02249 - Computationally Hard Problems, Assignment VII

Arianna Taormina, S163671

16 November 2020

Exercise Description

Consider the scenario underlying the problem GameTreeEvaluation from the lecture, but assume that the tree is a complete Quaternary (4-ary) one of depth 2k instead of a binary. Propose a modification of Algorithm 5.17 (randomized evaluation) for this type of game trees. Bound the expected number of leaves evaluated by the algorithm by some value that is lower than the total number of leaves.

Binary GameTree Properties

- the labels are from {0,1}
- all leaves are at level 2k(which implies that the depth of the tree is 2k)
- every internal node has exactly 2 children
- Such a game tree has $N = 2^{2k}$ leaves.

Problem 5.16 [GameTreeEvaluation]
Input: A game tree with the properties just stated and an assignment of {0,1} -labels to the leaves.
Output: The label of the root.

Algorithm 5.17 [Rand Eval]

```
1: $v \leftarrow$ root
2: result$\leftarrow$ evaluate ($v$)
3: proc evaluate ($v$)
4: if $v$ is a leaf then   return ($\ell(v)$)
5: else
6:     let $w_1$ and $w_2$ be the children of $v$ pick one child with probability 1/2 at random; call this $a$ and the other $b$
7:     $t \leftarrow$ evaluate ($a$)
8:     if ($v$ is max-node ) $\wedge$ ($t == 1$) then return (1)
9:     else if ($v$ is min-node ) $\wedge$ ($t == 0$)  then return (0)
10:    else return ( evaluate ($b$)) =0
```

Student note. I have tried to solve the problem in practice and implement the code I tried to describe here below. At the end there are some results from the above mentioned code. I also used quite some time to get into the demonstration of the upper bound. I hope it is correct, but I have to say it got difficult to calculate the probabilities recursively. Overall the results seem to make sense,as that with random choice the number of expected leafs to be examined is lower then the total. Also depending on the number of good matches in the various cases the upper bounds seemed to make sense (the higher the number of matches, the lower is the expected number of searches).

Exercise Resolution

Quaternary GameTree Properties

- the labels are from {0,1}
- all leaves are at level 2k(which implies that the depth of the tree is 2k)
- every internal node has exactly 4 children
- Such a game tree if complete has $N = 4^{2k} = 16^k$ leaves.

Algorithm 5.17 can be modified as follows.

Algorithm 5.17 [Rand Eval Mod]

```
$v \leftarrow$ root
result$\leftarrow$ evaluate ($v$)
proc evaluate ($v$)
**if** $v$ is a leaf **then** **return** ($\ell(v)$)
**else**
    $labChildren \leftarrow$ find labelled Children
    $0Children \leftarrow$ number of children with label 0
    $1Children \leftarrow$ number of children with label 1
    **if** ($v$ is P1-node ) $\wedge(1Children > 0)$ **thenreturn**(1)
    **else if** ($v$ is P2-node ) $\wedge(0Children > 0)$ **then return**(0)
    **else {**
    $unlabChildren \leftarrow$ find unlabelled Children
    analyze children $\leftarrow unlabChildren$
    **if** if the list is not empty **then**
        $a \leftarrow$ pickrandom ($unlabChildren$)
        $a \leftarrow$ drop from ($unlabChildren$)
        $t \leftarrow$ evaluate ($a$)
        **if** ($v$ is P1-node ) $\wedge$ ($t == 1$) **then return** (1)
        **else if** ($v$ is P2-node ) $\wedge$ ($t == 0$) **then return** (0)
        **else** analyze children $\leftarrow unlabChildren$
     **else** if the list is empty
          **if** ($v$ is P1-node ) **return** (0)
          **else** **if** ($v$ is P2-node ) **then return** (1)
        } =0
```

The depths alternate between Player 1 and Player 2.
Each depth of P1 has as a satisfying condition that at least one child is labelled as "1".
Each depth of P2 has as a satisfying condition that at least one child is labelled as "0".
We define a "match" when a child node satisfies the condition of its parent depth.
First the node is checked against the list of leaves of the tree, if it is a leaf then the label is given, otherwise we need to look at its children.
Let $labChildren$ be a list with all the labelled children of v with $i_{max} = 4$.
We count how many children are marked as "1" and how many children are marked as "0", it might also be that no child is marked at all yet.
If node v is at P1 depth, meaning that in the game is the P1 turn to move, if we find at least one child marked as "1", we mark the node as "1".
If node v is at P2 depth, meaning that in the game is the P2 turn to move, if we find at least one child marked as "0", we mark the node as "0".
If none of this cases sussists, we continue as follows.
Let $unlabChildren = \{w_1 ... \; w_i\}$ be a list with all the unlabelled children of v with $i_{max} = 4$. If the list is not empty, then pick one child of the unlabelled children with probability $1/i$ at random; call it a and then drop it from the list $unlabChildren$. Evaluate a, if it is a match, label the parent node accordingly, otherwise, recursively, analyze the remaining unlabelled children again choosing at random. If all the children are labelled and no match has been found the node at depth P1 is given to P2 (marked "0") and viceversa (marked "1").

The scope of the Algorithm is to be able to label the root node with an approach that is not exhaustive, meaning without labelling all the nodes,skipping some searches.

Evaluate expected number of leaves

Let T be a Quaternary game tree of depth $2k$ and let $M(T)$ be the expected number of leaves that algorithm Evaluate$_randm_odlooksat.Let$M$_{\max}(k) := \max\{M(T) \mid T$ is a game tree as above of depth $2k.\}$

For all game trees as above with depth $2k, M(T) \leq N^{0.807}$ and $M_{\max}(k) \leq N^{0.807}$ The proof is by induction on k and shows that $M_{\max}(k) \leq 7^k$. The claim then follows as

$$7^k = 2^{k\log_2(7)} = 16^{(k/4)\log_2(7)} = \left(4^k\right)^{\log_2(7)/4} = N^{\log_2(7)/4} = N^{0.807}$$

step 0 The induction starts with $k \equiv 0$. Then the tree T is just a leaf. Then $M(T) = 1 \leq 7^0$.
step 1 In this step of the induction, we consider two levels of the tree, the top-most being a P1-level(depth $2k+2$.), the second a P2-level (depth $2k+1$.) We want to consider the situation with 16 depth- $2k$ sub-subtrees $T_1, ...T_{16}$ The whole tree T has depth $2k+2$.
To evaluate P1, we need first upper bound from the P2 nodes at depth $2k+1$.

P2 depth

Consider a tree T of depth $2k+1$ with a P2-node v at the root. Node v has 4 subtrees, all depth 2k. 5 cases are possible.

1. case: all the subtrees are marked "0", None marked "1"
2. case: one of the subtrees is marked "0", the rest marked "1"
3. case: two of the subtrees are marked "0", the rest marked "1"
4. case: three of the subtrees are marked "0", the rest marked "1"
5. case: None of the subtrees is marked "0", all marked "1"

Lets look at the single cases, note that assigning label to one tree or the other is done without loss of generality.
Case 1: If all subtrees have a "0" label.
T1 is chosen with probability 1/4 to be evaluated first; same for T2,T3,T4. Independently of which tree is chosen, its value 0 determines labeling of the parent node P2. The other sub-trees of P2 are not evaluated. Hence, the chances to have the time M(T1) for evaluation T1 is 1/4, same for T2,T3,T4.

$$M(T) = \frac{1}{4}M\left(T_1\right) + \frac{1}{4}M\left(T_2\right) + \frac{1}{4}M\left(T_3\right) + \frac{1}{4}M\left(T_4\right) \leq M_{\max}(k)$$

Case 5: all the subtrees have label "1". Then all of them must be evaluated to exclude the possibility of finding the "0". in this only case the parent node is going to be labelled "1".

$$M(T) = M\left(T_1\right) + M\left(T_2\right) + M\left(T_3\right) + M\left(T_4\right) \leq 4M_{\max}(k)$$

Case 2: one of the subtrees is marked "0", the rest marked "1".(T1 is marked "0")
T1 with label "0" is chosen with probability 1/4 and only that one has to be evaluated.
One of the other trees with label 1 is chosen first with probability 3/4 .
Lets assume T2 is evaluated first,its label is 1, we still have to evaluate T1, in addition to determine the parent label to be 0. Now T1 has probability 1/3 to be chosen and evaluated, whereas T3 and T4 have probability 2/3 to be chosen next.

Lets assume T3 is evaluated second,its label is 1, we still have to evaluate T1, in addition to determine the parent label to be 0. Now T1 has probability 1/2 to be chosen and evaluated,same as T4.

Lets assume T4 is evaluated third,its label is 1, we still have to evaluate T1, in addition to determine the parent label to be 0. Now T1 has probability 1 to be chosen and evaluated

$$M(T) = \frac{1}{4}M(T_1) + \frac{3}{4}\left(M(T_2) + \frac{1}{3}M(T_1) + \frac{2}{3}\left(M(T_3) + \frac{1}{2}M(T_4) + \frac{1}{2}M(T_1)\right)\right) \leq 2.25M_{\max}(k)$$

Case 3: two of the subtrees are marked "0", the rest marked "1" (T1,T2 are marked "0").
A tree with label "0" is chosen with probability 1/2 and only that one has to be evaluated.
One of the other trees with label 1 is chosen first with probability 1/2 .
Lets assume T3 is evaluated first,its label is 1, we still have to evaluate either T1 or T2, in addition to determine the parent label to be 0. Now a tree labelled with "0" has probability 2/3 to be chosen and evaluated, whereas T4 has probability 1/3 to be chosen next.

Lets assume T4 is evaluated second,its label is 1, we still have to evaluate either T1 or T2, in addition to determine the parent label to be 0. Now either T1 or T2 is going to be chosen and evaluated with probability 1.

$$M(T) = \frac{1}{2}M(T_{label0}) + \frac{1}{2}\left(M(T_3) + \frac{2}{3}M(T_{label0}) + \frac{1}{3}\left(M(T_4) + M(T_{label0})\right)\right) \leq 1.67M_{\max}(k)$$

Case 4: three of the subtrees are marked "0", the rest marked "1" (T1,T2,T3 are marked "0").
A tree with label "0" is chosen with probability 3/4 and only that one has to be evaluated.
One of the other trees with label 1 is chosen first with probability 1/4 .
Lets assume T4 is evaluated first,its label is 1, we still have to evaluate either T1 or T2 or T3, in addition to determine the parent label to be 0. Now a tree labelled with "0" has probability 1 to be chosen and evaluated.

$$M(T) = \frac{3}{4}M(T_{label0}) + \frac{1}{4}\left(M(T_4) + M(T_{label0})\right) \leq 1.25M_{\max}(k)$$

Altogether :

- Fact 1 If the label of v is 0, the time is at most 1.67 Mmax(k)
- Fact 2 If the label of v is 1, then 4 Mmax(k) are sufficient.

P1 depth

Consider a tree T of depth $2k + 2$ with a P1-node v at the root. Node v has 4 subtrees, all depth 2k+1. 5 cases are possible

1. case: all the subtrees are marked "0", None marked "1"
2. case: one of the subtrees is marked "1", the rest marked "0"
3. case: two of the subtrees are marked "1", the rest marked "0"
4. case: three of the subtrees are marked "1", the rest marked "0"
5. case: None of the subtrees is marked "1", all marked "0"

Case 1: If all subtrees have a "0" label.
If all P2 nodes are labelled as 0, the algorithm evaluates all. By fact 1 we have:

$$M(T) \leq 1.67M_{\max}(k) + 1.67M_{\max}(k) + 1.67M_{\max}(k) + 1.67M_{\max}(k) = 6.68M_{\max}(k)$$

Case 5: all the subtrees have label "1". Then only one node is evaluated, by Fact2 we have:

$$M(T) \leq 4M_{\max}(k)$$

Case 2: one of the subtrees is marked "1", the rest marked "0". (T1 is marked "1"). T1 is chosen at first with probability 1/4, then in expectation $4M_{\max}(k)$ leaves are considered from its children.

If T2 marked ”0” is chosen first, with probability 3/4 then by Fact 1 and Fact 2 in expectation we need to evaluate all the leafs of T2 and then we will still have to evaluate T1. T1 will then have a probability to be chosen equal to 1/3, whereas T3 or T4 can be chosen with probability 2/3.

If T3 marked ”0”is chosen second, then we will need again to evaluate all the leaves in T3 and we will still have to evaluate T1. T1 will then have a probability to be chosen equal to 1/2, as T4.

If T4 marked ”0” is chosen third, then we will need again to evaluate all the leaves in T4 and we will still have to evaluate T1. T1 will then have a probability to be chosen equal to 1.

$$M(T) \leq \frac{1}{4}4M_{\max}(k) + \frac{3}{4}\left(1.67M_{\max}(k) + \frac{1}{3}4M_{\max}(k)) + \frac{2}{3}\left(1.67M_{\max}(k)) + \frac{1}{2}1.67M_{\max}(k)) + \frac{1}{2}4M_{\max}(k))\right)\right) \leq 5.05M_{\max}(k)$$

Case 3: two of the subtrees are marked ”1”, the rest marked ”0”. (T1,T2 marked ”1”). T1 or T2 is chosen at first with probability 1/2, then in expectation $4M_{\max}(k)$ leaves are considered from its children.

If T3 marked ”0” is chosen first, with probability 1/2 then by Fact 1 and Fact 2 in expectation we need to evaluate all the leafs of T3 and then we will still have to evaluate T1 or T2. T1or T2 will then have a probability to be chosen equal to 2/3, whereas T4 can be chosen with probability 1/3.

If T4 marked ”0”is chosen second, then we will need again to evaluate all the leaves in T4 and we will still have to evaluate T1 or T2. T1 or T2 will then have a probability to be chosen equal to 1.

$$M(T) \leq \frac{1}{2}4M_{\max}(k) + \frac{1}{2}\left(1.67M_{\max}(k) + \frac{2}{3}4M_{\max}(k) + \frac{1}{3}\left(1.67M_{\max}(k) + 4M_{\max}(k)\right)\right) \leq 5.11M_{\max}(k)$$

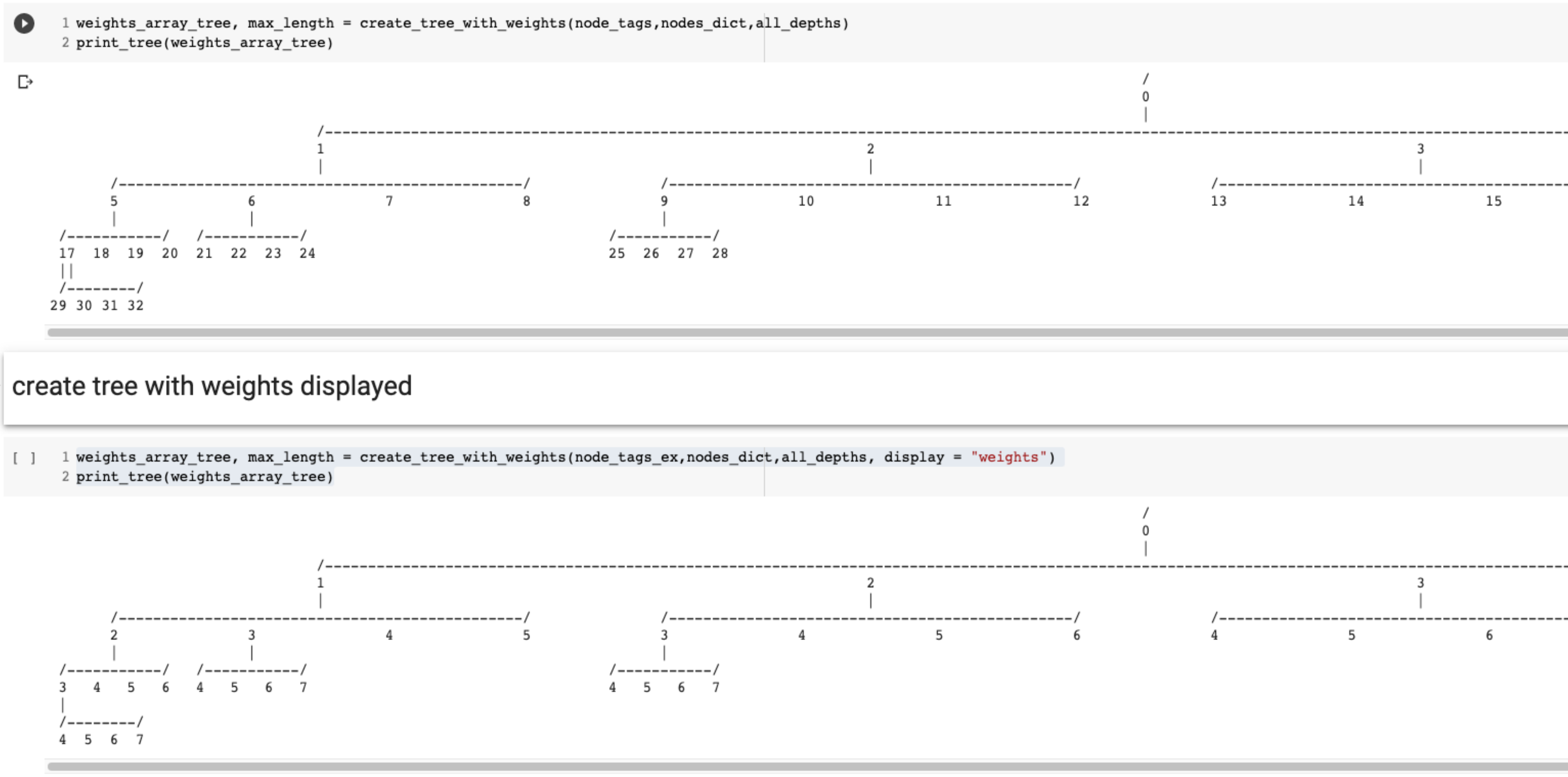
Case 4: three of the subtrees are marked ”1”, the rest marked ”0”. (T1,T2,T3 marked ”1”). T1 or T2 or T3 is chosen at first with probability 3/4, then in expectation $4M_{\max}(k)$ leaves are considered from its children.

If T4 marked ”0” is chosen first, with probability 1/4 then by Fact 1 and Fact 2 in expectation we need to evaluate all the leafs of T4 and then we will still have to evaluate T1 or T2 or T3. T1 or T2 or T3 will then have a probability to be chosen equal to 1. (1.67 $\mathrm{M}_{\max}(k))4M_{\max}(k)$

$$M(T) \leq \frac{3}{4}4M_{\max}(k) + \frac{1}{4}\left(1.67M_{\max}(k) + 4M_{\max}(k)\right) \leq 4.41M_{\max}(k)$$

Conclusion In any case, we have $M(T) \leq 7M_{\max}(k)$. Now the induction hypothesis is that $M(k) \leq 7^k$. Using this, we derive

$$M_{\max}(k+1) \leq 7M_{\max}(k) \leq 7 \cdot 7^k = 7^{k+1}$$



```
weights_array_tree, max_length = create_tree_with_weights(node_tags,nodes_dict,all_depths)
print_tree(weights_array_tree)
```

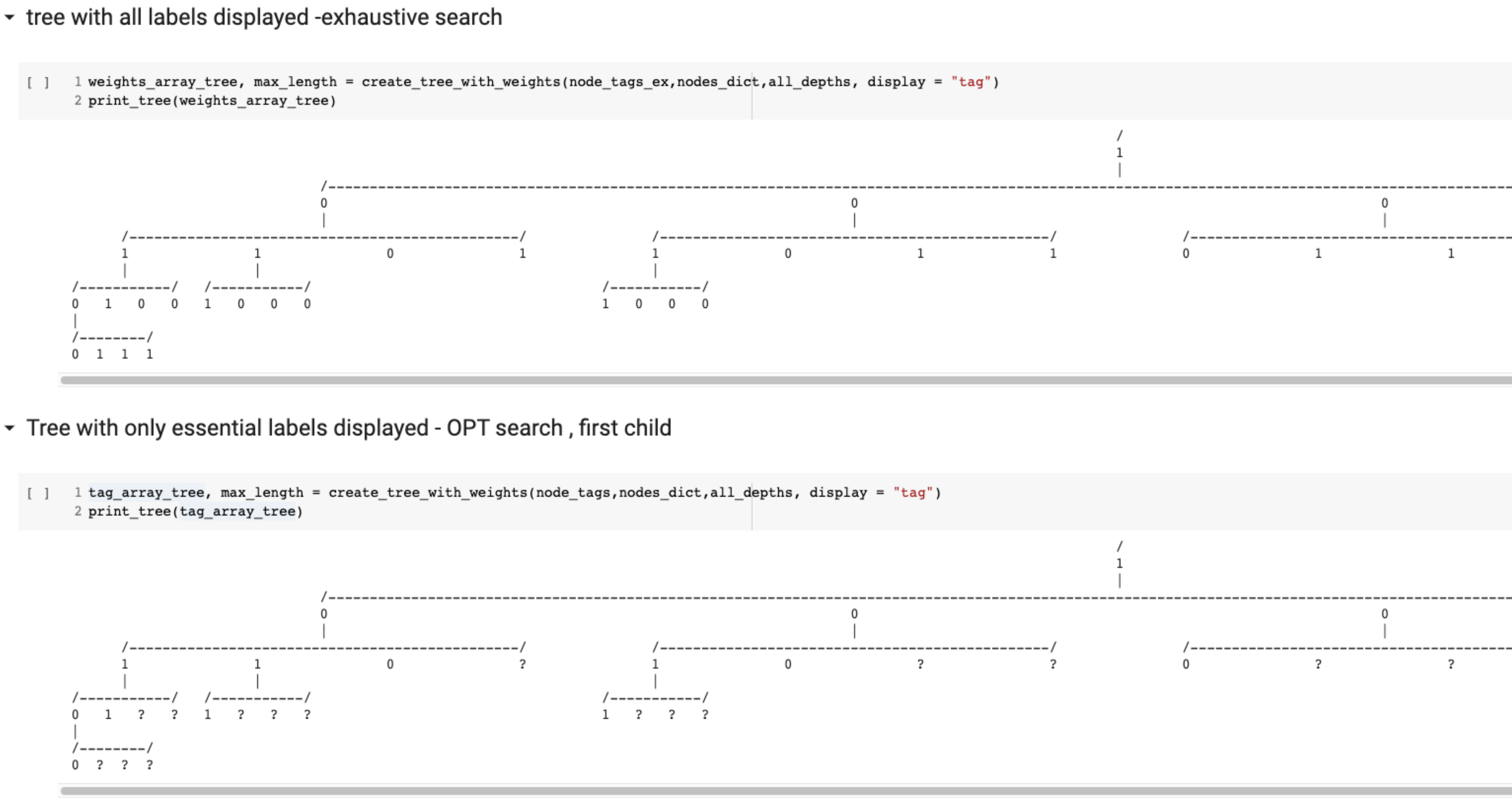
```
                                                                                                                               /
                                                                                                                               0
                                                                                                                               |
                               /-------------------------------------------------------------------------------------------------------------------------------------------------
                               1                                                               2                                                               3
                               |                                                               |                                                               |
       /------------------------------------------------/              /------------------------------------------------/              /-----------------------------------------
       5               6               7               8               9               10              11              12              13              14              15
       |               |                                               |
 /-----------/  /-----------/                                  /-----------/
 17  18  19  20  21  22  23  24                                  25  26  27  28
 ||
 /--------/
29 30 31 32
```

create tree with weights displayed

```
weights_array_tree, max_length = create_tree_with_weights(node_tags_ex,nodes_dict,all_depths, display = "weights")
print_tree(weights_array_tree)
```

```
                                                                                                                               /
                                                                                                                               0
                                                                                                                               |
                               /-------------------------------------------------------------------------------------------------------------------------------------------------
                               1                                                               2                                                               3
                               |                                                               |                                                               |
       /------------------------------------------------/              /------------------------------------------------/              /-----------------------------------------
       2               3               4               5               3               4               5               6               4               5               6
       |               |                                               |
 /-----------/  /-----------/                                  /-----------/
 3   4   5   6   4   5   6   7                                   4   5   6   7
 |
 /--------/
 4  5  6  7
```

Figure 1: QuaternaryTree of Game4, ID displayed (first) weights displayed (second)



tree with all labels displayed -exhaustive search

```python
weights_array_tree, max_length = create_tree_with_weights(node_tags_ex,nodes_dict,all_depths, display = "tag")
print_tree(weights_array_tree)
```

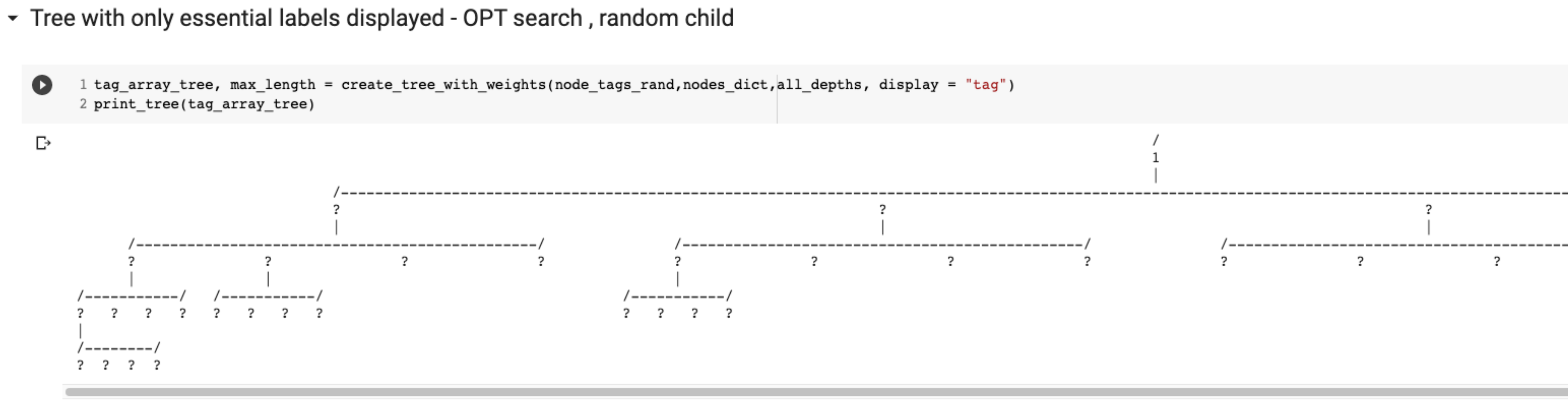
```
                                                                                                                              /
                                                                                                                              1
                                                                                                                              |
                              /-------------------------------------------------------------------------------------------------------------------------------------------------
                              0                                                               0                                                               0
                              |                                                               |                                                               |
      /-----------------------------------------------/               /-----------------------------------------------/               /---------------------------------------
      1               1               0               1               1               0               1               1               0               1               1
      |               |                                               |
/-----------/   /-----------/                                   /-----------/
0   1   0   0   1   0   0   0                                   1   0   0   0
|
/--------/
0  1  1  1
```

Tree with only essential labels displayed - OPT search , first child

```python
tag_array_tree, max_length = create_tree_with_weights(node_tags,nodes_dict,all_depths, display = "tag")
print_tree(tag_array_tree)
```

```
                                                                                                                              /
                                                                                                                              1
                                                                                                                              |
                              /-------------------------------------------------------------------------------------------------------------------------------------------------
                              0                                                               0                                                               0
                              |                                                               |                                                               |
      /-----------------------------------------------/               /-----------------------------------------------/               /---------------------------------------
      1               1               0               ?               1               0               ?               ?               0               ?               ?
      |               |                                               |
/-----------/   /-----------/                                   /-----------/
0   1   ?   ?   1   ?   ?   ?                                   1   ?   ?   ?
|
/--------/
0  ?  ?  ?
```

Figure 2: Labelling with exhaustive search(first), with fixed choice of child (second)



Tree with only essential labels displayed - OPT search , random child

```python
tag_array_tree, max_length = create_tree_with_weights(node_tags_rand,nodes_dict,all_depths, display = "tag")
print_tree(tag_array_tree)
```

```
                                                                                                                              /
                                                                                                                              1
                                                                                                                              |
                              /------------------------------------------------------------------------------------------------------------------------------------------------
                              ?                                                               ?                                                               ?
                              |                                                               |                                                               |
      /-----------------------------------------------/               /-----------------------------------------------/               /----------------------------------------
      ?               ?               ?               ?               ?               ?               ?               ?               ?               ?               ?
      |               |                                               |
/-----------/   /-----------/                                   /-----------/
?   ?   ?   ?   ?   ?   ?   ?                                   ?   ?   ?   ?
|
/--------/
?  ?  ?  ?
```

Figure 3: Labelling with randomized algorithm

∞