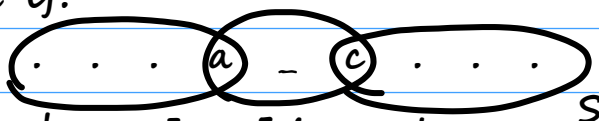
Temporal Pincer variant

Let's say we want to delete 'b' from the list and tell if it is 'good'

. . . a b c . . .

list is 'good' if:



Prefix is good [a, c] is good Suffix is good

What are the prefixes/suffixes? Let's see an example:

1 2 3 3 4 5

Prefixes

T [ ]
T [1]
T [1 2]
T [1 2 3]
F [1 2 3 3]
F [1 2 3 3 4]
F [1 2 3 3 4 5]

Suffixes

[1 2 3 3 4 5] F
[2 3 3 4 5] F
[3 3 4 5] F
[3 4 5] T
[4 5] T
[5] T
[ ] T

Let's write it in tabular form:

	1	2	3	3	4	5
prefix	T	T	T	T	F	F	F
suffix	F	F	F	T	T	T	T

Aka prefix is good: [1 2]

Suffix is good: [3 4 5]

And we must check whether we can tie up their ends:

	1	2	3	3	4	5
prefix	T	T	T	T	F	F	F
suffix	F	F	F	T	T	T	T

[prefix is safe.... x)

(x ... suffix is safe]

Temporal Pincer variant continued...

But do we need lists to store the prefix/suffix 'is good' values?

NO. The 'good'-nesses of prefixes and suffixes are monotonic!

Prefix is always Trues then Falses

Suffix is always Falses then Trues

Aka it is enough to store just an index where prefix/suffix turns T -> F

	1	2	3	3	4	5
prefix	T	T	T	T	F	F	F
suffix	F	F	F	T	T	T	T

So we need to only 'store' two integers (not lists)

Ok. We are going to form 2 teams and search for these indices.

Red Team will go forwards until they flip (T -> F)

	1	2	3	3	4
prefix	T	T	T	T	F	F	F
suffix	F	F	F	T	T	T	T

Blue Team will go backwards until they 'flip' (T -> F)

Then we'll check the possible choices where both are True

	1	2	3	3	4	5
prefix	T	T	T	T	F	F	F
suffix	F	F	F	T	T	T	T

Choice A

1	2	3	3	4	5
T	T	T	T	F	F	F
F	F	F	T	T	T	T

In this case (2, 3) is good so this is a valid choice

[1 2 _ 3 4 5]

Choice B

1	2	3	3	4	5
T	T	T	T	F	F	F
F	F	F	T	T	T	T

In this case (3, 4) is good so this is a valid choice

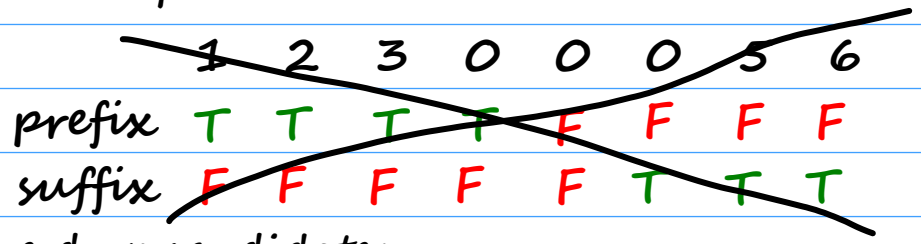
[1 2 3 _ 4 5]

So in this case, they are both good choices

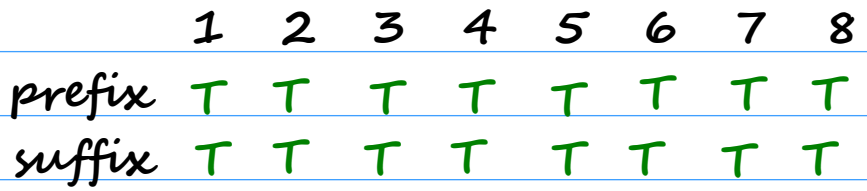
Temporal Pincer variant continued...

Of course, besides the double hit case, we have others like

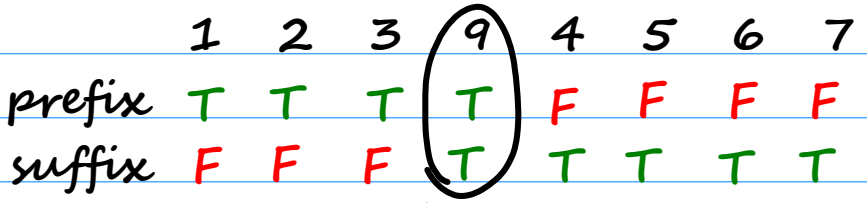
None are drop candidates...



All are drop candidates...

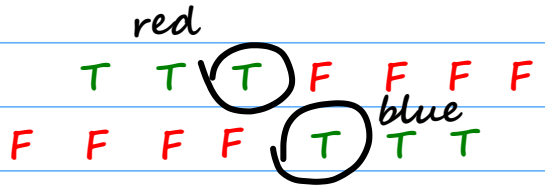


One drop candidate...



But these can be covered by simply looping from blue to red:

In the no drop candidate case we wont even enter the loop because blue > red



Summary

1) We are going to form two teams and find the two indices:

Red Team knows that before r everything safe

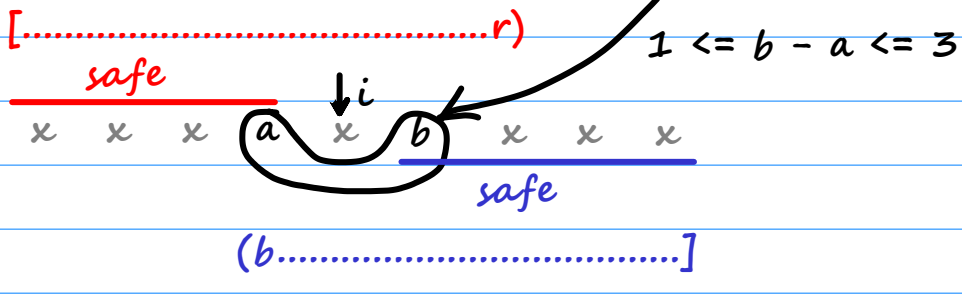
[.....r)

1 2 3 3 4

(b.....]

Blue Team knows that after b everything is safe

2) We'll loop over the possible choices to drop (i <- [blue..red]) and for each i we'll do a constant time check:



See below link for runtime behaviour:

https://youtu.be/gb06X_aCxDE?t=200