CS 577: Introduction to Algorithms	Program 7 - Game Tree
Out: 03/23/21	Due: 03/30/21

1 Coding Question:

Reminders

- Must be coded individually in your choice of either Python, Java, C, C++, or C#
- Submitted through Gradescope, and there are hidden test cases
- There is a class-wide runtime leaderboard on Gradescope
- We encourage the use of Piazza for debugging help
- Please do not cheat

Problem

You are given a full binary tree with n leaves that have either the value 1 or the value 0. You are playing the following game with your friend. You get to play first and decide whether to go to the left or the right subtree of the root and then your friend plays and decides whether to continue to the left or right subtree. You continue playing like that until you reach a leaf node. You win when you the game ends on a leaf node with value 1. Equivalently you may assume that each node of the binary tree is either a "max"/"or" node or a "min"/"and" node, see Figure 1. By $\vee$ we denote $\max$ or the "or" binary operator and by $\wedge$ we denote $\min$ or "and". In what follows we always assume that the root node is $\wedge$ or equivalently that player 0 plays first. We want to know which player wins which corresponds to the value at the root of the tree. It is easy to evaluate the value at the root node with a recursive algorithm that visits **all n leaves of the tree** (similarly to the first Midterm problem), see Figure 1 for an example. By using randomness we can improve this runtime. Instead of trying both left and right child at every round we consider Algorithm 1 which picks a random branch at every round. On expectation Algorithm 1 will require to visit much less than n leaves. This is because for $\wedge$ nodes or nodes where Player 0 plays if we look at a random branch and see that its value is 0 we immediately know that the value of the current node is 0 (this is because $0 \wedge 1 = 0$, $1 \wedge 0 = 0$, $0 \wedge 0 = 0$). Similarly, for $\vee$ nodes if the random branch returns 1 we do not need to recurse to the other branch.

In Algorithm 1 the global variable N is initialized to 0 and counts the number of leaves that this randomized algorithm visits. **Observe that since at every step in Algorithm 1 we pick a branch at random, the value of N when all recursive calls return is a random variable.** Your task is to design an algorithm that takes as input the values of the leaves of the tree as a binary string and outputs the expected number of leaves visited by the given randomized algorithm, Algorithm 1, i.e., the expected value of N after the WhoWins$(v, 0)$ returns.

Input:

- Input should be read in from stdin.
- The first line will contain the height of the tree (n).
- One n-character line will follow, with each character indicating whether player 0 or 1 wins.
- 0s and 1s are listed in the order they appear in a printout of the tree — first the element found by only taking left branches, then the element found by taking $n-1$ lefts and a right, then the element found by taking $n-2$ lefts, one right, and one left, then $n-2$ lefts and two rights, then $n-3$ lefts, one right and two lefts, etc.

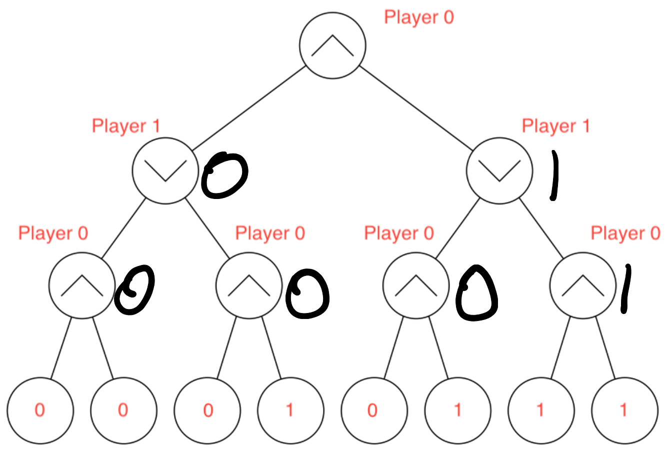


Figure 1: By $\vee$ we denote $\max$ or the "or" binary operator and by $\wedge$ we denote $\min$ or "and". In this case we have that the values of the bottom level $\wedge$ nodes are $0, 0, 0, 1$ from left to right. The values of the $\vee$ nodes are $0, 1$ from left to right. Therefore, the value at the root of the tree is 0 The expected value of `N` of Algorithm 1 in the above example is 3.875.

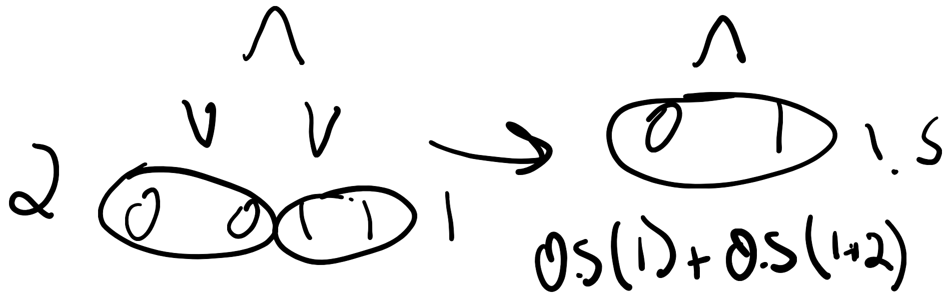
```
Algorithm 1: WhoWins($v$, player)
/* Initialize some global variable $N = 0$, i.e., $N$ is not re-initialized at every recursive call */
1  if $v$ is a leaf node then
2      $N = N + 1$
3      Return (value of $v$)
4  else
5      Let $s_1, s_2$ be the left and right subtrees of $v$ in a random order
6      next ← (player + 1) mod 2                    ▷ next contains the next player.
7      if WhoWins($s_1$, next) = player then
8          Return player                            ▷ player 0 is ∧, player 1 is ∨.
9      else
10         Return WhoWins($s_2$, next)
```

Output:

- The output should be written to stdout.
- The output should be the expected value of **total leaves**, `N`, at the end of execution (e.g. `2.5`). A small margin of error is allowed.

Constraints:

- $1 \leq n \leq 12$



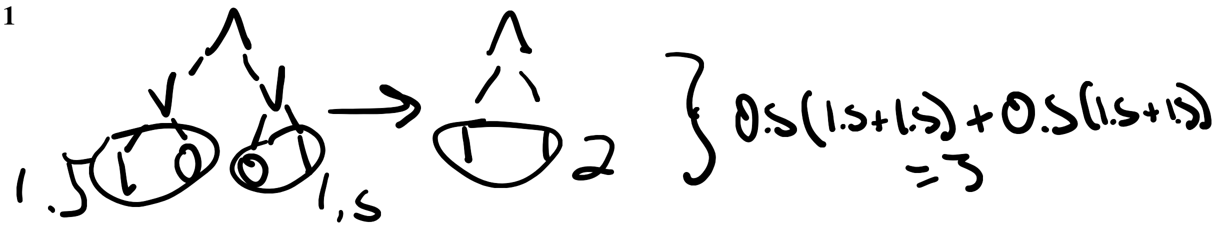
Examples

Example 1
input:

```
2
1001
```

output:

```
3.0
```

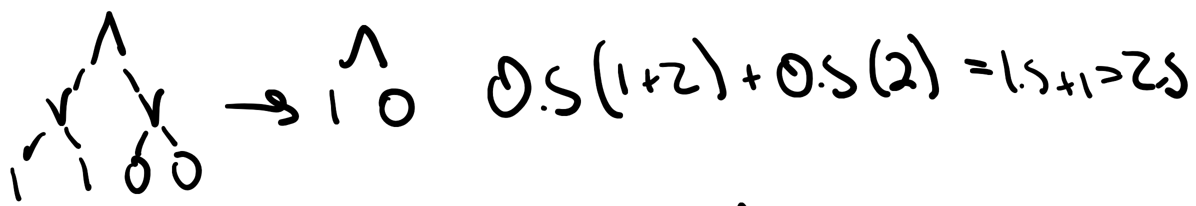


Example 2
input:

```
2
1100
```

output:

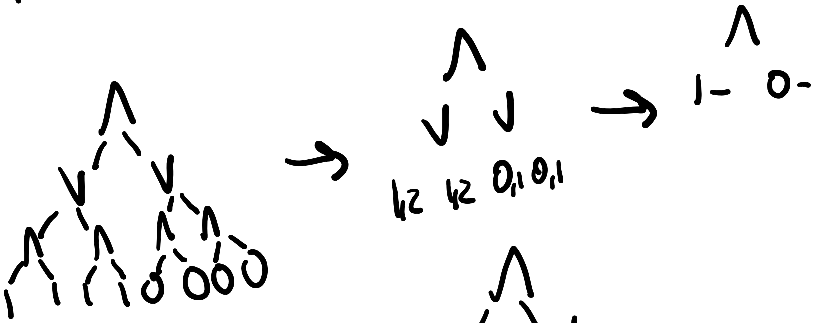
```
2.5
```



Example 3
input:

```
3
11110000
```

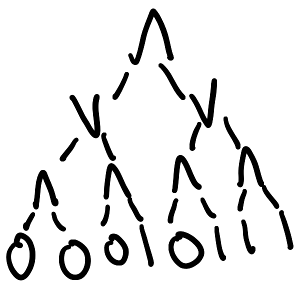
output: 3.0



Example 4
input:

```
3
00010111
```

output: 3.875



start at root

0.5(exp left + exp right if needed) + 0.5(exp right + exp left if needed)

how do we know if needed?

logic is there