Question 7

Part A: Proof that Consistency ⇒ Admissibility

1. Admissible heuristic $h(n)$:
Never overestimate the true cost to reach the goal from node n
Formally:
$h(n) \leq h^*(n)$ where $h^*(n)$ is the actual lowest cost from n to a goal

2. Consistent (monotone) heuristic $h(n)$ satisfies the triangle inequality for every node n and successor n'
$h(n) \leq c(n,n') + h(n')$ where $c(n,n')$ is the cost of moving from n to n'

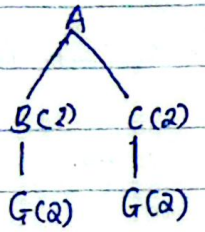
Proof [We want to show if h is consistent, then h is admissible]

1. Base Case (goal node G)
$h(G) = 0$ by definition. This is trivially admissible since the true cost from the goal to itself is 0

2. Inductive step:
Assume $h(n') \leq h^*(n')$ for all nodes n' at distance $\leq k$ from the goal
Consider node n one step away from successor n'
By consistency: $h(n) \leq c(n,n') + h(n')$
By the inductive hypothesis: $h(n) \leq c(n,n') + h^*(n')$
But $c(n,n') + h^*(n') \geq h^*(n)$ (the shortest path cost from n to the goal)
Thus: $h(n) \leq h^*(n)$
Hence consistency implies admissibility

Part B: Example of Admissible but inconsistent heuristic

Consider a search tree



True Paths costs:
A → B → G = 4
A → C → G = 4

heuristic h value
$h(A) = 3$
$h(B) = 2$
$h(C) = 2$
$h(G) = 0$

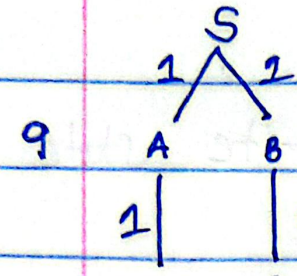
Check for admissibility:
for all nodes, $h(n) \leq$ true cost to goal

Check consistency:
$A \to B$: $h(A) \leq c(A,B) + h(B)$?
$h(A) = 3, c(A,B) = 2, h(B) = 2 \to 3 \leq 4$
$A \to C$: $h(A) \leq c(A,C) + h(C)$?
$h(A) = 3, c(A,C) = 2, h(C) = 2 \to 3 \leq 3$
$B \to G$: $h(B) \leq c(B,G) + h(G)$?
$h(B) = 2, c(B,G) = 2, h(G) = 0 \to 2 \leq 2$

Now tweak $h(B) = 3$ still admissible, since cost $B \to G = 2$, $h(B) = 3 \leq 2$? actually not admissible

$h(A) = 3, h(B) = 2, h(C) = 4, h(G) = 0$
Now $A \to C$: $h(A) = 3 \leq c(A,C) + h(C) = 2 + 4 = 6$ (still consistent) ✓
$h(B) = 2 \leq c(B,G) + h(G) = 2 + 0 = 2$ ✓ Okay
Example text shows sometimes for more complex graphs, admissible heuristics can jump, breaking monotonicity

The key idea is admissible only bounds final cost, while consistency ensures monotone decrease along edges. A heuristic can underestimate some nodes but violate the triangle inequality, making it inadmissible in graph search if used with A*



Edges:

$S \to A$ cost = 1
$A \to G$ cost = 1 → Total cost $S \to A \to G = 2$ (Optimal)
$S \to B$ cost = 4
$B \to G$ cost = 2 → Total cost $S \to B \to G = 3$ (sub optimal)

The Goal is to reach G

Heuristic (inadmissible)

Define heuristic h as:

$h(G) = 0$
$h(A) = 10$ ← Overestimates true remaining cost from A (true cost = 1)
$h(B) = 0$

h is inadmissible because $h(A) = 10 > 1 = h^*(A)$

A* run (trace, using $f = g + h$)

Start $g(S) = 0$, $f(S) = h(S)$. We expand S and push successors. After expanding S:

- Node A: $g(A) = 1$, $F(A) = g + h = 1 + 10 = 11$
- Node B: $g(B) = 1$, $F(B) = 1 + 0 = 1$
- frontier = {B(f=1), A(f=11)}

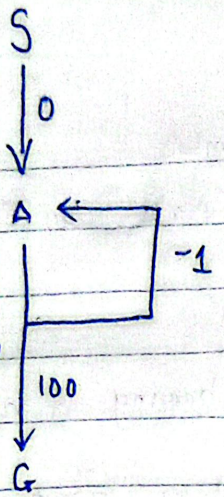
1. A* pops the smallest f → pop B (f=1), expand B
   Generate G via B: $g(G) = g(B) + cost(B, G) = 1 + 2 = 3$, $f(G) = 3 + h(G) =$
   frontier now = {G(f=3), A(f=11)}
2. A* pops the smallest f → pop G (f=3). G is a goal → A* returns path $S \to B \to G$ with cost 3 and stops

But the optimal path is $S \to A \to G$ with cost 2. A* returned a suboptimal solution because the heuristic caused it to explore the worst route first and accept the goal when it was first encountered

This Tiny example shows that if the heuristic overestimates (i.e is inadmissible), A* can be led to expand a suboptimal branch first and accept a suboptimal goal - Therefore inadmissibility can cause A* to return to a suboptimal solution

QUESTION 10

labeled edges:
S→A Cost = 0
A→A cost = -1
A negative self-loop that can be repeated a number of time
A→G Cost = 100



All nodes S, A, G are part of a finite space {S, A, G}

Why the Paradox?
Considering any path from S to G that loops K times around the A loop before going to G. The total Cost is:

$Cost_k = 0 + k \cdot (-1) + 100 = 100 - k$

for k = 0 (no loops), Cost = 100
for k = 50 Cost = 50
for k = 200 Cost = -100

As k increases without bound, Cost_k → infinity
That means:
for any finite no C, we can choose K large enough so that Cost_k < C
Therefore there is no finite lower bound attained by any finite path, the infimum of achievable cost is $-\infty$

If we interpret "optimal" as minimizing total cost, the best way to minimize Cost is to make k as large possible i.e loop forever on A and never go to G. Thus the "Optimal solution" (minimizing total cost) is to never reach the goal in finite time

That's the Paradox; the state space is finite, every state and edge are well defined, but because a negative cycle is reachable and can be repeated arbitrarily, the problem of finding a finite Path of minimum cost is ill posed
No infinite optimal path exists

FORMAL JUSTIFICATION

Let the set of all finite S → G Paths be P. for each path $p_k \in P$ that loops k times
The cost is $c(p_k) = 100 - k$
For any real number M there exists $k > 100 - M$ such that $c(p_k) < M$. Hence:

$$\inf_{p \in P} c(p) = -\infty$$

Since every finite path has finite Cost but the infimum is $-\infty$, no finite path attains the infimum. The only way to approach the infimum is to take infinitely many loop iteration (i.e never reach G). Hence an optimal policy that minimizes Cost would not reach the goal in finite time

Question 11

Formulation as a search problem

1. States: Each state represents a natural sequence of symbols chosen so far
   Eg: If the alphabet $\Sigma = \{a, b, c, \ldots\}$, then a state can be ('a', 'e').
   The start state is the empty sequence: ().

2. Actions: At any state of length $k < 5$, an action is choosing a new symbol from the alphabet and appending it.
   - Number of actions per state = size of alphabet eg 1000+

3. Transition model: Applying an action extends the current sequence by one symbol

4. Goal Test: A state is a goal if it satisfies the problem-specific criteria (eg a target sequence, valid word

5. Path cost: Each step costs 1 (or uniform cost). The cost is sequence length

Choice of Search Algorithm

DFS: Explores sequences deeply
   Advantage: low memory usage
   Risk: May find a suboptimal or very late goal sequence in huge alphabets

BFS: Explores sequences level by level
   Advantage: Guaranteed to find the shortest sequence (minimum length)
   Risk: Memory grows exponentially with alphabet size

Suggested Modification

Because the alphabet is huge:

1. Pruning: Avoid exploring sequences that are invalid or redundant
2. Iterative Deepening DFS (IDDFS): Combines DFS memory efficient with BFS optimality
3. Heuristic Search: If a heuristic is available (estimating closeness to goal), use A* to reduce explored states

Question 12

12

IDS performs depth-limited DFS repeatedly, increasing the depth limit each iteration

Guarantees finding the shallowest solution (shortest path in terms of number of actions)

Limitations: It does not account for different step costs, only the number of steps

Modification for lowest-cost paths

1. Track path costs:
   Each node keeps track of the cumulative cost from the start state.

2. Iterative cost limits instead of depth limits:
   Instead of limiting the depth, limit the total cost of paths explored in each iteration
   Start with a cost limit $C = 0$ or the minimum step cost
   Increment the cost limit in each iteration to explore more expensive paths

3. Expanded nodes:
   Only expand nodes whose cumulative cost $u \leq$ current cost limit
   Record the lowest-cost solution found so far

Proof of optimality

Let C^* be the cost of the true lowest-cost solution

Iterative cost limited DFS explores all paths with cost $\leq$ costlimit

Once costlimit $>= C^*$, the lowest cost solution is guaranteed to be found

Any solution returned is thus optimal because:

- No lower cost path exists (all paths $\leq C^*$ have been explored)
- All paths with cost $< C^*$ were explored in previous iterations and failed