

Manacher's Algorithm

2020.01.03

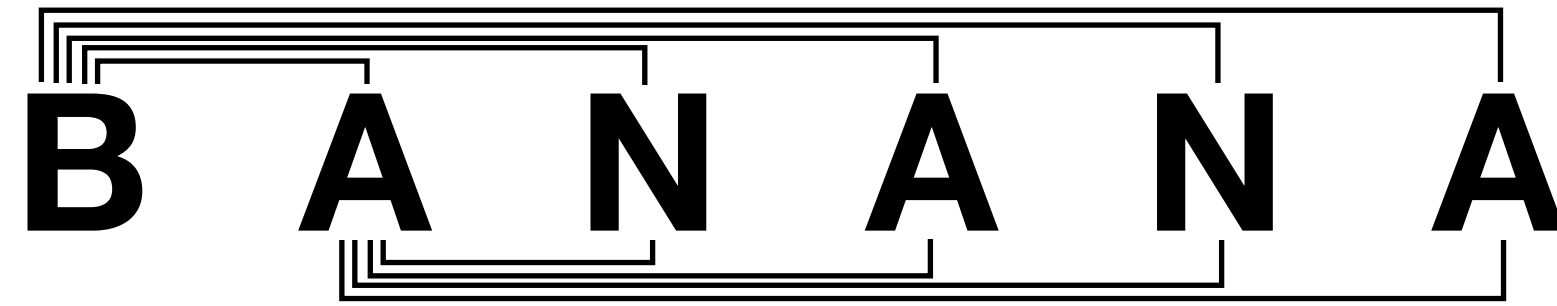
Manacher's Algorithm

| | | |
|-------|------------|----|
| mana하 | Manacher's | 10 |
|-------|------------|----|

- 가장 긴 팰린드롬 부분 문자열 찾기
 - $[p, q]$ 범위의 부분 문자열이 팰린드롬인지 묻는 복수의 쿼리 응답
- ➔ In linear time!

Naive Approach

B A N A N A



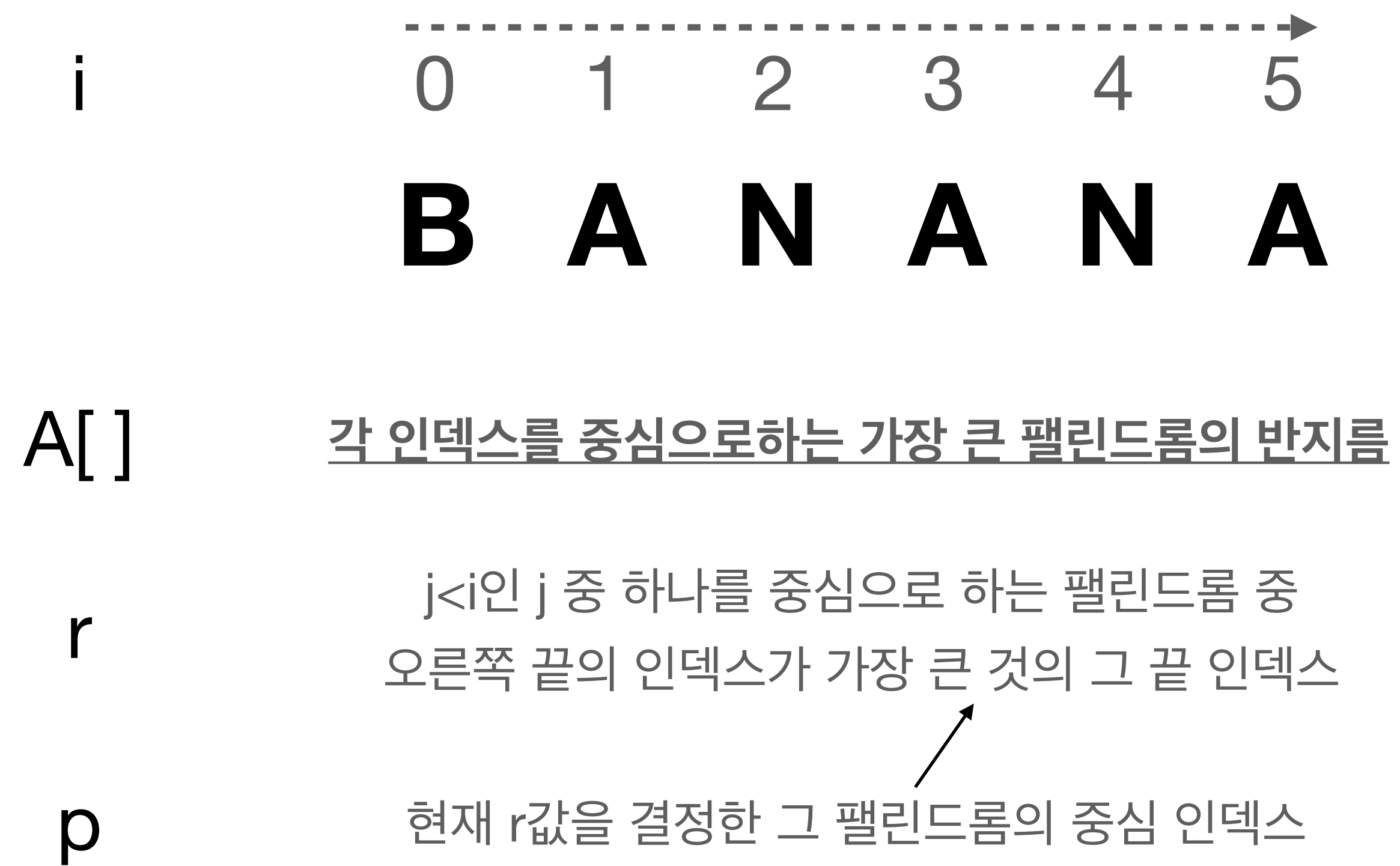
┆ 가장 큰 팰린드롬 부분 문자열 찾기

➔ 모든 부분 문자열에 대해서 체크: $O(n^2)$

┆ $[p, q]$ 범위의 부분 문자열이 팰린드롬인지 묻는 복수의 쿼리 응답

➔ 쿼리의 수 x 부분 문자열 체크: $O(mn)$

How it works



How it works

i

012345

BANANA

A[]

r

p

각 인덱스를 중심으로하는 가장 큰 팰린드롬의 반지름

j<i인 j 중 하나를 중심으로 하는 팰린드롬 중 오른쪽 끝의 인덱스가 가장 큰 것의 그 끝 인덱스

현재 r값을 결정한 그 팰린드롬의 중심 인덱스

- |

if (i > r)

➡ 아직 팰린드롬에 포함되지 않음
- |

if (i <= r)

➡ 이미 어떤 팰린드롬에 포함되어 있음

각 경우에 따라 A[i]의 초기값 설정 후
팰린드롬이 형성되지 않을 때까지 A[i]를 1씩 증가하여 A[i]가 결정됨
이 때 A[i]의 초기값은 i를 중심으로 하는 팰린드롬의 반지름이 적어도 A[i]임을 의미

| | | | | | | |
|------|---|---|----------|---|---|---|
| i | 0 | 1 | 2 | 3 | 4 | 5 |
| | B | A | <u>N</u> | A | N | A |
| A[] | 0 | 0 | | | | |
| r | 0 | 1 | | | | |
| p | 0 | 1 | | | | |



How it works

i

012345

BANANA

A[]

r

p

----->

각 인덱스를 중심으로하는 가장 큰 팰린드롬의 반지름

j<i인 j 중 하나를 중심으로 하는 팰린드롬 중
오른쪽 끝의 인덱스가 가장 큰 것의 그 끝 인덱스

현재 r값을 결정한 그 팰린드롬의 중심 인덱스

- |

if (i > r)

➡ 아직 팰린드롬에 포함되지 않음
- |

if (i <= r)

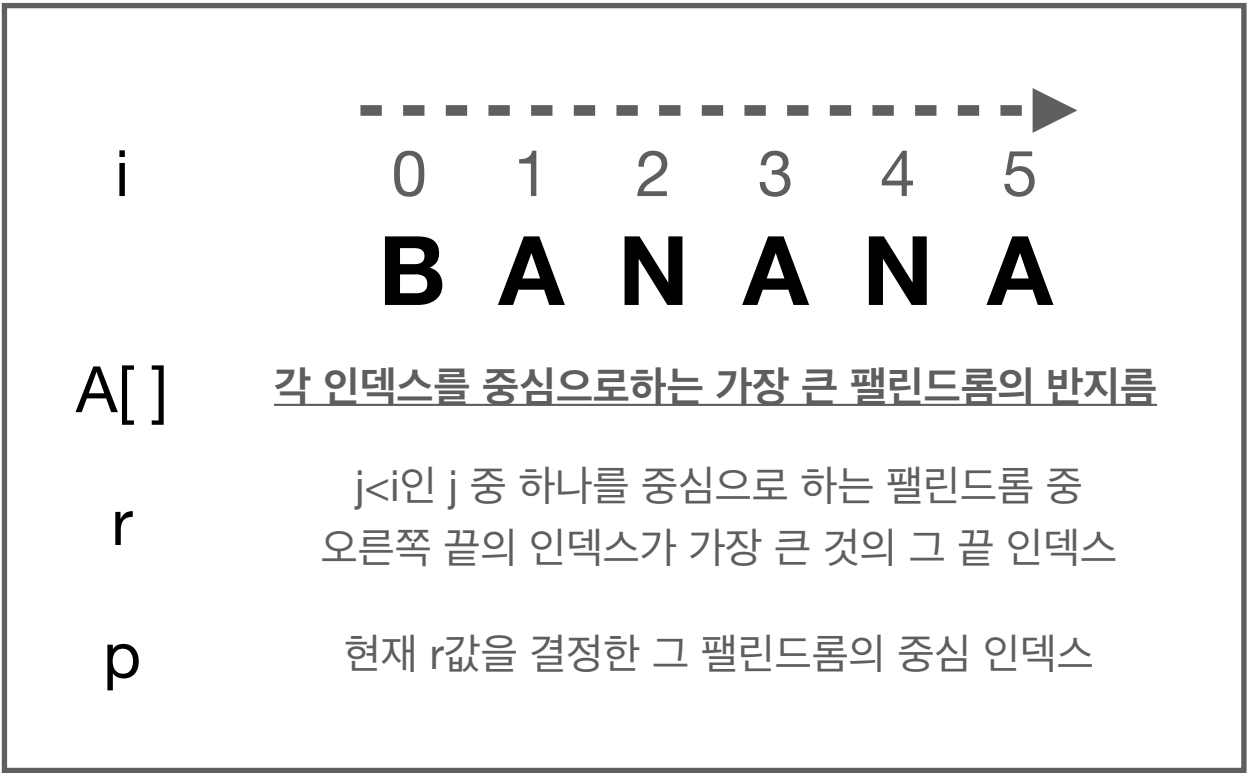
➡ 이미 어떤 팰린드롬에 포함되어 있음

각 경우에 따라 A[i]의 초기값 설정 후
팰린드롬이 형성되지 않을 때까지 A[i]를 1씩 증가하여 A[i]가 결정됨
이 때 A[i]의 초기값은 i를 중심으로 하는 팰린드롬의 반지름이 적어도 A[i]임을 의미

| | | | | | | |
|------|---|---|---|---|---|---|
| i | 0 | 1 | 2 | 3 | 4 | 5 |
| | | | | | | |
| | B | A | N | A | N | A |
| | | | | | | |
| A[] | 0 | 0 | 1 | 2 | | |
| | | | | | | |
| r | 0 | 1 | 3 | 5 | | |
| | | | | | | |
| p | 0 | 1 | 2 | 3 | | |



How it works



| if (i > r)

➔ A[i] = 0

| if (i <= r)

➔ A[i] = min(A[2p - i], r - i)

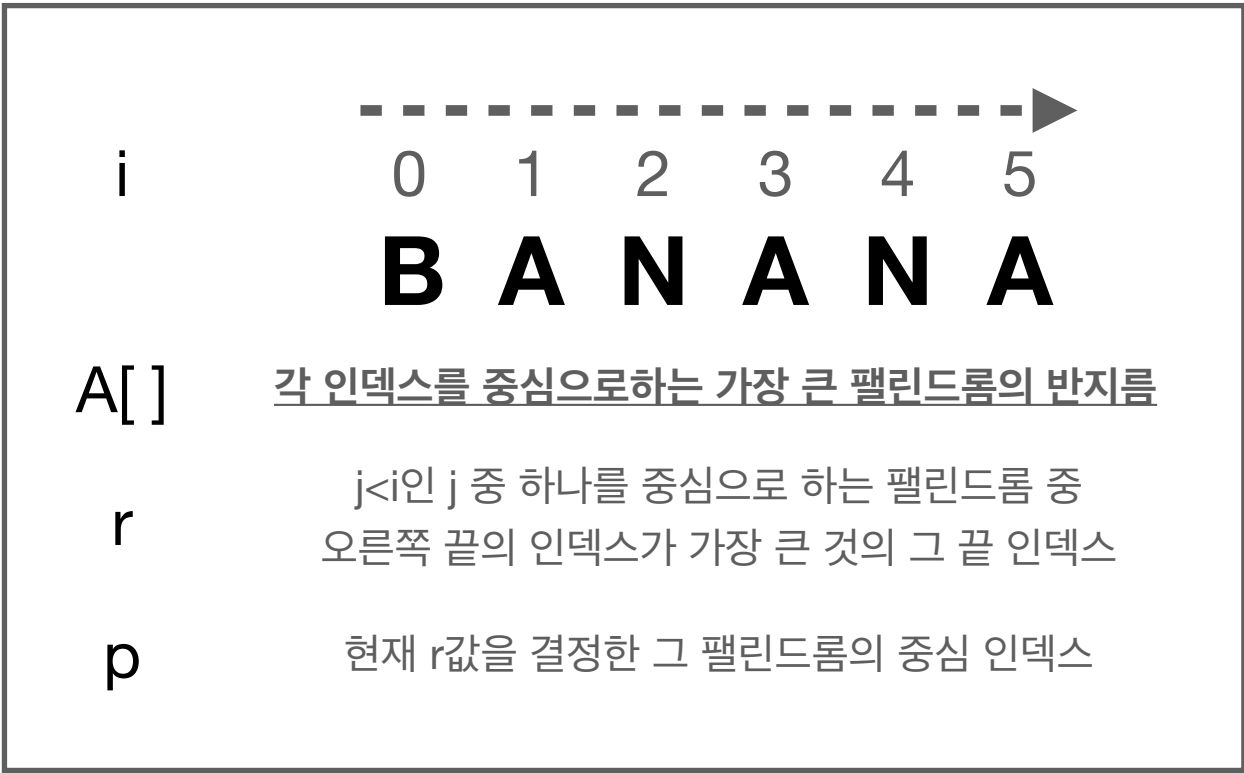
2p-i: 팰린드롬의 중심을 기준으로 대칭 위치에 있는 지점
단, i + A[2p-i] 이 r을 벗어나면 s[i+A[2p-i]] ~ s[i+A[2p-i]]이
팰린드롬임을 보장할 수 없으므로 min으로 제한

➔ 유효한 범위 내에서 (s[i - A[i] - 1] != s[i + A[i] + 1]) 일 때까지 A[i]를 1씩 증가

➔ r < i + A[i] 일 경우 r과 p를 업데이트

| | | | | | | |
|------|----|---|---|---|---|---|
| i | 0 | 1 | 2 | 3 | 4 | 5 |
| | B | A | N | A | N | A |
| A[] | 0 | 0 | 1 | 2 | 1 | 0 |
| r | -1 | 0 | 1 | 3 | 5 | 5 |
| p | -1 | 0 | 1 | 2 | 3 | 3 |

How it works



| if ($i > r$)

→ $A[i] = 0$

| if ($i \leq r$)

→ $A[i] = \min(A[2p - i], r - i)$

→ 유효한 범위 내에서 ($s[i - A[i] - 1] \neq s[i + A[i] + 1]$) 일 때까지 $A[i]$ 를 1씩 증가

→ $r < i + A[i]$ 일 경우 r 과 p 를 업데이트



가장 긴 팰린드롬 부분 문자열이 뭔가요?

→ 배열 A 순회

부분문자열 $[p, q]$ 가 팰린드롬인가요?

→ p, q 의 중심점을 mid 라고 할 때,
 $q \leq mid + A[mid]$ 이면 true, 아니면 false

Does it work..?

- 홀수 길이 팰린드롬만 찾을 수 있는거 아니예요?
- 이거 $O(n)$ 으로 구하는 거 맞아요?

Does it work..?

| 홀수 길이 팰린드롬만 찾을 수 있는거 아니예요?

➔ 항상 홀수 길이가 되도록 더미 문자를 넣어주기

E # E # V # E # E

Does it work..?

| 이거 $O(n)$ 으로 구하는 거 맞아요?

➔ amortized $O(n)$

| if ($i > r$)

➔ 1) $A[i] = 0$

| if ($i \leq r$)

➔ 2) $A[i] = A[2p - i]$

➔ 3) $A[i] = r - i$

1) $A[i]$ 가 증가할 수도 안 할 수도 있으며 항상 $r = i + A[i]$ 를 수행한다.

2) i 의 대칭점을 중심으로 하는 팰린드롬이 p 를 중심으로 하는 팰린드롬에 포함되므로 i 를 중심으로 하는 팰린드롬도 i 의 대칭점의 것과 완전 동일하다.

따라서 $A[i]$ 는 더이상 증가되지 않으며 r 도 갱신되지 않는다.

3) $A[i]$ 가 증가할 수도 안 할 수도 있으며 $A[i]$ 가 증가하면 $r = i + A[i]$ 를 수행한다.

➔ $A[i]$ 증가 연산은 $i + A[i]$ 가 r 보다 커질 때 최대 그 차이만큼 수행된다.

그런데 r 은 항상 원래 문자열의 길이보다 작으므로
증가 연산의 총 수행 횟수는 amortized $O(n)$

Implementation

```
// 더미 문자가 추가된 문자열 만들기
cin >> raw;
s = "";
for (auto c: raw) {
    s += '#';
    s += c;
}
s += '#';

r = p = -1;
for (int i=0; i<s.size(); i++) {
    // A[i] 초기화
    if (i <= r) {
        A[i] = min(r-i, A[2*p-i]);
    }
    else {
        A[i] = 0;
    }

    // A[i] 증가
    while (i-A[i]-1 >= 0 && i+A[i]+1 < s.size() && s[i-A[i]-1] == s[i+A[i]+1]) {
        A[i]++;
    }

    // r, p 갱신
    if (r < i+A[i]) {
        r = i+A[i];
        p = i;
    }
}
```

You, 2 days ago • Good Bye 2020!

Related Problems

- [BOJ 14444] 가장 긴 팰린드롬 부분 문자열
- [BOJ 11046] 팰린드롬??
- [BOJ 14417] 팰린드롬과 쿼리2 (Manacher + Segment Tree)