CPSC 4610 – Artificial Intelligence

Homework 1

Seattle University

Jakob Balkovec

01/08/2025

Q1 (15 points) – AI

Explain why an AI solution to a costly problem may be described as a path generation problem. Your explanation should include some detail(s) about heuristics.

A1

We can describe AI as a path generation problem because it often involves finding an optimal or a good approximate sequence of steps/instructions/path to transition from either an initial state or an intermediate state to the desired goal state while minimizing costs. In this context, "costs" is a relative/flexible term that depends on the situation--it could mean money, resource limits, or even computational effort.

Path Generation

PG problem mostly include graphs (at least from my experience), where nodes represent initial/intermediate/goal states and edges represent actions or transitions between states. Essentially our AI model must navigate this graph to reach the goal state. Each edge (transition) in the graph is associated with a cost*, and the total cost of a path is the sum of these costs. The model's objective is to generate a path that minimizes or optimizes this total cost. Finding the best path requires the model to explore the state space in some "efficient" way (again "efficient" is a relative term here). Since most of real world problems involve large (maps) or even infinite spaces (navigating a drone in the air, Tesla Autopilot) brute force is usually not the right way to go about them as it tends to be computationally intensive and requires a significant amount of memory (Knapsack problem).

*** OFFICE HRS: What if they're not weighed?**

Heuristics

Heuristics are a crucial part of AI models. They play a crucial role in making problems like PG computationally feasible. In general, a heuristic is a problem specific rule or estimate that guides the search process towards the goal state more efficiently than an algorithm. It does that by prioritizing certain paths over others. Now, by using heuristics we can help reduce the search space by eliminating paths that are unlikely to lead to an optimal solution (branch pruning), allowing the model to focus its resources on "promising" areas of the state space.

Heuristics vs Algorithm Example (Google Maps)

Imagine we need to get from Cap Hill to Downtown Seattle. Google's A* algorithm would find the most optimal/fastest route that would get us there in say 8 minutes in no time. This becomes a problem if we were planning a cross-country trip (e.g. SEA - NYC). A heuristic approach on the other hand is used to quickly find other possible routes (no tolls, congestion avoidance, etc.).

AI Algorithms and Heuristics in PG

Most common "algorithm" in PG is A* (A-star) which combines actual cost from the start to a state ($g(n)$) with a heuristic estimate of the cost to the goal ($h(n)$), making it both optimal and efficient. The other ones are greedy algorithms. They use heuristics to make locally optimal choices at each step, often at the expense of global optimality.

Q2 (30 points) – Search

Create a weighted graph with at least 10 nodes which include an identified starting node and one or more goal (end) nodes), and a heuristic, such that A* outperforms Depth-First search and UCS (uniform cost search). Explain why.

A2

NOTE:

The visual representation of the graph, adjacency matrix, and the heuristic can be found in the attachment under the same assignment. List of files is noted below

Comparing the Search Methods

PATH1: $N5 \rightarrow N6 \rightarrow N8$
PATH2: $N8 \rightarrow N10$

DFS

- It explores a branch (consequently, the graph) as deep as possible before backtracking. It might explore some inefficient paths, for example using **PATH1** to get to the goal state, which is expensive. The depth-driven nature of the algorithm is the main cause for the slower performance and in some cases even suboptimality as it does not consider costs or heuristics.

UCS

- It explores the lowest-cost paths first, but does not use a heuristic. It could potentially waste time exploring costly paths with “lower” weights initially (e.g. **PATH1**), instead of focusing on the goal. We can “avoid” that by pruning the branches that will not give us an optimal path with some degree of certainty. So in summary, UCS guarantees the optimal path but may explore too many irrelevant nodes/paths.

A*

- It combines the cost to reach a node ($g(n)$) and the heuristic estimate to the goal ($h(n)$). It is guided efficiently toward the goal, prioritizing **PATH2** due to the lowest $g(n) + h(n)$ value. In essence A* will find the optimal path faster than UCS by leveraging the heuristic ($h(n)$).

Difference in Performance

DFS fails because it lacks cost-awareness and may follow deep but non-optimal paths. **UCS** fails because it lacks heuristic guidance, exploring irrelevant nodes/paths with low immediate costs. **A*** outperforms both because it balances between exploring the cheapest paths and the most goal-directed paths using ($f(n) = g(n) + h(n)$). The heuristic prevents unnecessary exploration, which saves us a lot of time especially for larger applications, therefore making it very scalable.

Files:

adjmatrix_cpsc4610.pdf
graph_cpsc4610.pdf
heuristic.pdf

Q3 (25 points) – Alpha Beta Pruning

Consider an initialized maxheap data structure ($\boldsymbol{A}[1]$ is the largest value; the second largest value is either $\boldsymbol{A}[2]$ or $\boldsymbol{A}[3]$. Recall, node A[k] has left child $\boldsymbol{A}[2\boldsymbol{k}]$ and right child $\boldsymbol{A}[2\boldsymbol{k}+1]$ and all subtrees adhere to the heap property). View this tree as a game tree (recall the meaning of $\boldsymbol{MAX}$ and $\boldsymbol{MIN}$ nodes)

i. Identify what nodes would not be pruned, regardless of the values of leaf nodes. **Explain why.**

ii. Assume $A[1], A[4] \ldots A[7]$ are MAX nodes, $A[2]$ and $A[3]$ are MIN nodes. Assign values to the MIN leaf nodes $A[8]\ldots A[15]$ so that the:
   1. minimum number of nodes would be pruned. Explain
   2. maximum number of nodes would be pruned. Explain

iii. Answer a) for a minheap data structure.

iv. Repeat b) as well: Assume $A[1], A[4] \ldots A[7]$ are MIN nodes, $A[2]$ and $A[3]$ are MAX nodes. Assign values to the MIN leaf nodes $A[8] \ldots A[15]$

A3

i. The nodes that must be evaluated to determine the outcome of the game, are the ones that would not get pruned.

Example (Tree with 7 nodes)

1. **Leaf Nodes** ($\boldsymbol{A}[2\boldsymbol{k}]$ & $\boldsymbol{A}[2\boldsymbol{k}+\mathbf{1}]$; $\forall \boldsymbol{k}$), Pretty much all leaf nodes at the bottom must be evaluated because their values are the basis for decisions made by their parent nodes. We will always evaluate the left most subtree (basis) but might prune the right most subtree

2. **Immediate Ancestors of the Leaf Nodes** ($\boldsymbol{A}[2]$ & $\boldsymbol{A}[3]$), These nodes are MIN nodes in the game tree. A MIN node must evaluate all of its children to find the smallest value to propagate upward. None of these evaluations can be skipped (pruned) because the MIN decision depends on the smallest value, and missing even one value could result in an incorrect decision

3. **Root Node** ($\boldsymbol{A}[\mathbf{1}]$), at each higher level, or this case the root, nodes whose decisions directly influence the final outcome at the root cannot be pruned

ii. PDF

Files:
hw1prunning.pdf

iii. Works the same as in a maxheap. The nodes that must be evaluated to determine the outcome of the game are the ones that would not get pruned.

Example (Tree with 7 nodes**)**

1. **Leaf Nodes** ($\boldsymbol{A[2k]}$ **&** $\boldsymbol{A[2k+1];\ \forall k}$**)**, Pretty much all leaf nodes at the bottom must be evaluated because their values are the basis for decisions made by their parent nodes. For example, we will always evaluate the left most subtree (basis) but might prune the right most subtree.

2. **Immediate Ancestors of the Leaf Nodes** ($\boldsymbol{A[2]}$ **&** $\boldsymbol{A[3]}$**)**, These nodes are MAX nodes in the game tree. A MAX cannot be pruned because they need to evaluate all their children (leaf nodes) to determine the largest value

3. **Root Path Nodes** ($\boldsymbol{A[1]}$**)**, nodes along the path from the root to the optimal leaf value cannot be pruned. This includes the root and an nodes that “directly” influence the root’s decision on the way up.

Q4 (30 points) – Expectimax

Read A. Danyluk's presentation. Consider the tree given in slide 13 (same structure used in slides 23, 24, 25) where the chance distribution is an even 50/50.

a) Identify the expected value for MAX, and explain its derivation.

b) **Without modifying leaf node values, try to** give altered values for the chance distribution to yield expected values for MAX as specified below. When it is not possible to do so, explain why
  1. A larger value
  2. A smaller value
  3. A negative value

c) **Without modifying leaf node values,** alter the $MIN - MAX$ roles (MIN becomes the root; MAX the interior nodes) and identify the expected value for MIN, and explain its derivation.

d) **Without modifying leaf node values, try to** give altered values for the chance distribution to yield expected values for MIN as specified below. When it is not possible to do so, explain why
  1. A larger value
  2. A smaller value
  3. A negative value

A4

a) The value of the root node (MAX) is **1**. Derivation can be found in the **PDF** below.

b) **PDF**

c) The value of the root node (MIN) is **5.5**. Derivation can be found in the **PDF** below.

- I think it's worth noting that the value of the root is implementation dependent in this case. Both options are equally optimal and both yield a result of 5.5

d) **PDF**

Files:
hw1expectimax.pdf

Adjacency Matrix Q2 (LaTeX)

Jakob Balkovec

2025-01-12

```
# no eval
```

Adjacency Matrix (in table form):

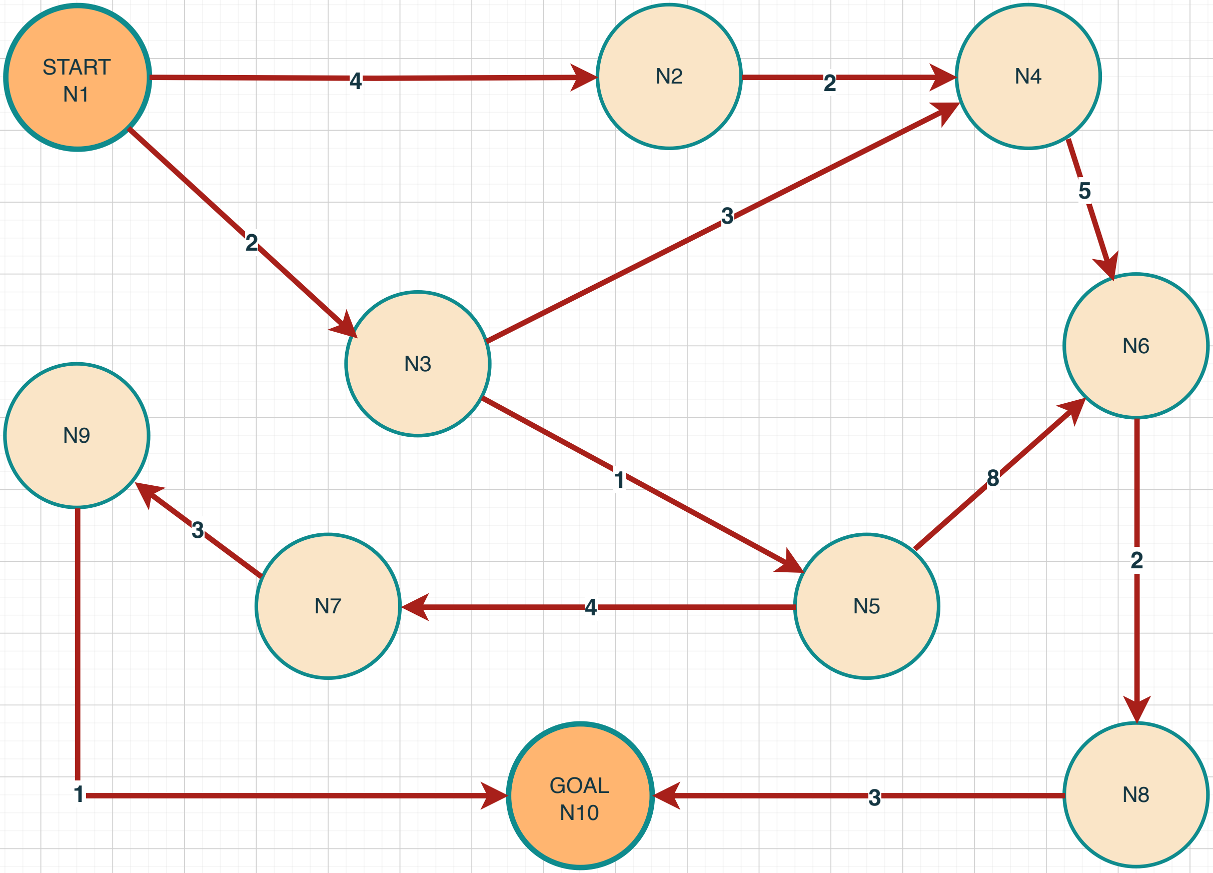
	$N1$	$N2$	$N3$	$N4$	$N5$	$N6$	$N7$	$N8$	$N9$	$N10$
$N1$	0	4	2	0	0	0	0	0	0	0
$N2$	0	0	0	2	0	0	0	0	0	0
$N3$	0	0	0	3	1	0	0	0	0	0
$N4$	0	0	0	0	0	5	0	0	0	0
$N5$	0	0	0	0	0	8	4	0	0	0
$N6$	0	0	0	0	0	0	0	2	0	0
$N7$	0	0	0	0	0	0	0	0	3	0
$N8$	0	0	0	0	0	0	0	0	0	3
$N9$	0	0	0	0	0	0	0	0	0	1
$N10$	0	0	0	0	0	0	0	0	0	0

Adjacency Matrix (in matrix form):

$$A = \begin{bmatrix} 0 & 4 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 8 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

HW1 Question 2 Graph

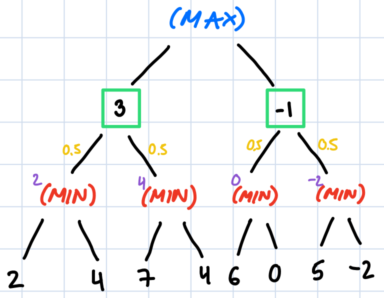
This is a graphical representation of the graph used throughout **Question 2**. The adjacency matrix, as well as the heuristic, can be found in the attachment (**Canvas | HW1.adjmatrix_cpsc4610**)



```
// CPCS 4610
// Jakob Balkovec
// HW1.Q2

// The heuristic needs to be admissible and consistent.
// - Admissible: should not overestimate the actual cost to the goal.
// - Consistent: must satisfy the triangle inequality, ensuring
//    the estimated cost from the current node to the goal is no greater
//    than the cost of reaching a neighbor plus the estimated cost from the neighbor
 to the goal.

// Pulled from: [https://en.wikipedia.org/wiki/Consistent_heuristic]

// Triangle inequality | Cons: h(n) <= c(n, n') + h(n')
// Admissible: h(n) <= h*(n)

// Idea: Use straight line distances or some logical esitmates to guide A*
//       effectively towards the goal while misleading DFS and UCS.

// Simple Pseudo Code:

function heuristic (current_node, goal_node):
  // Input:
  //  current_node: The current node in the graph.
  //  goal_node: The goal node in the graph.

  // Output:
  //  (int) The estimated cost to reach 'goal' from 'current'.

  // Straight line distance between two points
  // Euclidean distance, could use others like Manhattan or Chebyshev
  // Pulled from the powerpoint slides (Wednesday)
  return sqrt((current_node.x - goal_node.x)^2 + (current_node.y - goal_node.y)^2)

// Mock results:
// Abbreviated 'heuristic' to 'h' for brevity
// Nn = current_node, N10 = Goal state
//
// h(N1, N10) = 10
// h(N2, N10) = 8
// h(N3, N10) = 7
// h(N4, N10) = 6
// h(N5, N10) = 5
// h(N6, N10) = 4
// h(N7, N10) = 3
// h(N8, N10) = 2
// h(N9, N10) = 1
// h(N10, N10) = 1
```

HW 1

⇒ Q4.a



$P_2 = 0.5$

$P_4 = 0.5$

$P_0 = 0.5$

$P_{-2} = 0.5$

→ A[2]:

$= 0.5 \cdot 2 + 0.5 \cdot 4$

$= 3$

→ A[3]:

$= 0 \cdot 0.5 + (-2) \cdot 0.5$

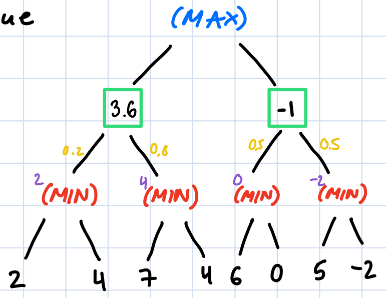
$= -1$

⇒ A[1] = 3

$\boxed{(MAX) = 3}$

⇒ Q4.b

Larger Value



$P_2 = 0.2$

$P_4 = 0.8$

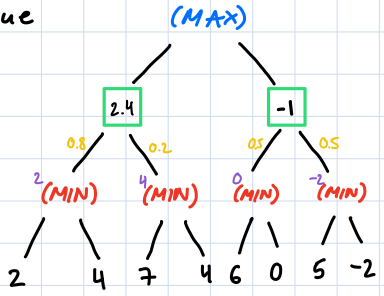
$P_0 = 0.5$

$P_{-2} = 0.5$

(MAX) = 3.6

↳ $3.6 > 3$ (MAX vs prev MAX)

Smaller Value



$P_2 = 0.8$

$P_4 = 0.2$

$P_0 = 0.5$

$P_{-2} = 0.5$

(MAX) = 2.4

↳ $2.4 < 3$ (MAX vs prev MAX)

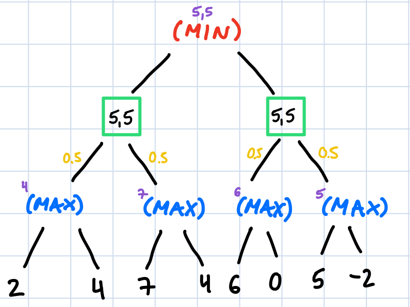
Negative Value

→ There exists no combinantion of x, y; where $x + y = 1$, that yields a negative number.

↳ In other words:

There do not exist $x, y \in [0,1]$ such that $x + y = 1$, and $(x \cdot a) + (y \cdot b)$, where $a, b \geq 0$ (in our case $a = 2$, $b = 4$)

⇒ Q4. c



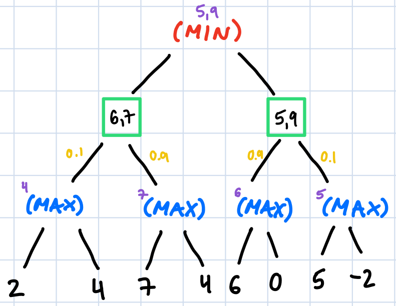
$P_4 = 0.5$
$P_7 = 0.5$
$P_6 = 0.5$
$P_5 = 0.5$

→ A[2]:
= 0.5 · 4 + 0.5 · 7
= 5,5

→ A[3]:
= 0.5 · 6 + 0.5 · 5
= 5,5

⇒ A[1] = 3
$\boxed{\text{(MIN)} = 5,5}$

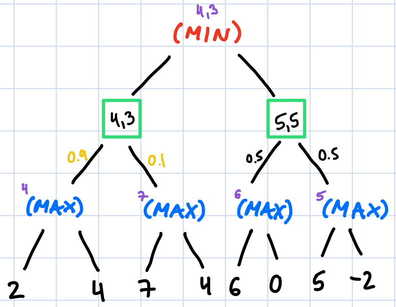
Larger value



$P_4 = 0.1$
$P_7 = 0.9$
$P_6 = 0.9$
$P_5 = 0.1$

(MIN) = 5.9
↳ 5.9 > 5.5 (MIN vs prevMIN)

Smaller value



$P_4 = 0.9$
$P_7 = 0.1$
$P_6 = 0.5$
$P_5 = 0.5$

(MIN) = 4,3
↳ 4,3 < 5,5 (MIN vs prevMIN)

Negative Value

→ All children of chance nodes are positive

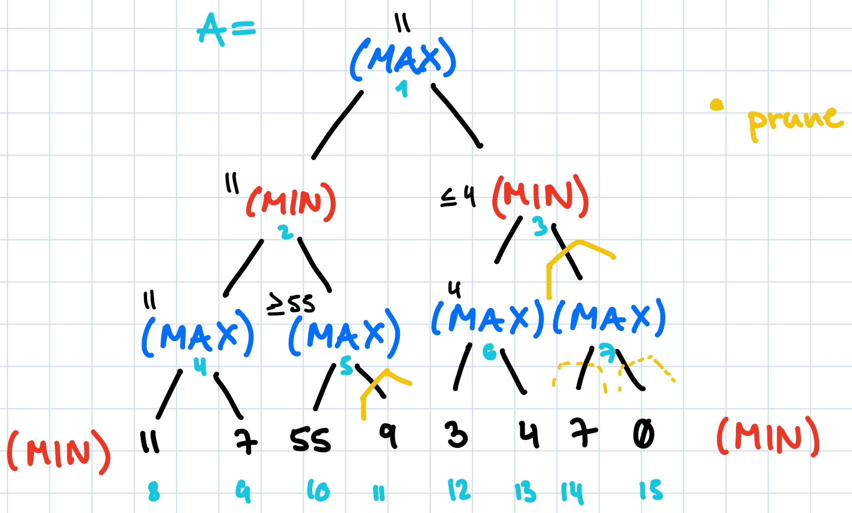
→ There exists no combinantion of x,y; where x+y = 1, that yields a negative number.

↳ In other words:

There do not exist $x,y \in [0,1]$ such that $x+y = 1$, and $(x \cdot a) + (y \cdot b)$, where $a,b \geq 0$ (in our case $a = 2$, $b = 4$)

HW 1
⇒ Q3.ii

Tree MAX pruning



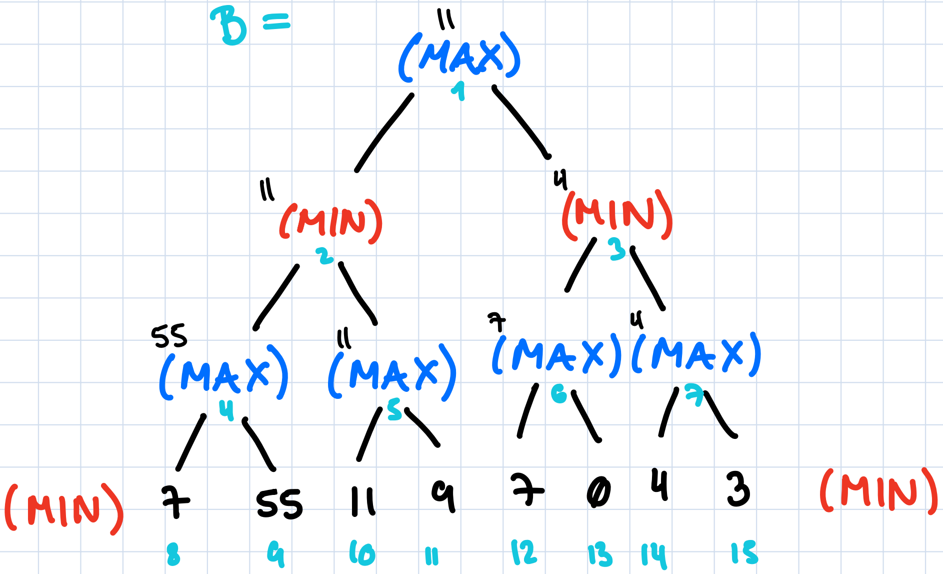
Pruned Nodes:

Node A[11] ; A[10] is used as max and since A[10] > A[4] , we prune A[11].
(MIN → A(2)

Node A[7]; A[6] is used as min and since A[1] is a max node it will pick A[2] over A[3], making A[7] useless /prunable

Σ: 2 (4 if we count the leaf nodes of A[7])

Tree MIN pruning

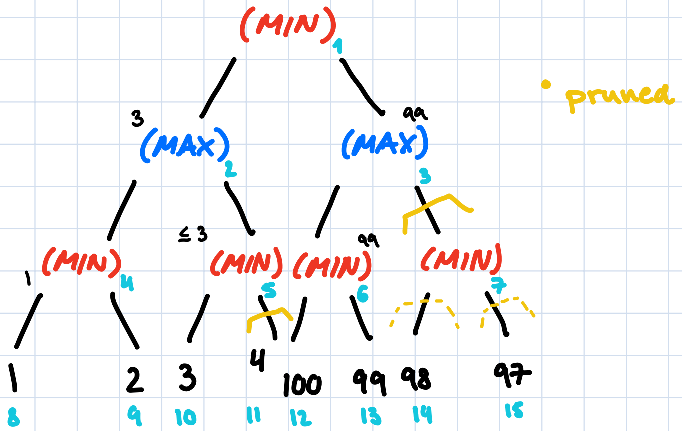


Pruned Nodes: None | []

$\boxed{\Sigma : 0}$

⇒ Q3. iv

Tree MAX pruning



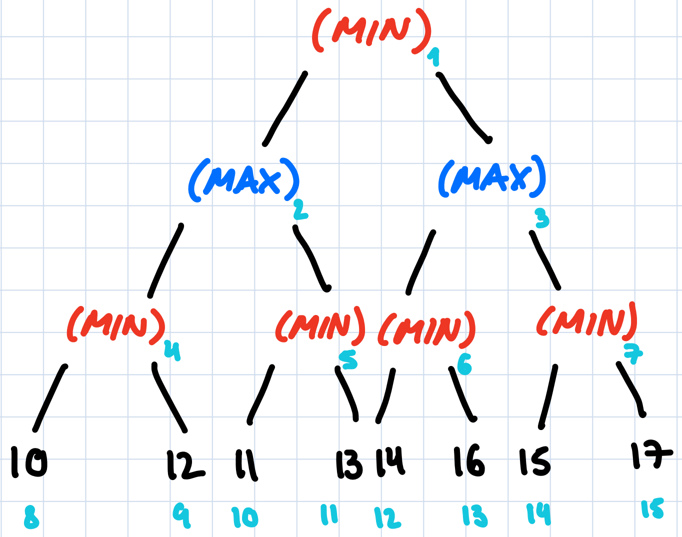
Pruned Nodes:

Node A[11]: Use A[10] as max for A[2] since A[10] > A[4]

Node A[7]: No point in exploring since A[1] would take A[2] over A[3], even if A[7] < A[6].

Σ: 2 (4 if we count the leaf nodes of A[7])

Tree MAX pruning



Pruned Nodes: None ([])

Σ : 0