Information

Animal separation

Consider a sequence `S` of characters describing an animal shelter. The characters are `|` (viewed as "wall"), `.` (viewed as "nothing"), and `*` (viewed as "animal"). The first and last characters of `S` are always a wall.

I want to check that the shelter is "safe", i.e., all animals are separated by walls. For instance, `|..|*|*|*|*|..|` is "safe", but `|**|` is not, and neither is `|..|*.......*|*|`.

Assume the sequence `S` is given as a linked list; each `Node` in the linked list has two instance variables: `c` is the character stored at this node (so `c` is one of `|.*`), and `next` is a reference to the following node.

To solve the safety problem, my program uses two nested `while`-loops; resetting an animal counter each time it "starts a new room". A pointer `current` to the current Node will advance to the right, a counter `animals_in_current_room` counts the animals encountered since the last `|`. In the middle of an execution, the situation could look like this:

```
|..|*.....*|*|
        ^
        |
    current

animals_in_current_room: 1
```

Here is a high-level sketch, it assumes that `current` begins by pointing to the first (leftmost) Node:

```
animals_in_current_room = 0
while True:
    while current.c != '|':
        if current.c == '*':
            animals_in_current_room += 1
        if animals_in_current_room == 2:
            exit("unsafe!")
        current = current.next
    animals_in_current_room = 0
    if current.next != null:
        current = current.next
    else
        break
print("safe!")
```

You are strongly encouraged to understand this on the level of pseudocode. As a service of *highly* questionable usefulness, here are also two minimal implementations in Java and Python that make the above idea concrete:

```
public class Animals
{
    static class Node {
    char c;
    Node next;
    }

    static Node build(String C) {
    Node prev = null;
    for (int i = C.length() - 1; i >= 0; --i) {
        Node fresh = new Node();
        fresh.next = prev;
        fresh.c = C.charAt(i);
        prev = fresh;
    }
    return prev;
    }

    public static void main(String[] args) {
    Node S = build("|**...|..*|*..|..|");

    Node current = S;
    int animals_in_current_room = 0;
    while (true) {
        while (current.c != '|') {
        if (current.c == '*')
            animals_in_current_room += 1;
        if (animals_in_current_room == 2) {
            System.out.println("unsafe!");
            return;
        }
        current = current.next;
        }
        animals_in_current_room = 0;
        if (current.next != null)
        current = current.next;
        else
        break;
    }
    System.out.println("safe!");
    }
}
```

```
class Node:
    slots = ['c', 'succ']
    def __init__(self, c, next):
        self.c = c
        self.next = next

def build(chars) -> Node:
    prev = None
    for c in reversed(chars):
        prev = Node(c, prev)
    return prev

S = build("|...|..*|*..|..|")

current = S
animals_in_current_room = 0
while True:
    while current.c != '|':
        if current.c == '*':
            animals_in_current_room += 1
        if animals_in_current_room == 2:
            print("unsafe!")
            exit(0)
        current = current.next
    animals_in_current_room = 0
    if current.next:
        current = current.next
    else:
        break
print("safe!")
```

Question **17**

Not yet answered

Marked out of 1.00

Professor Ynot thinks this is a silly implementation and suggests the following idea:

"First, transform `S` into a string or an array of characters. That takes linear time and allows constant-time access to the positions using `S[i]` or `S.charAt(i)`. And then I can just iterate over all pairs of `*` and check that there is a `|` between them."

Explain to the good professor the error of their ways. Mark all the correct responses.

Select one or more:

- [x] a. It sounds to me like your solution uses cubic time (in the length of `s`) in the worst case, unless you also use a symbol table or something. That's _really_ slow.
- [ ] b. Huh? That's basically the same solution; I'm just using lists instead of arrays. There is no difference in running time.
- [x] c. Wasteful cretin! That takes quadratic time *at least*! If there are `k` many `*`s in the input then you'd have to check $\binom{k}{2}$ many pairs of animals, and since `k` can be linear in `n`, the value of $\binom{k}{2} \sim \frac{1}{2}k^2$ is quadratic in `n`.
- [ ] d. My answer is faster because linked lists *also* have constant-time access and use memory much more efficiently.
- [ ] e. There's no way you can transform the given linked list into an array in linear time, silly man! You'd be wasting all your time in the very first step.

Question **18**

Not yet answered

Marked out of 1.00

What would be an appropriate *cost model* to analyse Ynot's algorithm?

Select one or more:

- [x] a. Number of assignments
- [ ] b. Number of function calls
- [ ] c. Number of times a pointer (reference) is followed.
- [ ] d. Number of `print` statements
- [x] e. Number of comparisons (including `!=` and `==`, on integers or characters)
- [x] f. Number of array accesses
- [ ] g. Number of multiplications

Question **19**

Not yet answered

Marked out of 1.00

What is the largest number of iterations of a single execution of the inner `while`-loop in the worst case, and how can such a worst case input look?

Select one or more:

- [ ] a. Constant
- [ ] b. Logarithmic
- [x] c. Linear
- [ ] d. Linearithmic
- [ ] e. Quadratic
- [x] f. `|.........….....|`
- [ ] g. `|*|*|*|*|...|*|`
- [ ] h. `|*******...**|`

Question **20**

Not yet answered

Marked out of 1.00

What is the worst-case running time for our algorithm, and how can such a worst case input look?

Select one or more:

- [ ] a. Constant
- [ ] b. Logarithmic
- [x] c. Linear
- [ ] d. Linearithmic
- [ ] e. Quadratic
- [x] f. |.....|
- [x] g. |*|*|*|*|...|*|
- [ ] h. |*******...**|

Question **21**

Not yet answered

Marked out of 1.00

What is the best-case running time for our algorithm, and how can such a best-case input look?

Select one or more:

- [x] a. Constant
- [ ] b. Logarithmic
- [ ] c. Linear
- [ ] d. Linearithmic
- [ ] e. Quadratic
- [ ] f. |.....|
- [ ] g. |*|*|*|*|...|*|
- [x] h. |*******...**|

Question **22**

Not yet answered

Marked out of 1.00

Which of the following arguments for the worst-case running time of our algorithm are both true and relevant?

Select one:

- a. The number of nodes is halved in every iteration. This can happen only a logarithmic number of times.
- b. In each iteration of the inner loop, `current` advances by one node, and `current` is never reset. Since there are exactly as many nodes as the input size, this can happen at most a linear number of times.
- c. The `break` statement must be executed sooner or later, otherwise the outer loop would run indefinitely.
- d. The number of pairs of animals in the input is at most quadratic.
- e. We can speed up the running time by sorting the input.
- f. The inner loop takes at most linear time each time it is executed. Two nested loops with t_1 and t_2 may iterations, respectively, take at most $\min(t_1, t_2)$ many iterations in total. Therefore, the whole programme takes linear time.
- g. There can be at most a linear number of animals in the input.

Clear my choice