Splitting Algorithm:-

1. splited Reg = split_Regex ()
2. NFA = creat_NFA_from_splits ()

---

split_Regex (Regex) → ~~array of strings~~ (New Reg)

1. iterate over each character in Regex
   1. if c == '(' ✓
      1. bracket is open = True // we should handle // the nested brackets
   2. if c == ')' ✓
      1. List.append (created_str)
         bracket is open = false
   3. else
      created_str += c
      → take care of symbols (→ * → + ?) ! should be treated

و هعمل map جواها كل حرف الرموز الجديدة
بتشاور على الحروف بتاعتها

f1 → (abc)
f2 → (f1 | f2)

---

create_NFA_from_splits ()

1. Zu Z Algo for each symbol
2. Merge ✓✓

Scanned with CamScanner

) NFA Algorithm:

1. Assuming No symbols or brackets for now.

1. Define a bool indicates whether we should insert empty state → is Epsilon
2. Define a list which contains our states → [ ] empty.
3. initialize characters idx which points to the current character of the given regular expression with -1
   عشان اول step بيبص على اللي بعده.
4. create initial state S1 (init=T, Term= F, trans={ })
5. Append it to the states list
6. iterate for each state in the states list
   - 6.1. apply logic (state, charIdx, isEpsilon, regex, states
   - 6.2. is Epsilon = ~ is Epsilon → toggle it
   - 6.3. charIdx += ! isEpsilon we move the pointer in case that we are not @ Epsilon state

//we should continue the NFA algo, but lets stop here now & implement apply logic
thompson

---

apply logic (state, charIdx, is Epsilon, regex, states

1. create new empty state → Si
2. Append it to the states list → states. append(Si)
3. if (is Epsilon)
   - 3.1. state.Trans{'ε'}. append[Si]
4. else
   - 4.1. if (regex(charIdx + 1) is alpha numeric)
     - 4.1.1. state. Trans{*}. append [Si]
   - 4.2. else
     //we should think how to handle these cases.

→ expand_square_brackets (str → [data])

1. Two pointers s & f = 0, 1

a-zA-Z0-9

2. while (s < str.len)

ret str

1. if (f == '-')

f++

for (c = s, c < f, c++)

retStr += 'c'

2. elif (f == str.len)

retstr += str(s)

3. elif (s != f)

s++

f++