**Objašnjenje pristupa**

1. **Provera** **osnovnih** **uslova**

* Ako je string null ili prazan, metoda vraća dužinu 0.
* Ako je string dužine jednog karaktera, metoda vraća dužinu 1 jer je to validan palindrom.

1. **Inicijalizacija**

* Inicijalizuje se promenljiva ‘maxPalindrome’ koja će čuvati najduži palindrom u stringu.

1. **Iteracija** **kroz** **string**

* For petlja se koristi za iteraciju kroz string, počevši od drugog elementa (indeks 1). Ideja je da se za svaki karakter traži centar potencijalnog palindroma.
* Ukoliko bi se islo logikom da se samo proveravaju susedi i ako su isti, dalje proširivati, onda se ne bi pokrili slučajevi kada su isti karakteri susedi odnosno kada je paran polindrom. Iz tog razloga se na početku for petlje određuju najduži neparni kao i najduži parni polindrom za datu iteraciju tj. za trenutni centar samog palindroma.

1. **Proširenje palindroma**

* Metoda ’expandPalindrome’ širi palindrom oko centra sve dok su karakteri sa leve i desne strane jednaki.
* Ako su karakteri jednaki, pokazivači ( ‘left’ , ‘right’) se pomeraju u levo i u desno.

1. **Ažuriranje najdužeg** **palindroma**

* Nakon što se pronađe potencijalni palindrom (neparni ili parni) proverava se da li je duži od trenutno sačuvanog najdužeg palindroma( ‘maxPalindrom’).
* Ako jeste, ažurira se ‘maxPalindrome’.

1. **Vraćanje** **rezultata**

* Nakon završetka iteracije, metoda vraća dužinu najdužeg pronađenog palindromskog podstringa, koja se dobija preko ‘maxPalindrome.length()’.

**Objašnjenje vremenske složenosti rešenja**

* Vremenska složenost ovog algoritma je: *O(n2)* , gde je *n* dužina ulaznog stringa.

**Računanje vremenske kompleksnosti**

1. **For** **petlja**

* For petlja prolazi kroz svaki karakter u stringu, što daje složenost *O(n)*.

1. **Proširenje palindroma**

* Unutar svake iteracije for petlje, metoda ‘expandPalindrome’ se poziva dva puta (za neparni i parni palindrom). U najgorem slučaju, metoda može proširiti palindrom do celog stringa, što znači da ima složenost *O(n).*

1. **Ukupna složenost**

* Kombinovanjem ovih složenosti, dobijamo ukupnu vremensku složenost algoritma *O(n)∗O(n)=O(n2)*.