9) Check if expressions same or not?

a) $-(a+b+c)$ } Yes
b) $-a-b-c$

a) $a-b-(c-d)$ } NO
b) $a-b-c-d$

Condition are:
- operator = (+ / -)
- operands = (only small alphabets) No repetition of term
- Both will have same operands

⇒ Solutions

1) Converting both in postfix & comparing them. [WO]

⇒ Given an expression how to get its bool array?

$-(a+b+c) = -a-b-c$

- will be applied to all characters with in brace

$-(a-b-c) = -a+b+c$

global sign — local sign

[Sign of a character is calculated] = (Global sign × Local sign)

+ × +	= +
+ × −	= −
− × +	= −
− × −	= +

[Sign before operand] — If before a char we have (b — local sign of b is +)

Step: Iterate on expression & fill bool exp[] value with local signs

⇒ Distinct char : 26

[# of signs of all char of exp1 = signs of all char of exp2]

We can use bool exp1[26], exp2[26] to store signs

exp1[0] — This contains sign of a of exp1

exp1[1] — This contains sign of a in exp1

0 – 'a'
1 – 'b'
⋮
25 – 'z'

// How to get global signs

$a+b+(d-e-(g-f+h)+l)$

$l = -$ $l+1$
$g = -$ $g = -$
$a+1$ $a = -$ $a = +$

$[a-b-(c-d-(f+h)+k-(l+m-o))] = (p+q)) + n$

F + − − + + − + + −

Algo:

1) If str[i] = '('
   - → Get str[i-1] : l
   - → Get top of stack : g
   - → If (l == g) { push + in stack }
   - else { push − in stack }

To avoid a lot of edge cases Assume entire expression is in (...) So start we can always initialize with +

else if str[i] = ')'
   - → pop top element from stack

else if str[i] = '+' or '−'
   - → Continue

else
   - → we are on an operand
   - → ch = str[i]
   - → Get top of stack : g
   - → Get local of str[i] = l
   - If (g == l)
     - sign of str[i] = '+'
   - else
     - sign of str[i] = '−'

Manacher's: We can get length of longest palindromic substring.

TC: $O(N^2)$

⇒ { two pointer approach for all odd / even centers }

Str : f a b b a f

Str : @ # f # a # b # b # a # f # $ (p1, p2, 6)

P[i] = [Number of times we can extend P1 & P2 with i as center]

Str = N

MStr = 2N+3

P[2N+3] = 0

for (int i = 1; i < 2N+2; i++)
{
  P1 = i−1; P2 = i+1;
  while (str[P1] == str[P2])
  {
    P1−−; P2++; P[i]++;
  }
}

TC: $O(N^2)$, SC: $O(N)$

Final ans: max of P[]

Str = b c b a b c b

MStr = @ # b # c # b # a # b # c # b # $

P[] = 0 1 0 3 0 1 0 7 0 1 0 3

id, i, x = i − id, n = i − id

id − i' = i − id

id − i + id = i'

i' = id − (i − id)
= id − i + id
= 2id − i

i' = 2id − 1

{ P[i] = P[2id − i] } Intuition 1

What we can copy

diff = P[id] + id − i

P[id] + id

collapsed Str

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
@ # e # f # k # f # e # a # e # f # k # f # d # $
P[] 0 0 1 0 1 0 5 0 1 0 1 0 9 0 1 0 1 0 5 3

For a given i at max how much can we extend on right side

P[i] = min(P[2id − i], P[id] + id − i) → Intuition 2

How much to extend on right

We will use this equation to fill values

Cases:

18 − 3 − 1 P1 P2 = 18 + 3 + 1

id + P[id]

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
@ # d # f # k # f # e # a # e # f # k # f # d # a # $
P[] 0 0 1 0 1 0 3 0 1 0 1 0 9 0 1 0 1 0 3

id, i

i = 22 − P[i] = 1

i + P[i] > 21

10^6

P[]

id = 1

for (int i = 1; i < 2N+1; i++)
{
  if (id + P[id] >= i)
  {
    P[i] = min(P[2id − i], P[id] + id − i)
  }
  else
  {
    P1 = i − P[i] − 1, P2 = i + P[i] + 1;
    while (str[P1] == str[P2])
    {
      P1−−; P2++; P[i]++;
    }
  }
  if (id + P[id] < i) { i is outside segment, default P[i] = 0 }
  if (id + P[id] > i) { if we are exceeding current segment, then update id = center }
}

i in segment

→ But we don't have to work above conditions

Initialize

Extend

Iterate on center P[] & return max value