negation in plain English:

"There exists a smart student who is not enrolled in exactly two courses" or "At least one smart student is enrolled in a number of courses other than exactly two."