A Lower Bounding Technique

- A bit Game Theory
- Yao's Minimax Principle
- Application: Game Tree Evaluation

A Game

	Scissors	Paper	Stone
Scissors	0	1	-1
Paper	-1	0	1
Stone	1	-1	0

Game Theory

- A game as this one is called *two-person zero-sum game*.
- It is called zero-sum because the net amount won of a player is always exactly 0, i.e. if one player gain c in a game, the other player loses c (gains $-c$).
- The matrix is called the *Payoff Matrix*.
- Every two-person zero-sum game can be represented by a payoff matrix.

Basics

- Let R be the row player, C be the column player and M be the matrix.
- If R chooses strategy i, then he is guaranteed a payoff of $\min_j M_{ij}$.
- An *optimal strategy* for R is an i that maximizes $\min_j M_{ij}$.
- Let $V_R = \max_i \min_j M_{ij}$ and $V_C = \min_j \max_i M_{ij}$.
- Claim: $V_R \leq V_C$.
- If $V_R = V_C$, than the game has a *solution* and the *value* of the game is $V = V_R = V_C$.

A modified game

	Scissors	Paper	Stone
Scissors	0	1	2
Paper	-1	0	1
Stone	-2	-1	0

von Neumanns' Minimax Theorem

For any two-person zero-sum game specified by a matrix M

$$\max_p \min_q p^T M q = \min_q \max_p p^T M q.$$

- The largest expected payoff that R can guarantee is equal to the smallest expected payoff C can guarantee
- This common expected payoff is call the value of the game V.

Randomization

- A *randomized* or *mixed* strategy if a probability distribution on the set of possible strategies.
- The row player picks a vector $p = (p_1, \ldots, p_n)$, a probability distribution as strategy.
- Analogously the column player picks a vector $q = (q_1, \ldots, q_m)$.
- The payoff is a random variable, and its expectation is given by

$$E[\text{payoff}] = p^T M q = \sum_{i=1}^{n} \sum_{j=1}^{m} p_i M_{ij} q_j$$

Application for Lower Bounding

- A randomized algorithm can be viewed as a probability distribution over deterministic algorithms:
  Fix all random bits generated in the algorithm in advance.
- The algorithm designer plays the role of the column player
- The columns correspond to the possible deterministic algorithms $\mathcal{A}$.
- The rows correspond to the possible instances $\mathcal{I}$
- An entry in the matrix C represents the running time (or something else we are interested in) of this particular algorithm an this particular instance.

Application for Lower Bounding

- Once the probability distribution p is fixed, the expected running time is given by:

$$\max_{I \in \mathcal{I}} E[C(A_p, I)].$$

- In order to show a lower bound f, we have to show

$$\forall p : \mathcal{A} \mapsto [0,1] : \exists I \in \mathcal{I} : E[C(A_p, I)] \geq f$$

Yao's Minimax Principle

- In order to show a lower bound of f on the cost of all randomized algorithms in $\mathcal{A}$, one needs to show only

$$\exists q : \mathcal{I} \mapsto [0,1] : \forall A \in \mathcal{A} : E[C(A, I_q)] \geq f$$

- The advantage lies
  - in the flexibility in the choice of q and
  - you don't have to argue over all probability distributions but over deterministic algorithms over a fixed distribution q.
- We will see an application in the next section.

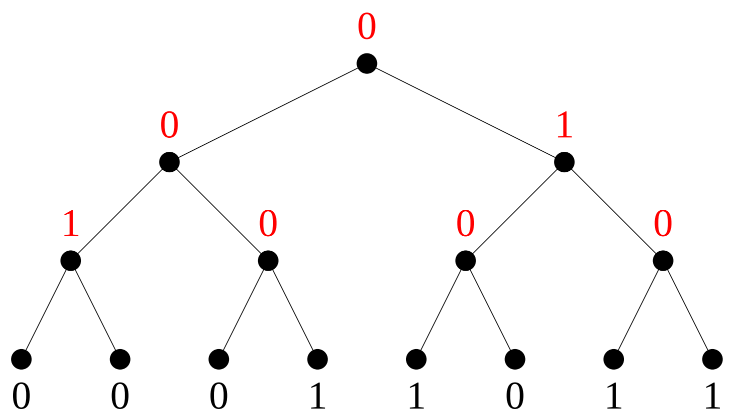
Yao's Minimax Principle

Yao's Lemma:
For all distributions p over $\mathcal{A}$ and q over $\mathcal{I}$

$$\max_{I \in \mathcal{I}} E[C(A_p, I)] \geq \min_{A \in \mathcal{A}} E[C(A, I_q)],$$

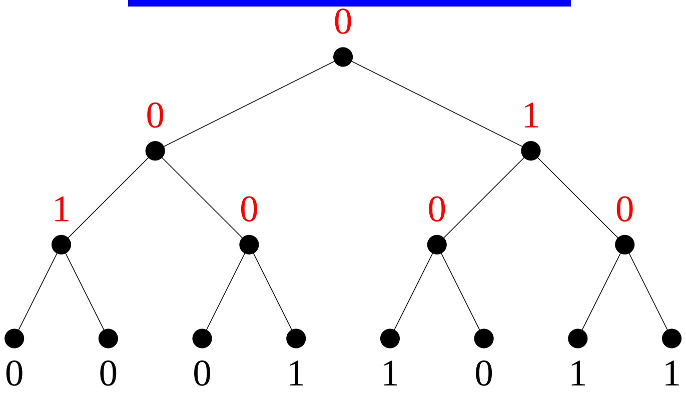
provided $E[C(A_p, I)]$ is finite for all $I \in \mathcal{I}$.

Tree Evaluation



- Input: $n = 2^k$ bits for k even, the values of the leaves
- Value of a node: NOR of the bits of the children
- Value of the tree: Value at the root

Randomized Algorithm



1. Pick a child at random.
2. Recursively evaluate this child.
3. If the value of this child is 1 return 0
4. Otherwise evaluate other child and return the correct value.

Running Time of the Algorithm

We want to derive a recurrence for the cost $C(h)$ of evaluating a tree of height h.

Case 2: Assume the value of v is 1.

- Both children have value 0.
- Each child has expected cost $C(h-2) + 1/2 \cdot C(h-2)$
- Thus $C(h) \leq 2 \cdot 3/2 \cdot C(h-2) = 3C(h-2)$.

Clearly $C(0) = 1$ and hence
$C(h) \leq 3^{h/2} = 3^{(\log_2 n)/2} = 3^{\log_4 n} = n^{\log_4 3}$

Running Time of the Algorithm

We want to derive a recurrence for the cost $C(h)$ of evaluating a tree of height h.

Case 1: Assume the value of v is 0.

- At least one child has value 1
- With probability at least $1/2$ we evaluate only one child.
- The expected cost of the evaluation of each child can be estimated by $2C(h-2)$.
- Thus $C(h) \leq 2C(h-2) + 1/2 \cdot 2C(h-2)$

A Lower Bound

- We have to fix a probability distribution over the input:
  We set the value of a leave to 1 with probability $r = (3-\sqrt{5})/2$.
- The probability that the NOR operation returns a 1 is
  $(1-r)^2 = ((\sqrt{5}-1)/2)^2 = (3-\sqrt{5})/2 = r$.
- **Claim:** Let T be a NOR-Tree whose leaves are independently set to 1 with a certain probability c. If there is a deterministic algorithm that runs in expected cost $W(T)$, than there is a depth-first pruning algorithm with expected running time is at most $W(T)$.
- Thus we only consider depth-first pruning algorithms.

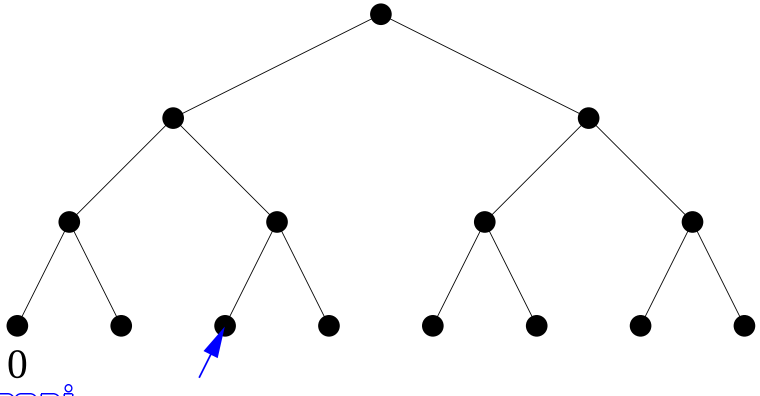
A Lower Bound

- Every algorithm has to evaluate at least one child.
- The other child has to be evaluated if the first child returns 0.
- Thus, $W(h) \geq W(h-1) + (1-r) \cdot W(h-1)$.
- The solution of this recurrence is $W(h) \geq (2-r)^h$.
- Substituting h with $\log_2 n$ yields a lower bound of $n^{0.694\ldots}$.
- The upper bound was $n^{\log_4 3} = n^{0.792\ldots}$.
- A harder analysis shows that the algorithm is optimal.

Deterministic Evaluation

Claim: Any deterministic alg. has worst case running time at least n.

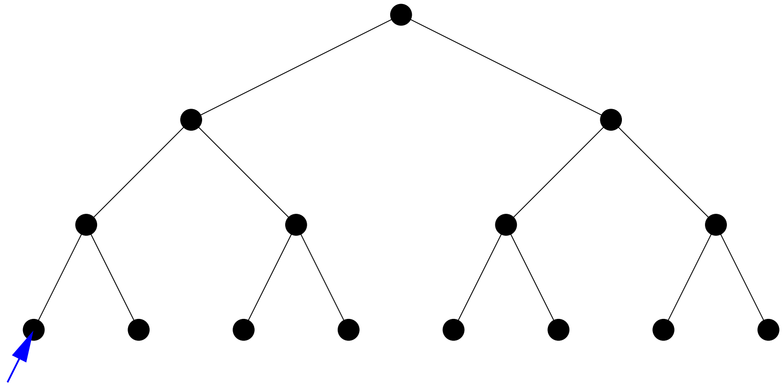
- Enforce the algorithm to evaluate both children to evaluate a node.
- We make the value of the children that is evaluated first 0.



Deterministic Evaluation

Claim: Any deterministic alg. has worst case running time at least n.

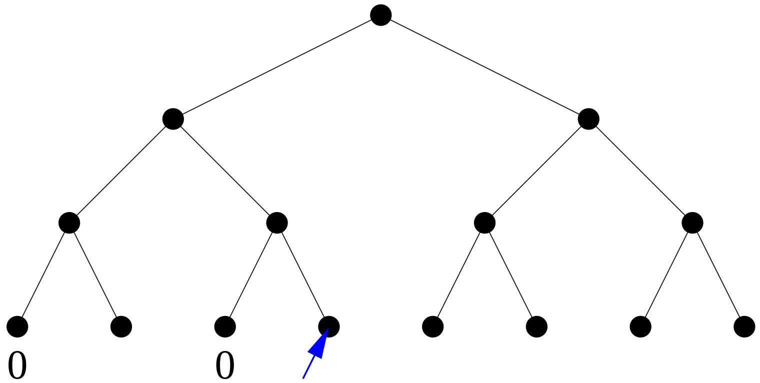
- Enforce the algorithm to evaluate both children to evaluate a node.
- We make the value of the children that is evaluated first 0.



Deterministic Evaluation

Claim: Any deterministic alg. has worst case running time at least n.

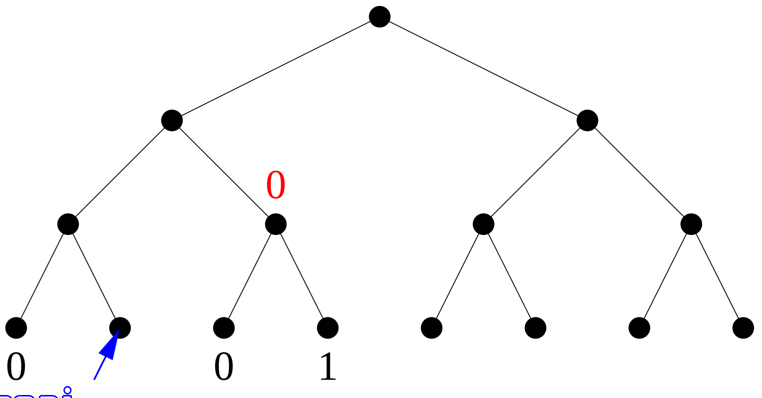
- Enforce the algorithm to evaluate both children to evaluate a node.
- We make the value of the children that is evaluated first 0.



Deterministic Evaluation

Claim: Any deterministic alg. has worst case running time at least n.

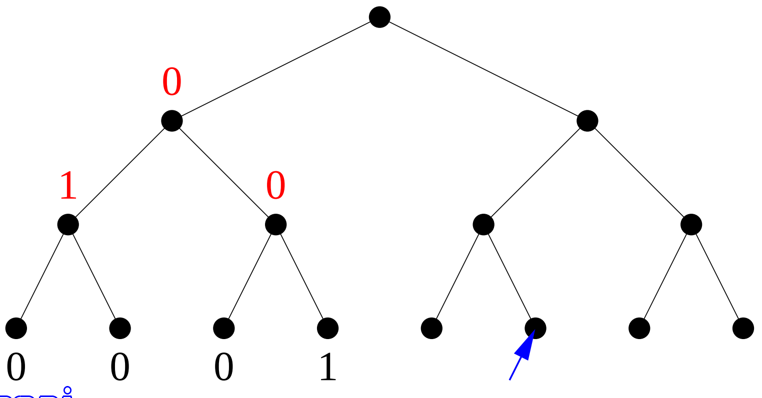
- Enforce the algorithm to evaluate both children to evaluate a node.
- We make the value of the children that is evaluated first 0.



Deterministic Evaluation

Claim: Any deterministic alg. has worst case running time at least n.

- Enforce the algorithm to evaluate both children to evaluate a node.
- We make the value of the children that is evaluated first 0.



Summary

- We have seen a lower bounding technique with an application.
- We have seen an example of a problem, where we can proof that our randomized algorithm is faster than any deterministic algorithm.