

# 文字列アルゴリズム

北海道大学 理学部 化学科 B3 TAB

# 参考資料

- snuke さんのブログ  
文字列の頭いい感じの線型アルゴリズムたち  
(MP, KMP, Manacher, Z-algorithm)
- Competitive Programmer's Handbook  
海外版の蟻本的なもの (Z-algorithm)
- Codeforces の記事 (Z-algorithm)
- @kyuridenamida さんのスライド (KMP, BM, など)
- potatosensei さんのブログ (KMP)

# 今回の発表内容

- Manacher's algorithm
- Z-algorithm
- MP 法
- KMP 法

# 今回使う記号

- $|S|$  : 文字列  $S$  の長さ
- $S[i]$  : 文字列  $S$  の  $i$  文字目 (0-indexed とする)
- $S[i, j]$  : 文字列  $S$  の  $i$  文字目から  $j$  文字目までを取り出した文字列

# Manacher's algorithm

文字列  $S$  に対して,  $S[i]$  を中心とした回文 (奇数長) の半径の最大値を各  $i$  に対して求めるアルゴリズム。

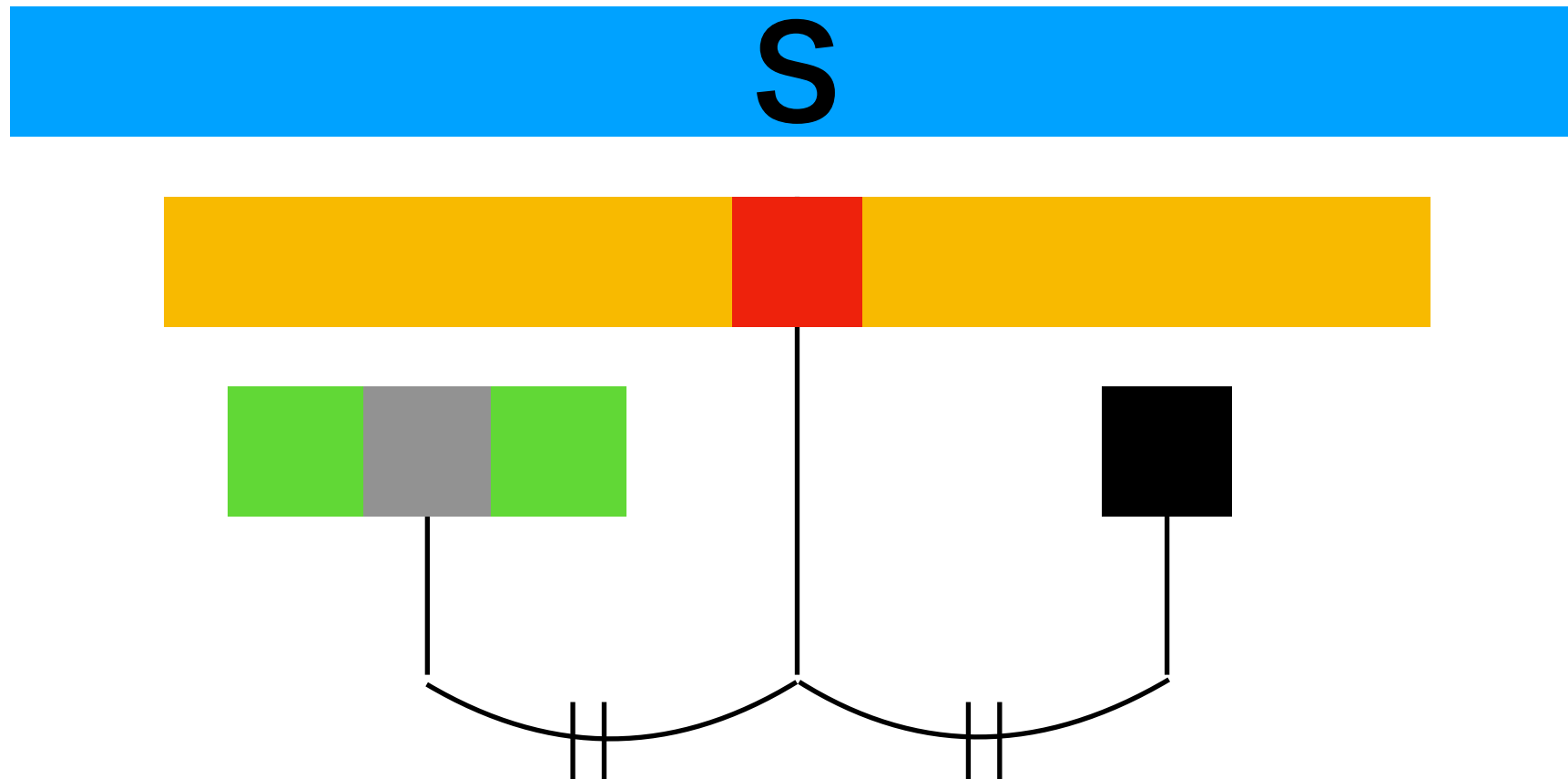
全体で  $O(|S|)$

回文の半径 :  $(\text{回文の長さ} + 1) / 2$

例えば, “TABAT” の半径は 3

S	a	b	a	a	a	b	a	b	a
R	1	2	1	4	1	2	3	2	1

# ベースとなるアイデア



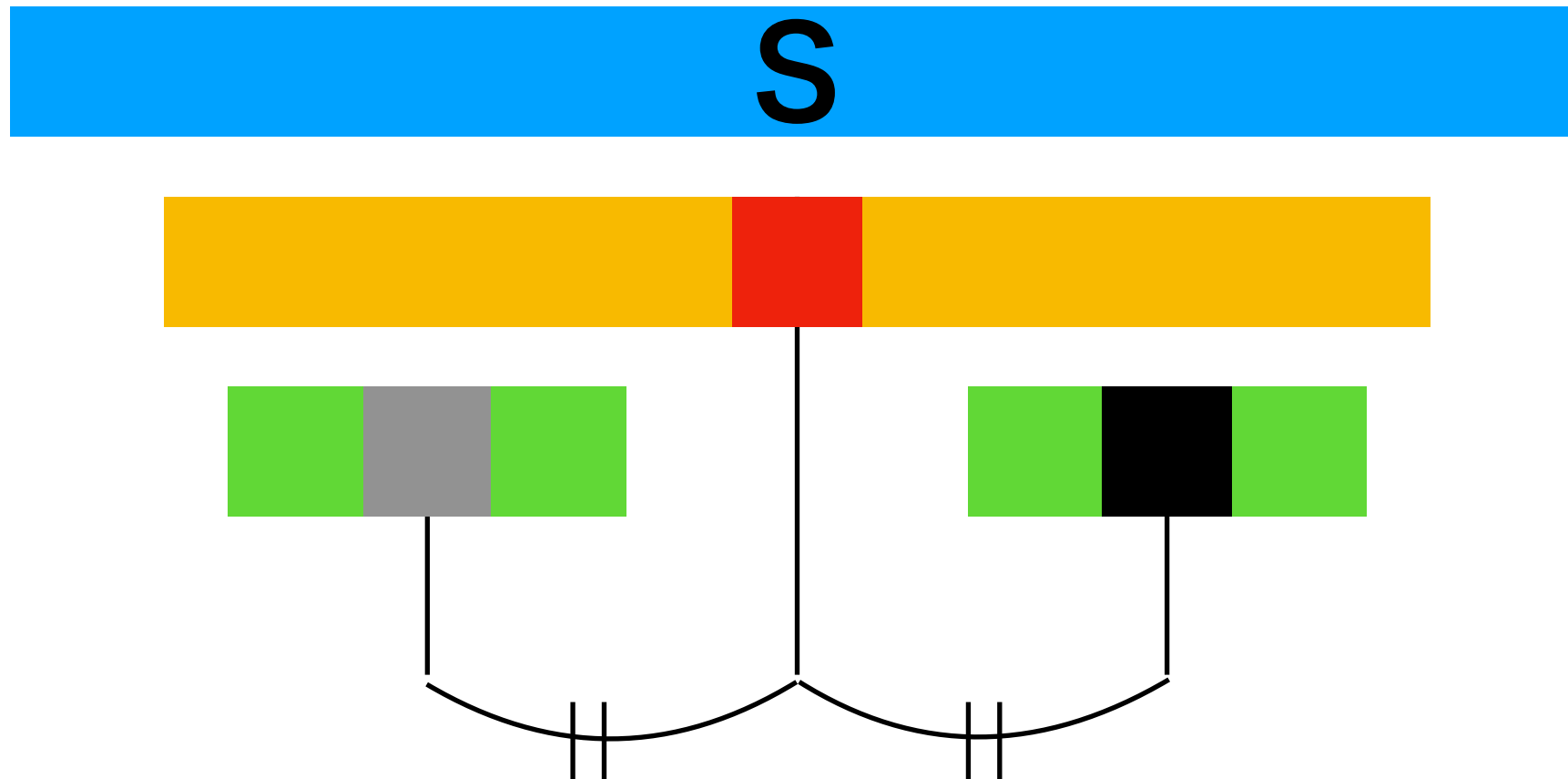
赤、グレーを中心とした回文の半径はすでにわかっている

黒を中心とした回文の半径が知りたい

→グレーを中心とした時の値を見れば良い

黒を中心とした時と、グレーを中心とした時は同じ値

# ベースとなるアイデア



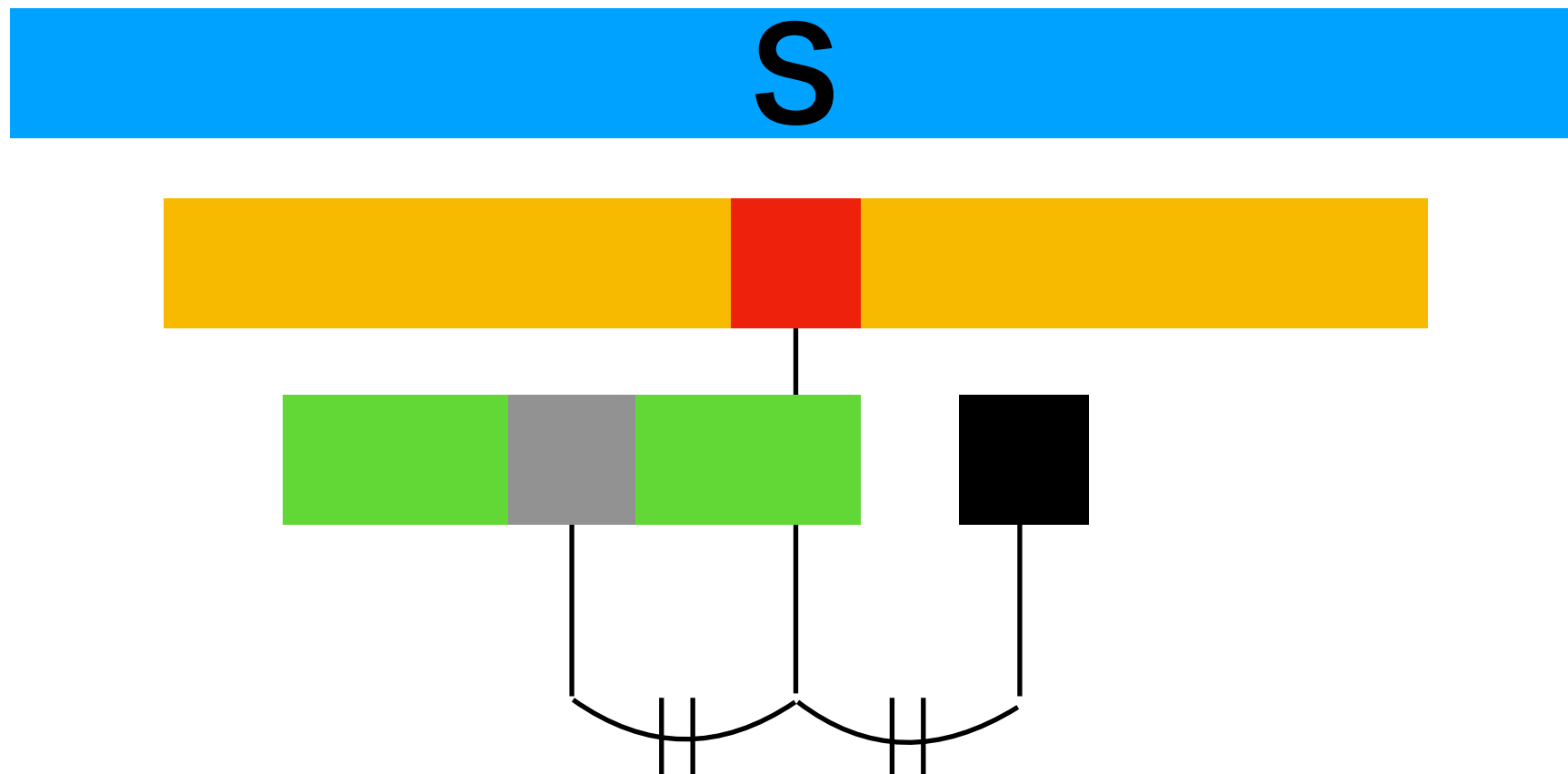
赤、グレーを中心とした回文の半径はすでにわかっている

黒を中心とした回文の半径が知りたい

→グレーを中心とした時の値を見れば良い

黒を中心とした時と、グレーを中心とした時は同じ値

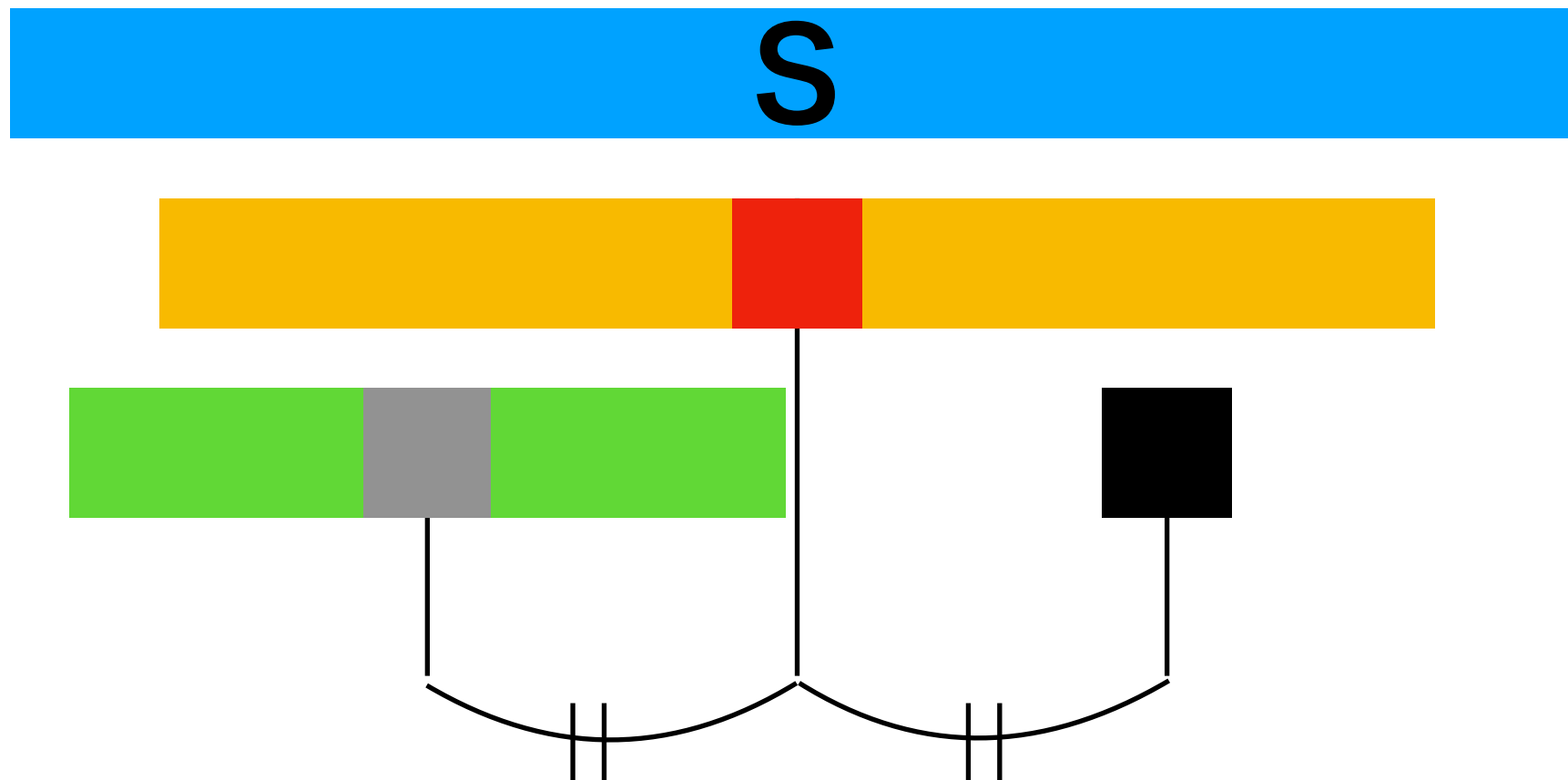
# ベースとなるアイデア



回文が中心をまたいでいても同じことが言える。



# ベースとなるアイデア

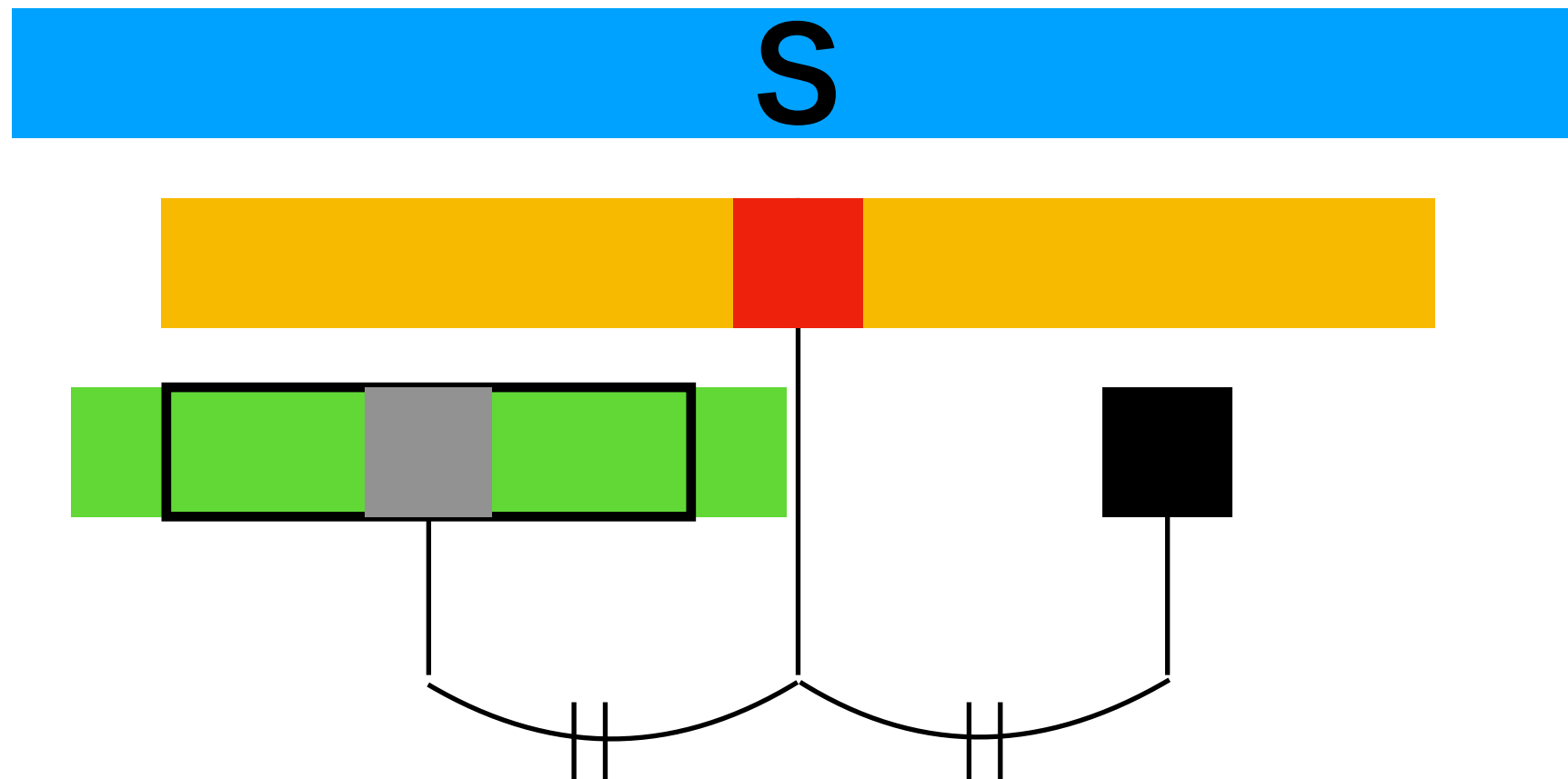


緑の回文が黄色の回文の外に出ている場合

黒を中心とした回文の半径の最大値と、緑の回文の半径が同じとは限らない

→今わかっている範囲で伸ばせるところまでは伸ばして、残りは愚直に求める

# ベースとなるアイデア

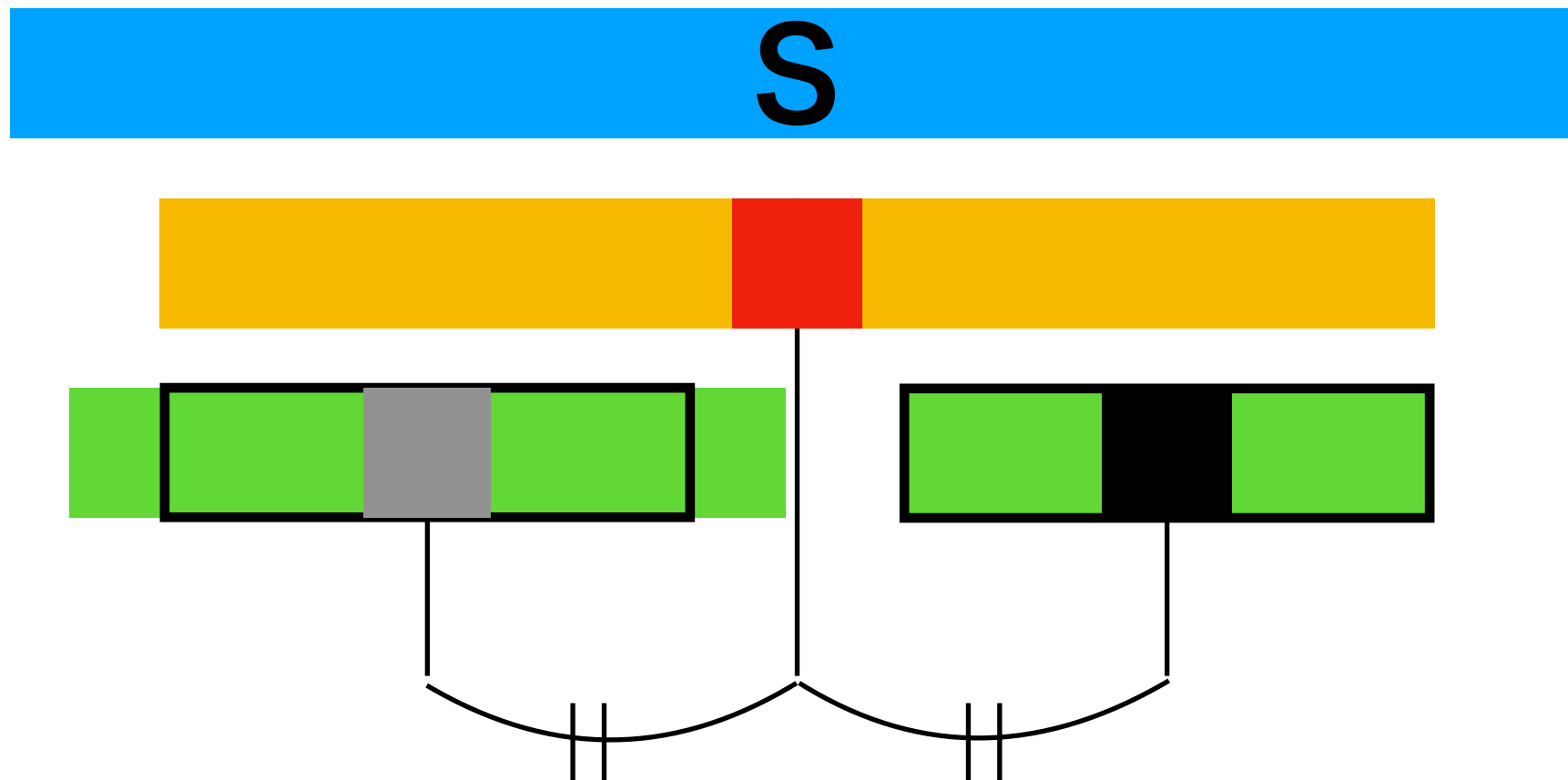


緑の回文が黄色の回文の外に出ている場合

黒を中心とした回文の半径の最大値と、緑の回文の半径が同じとは限らない

→今わかっている範囲で伸ばせるところまでは伸ばして、残りは愚直に求める

# ベースとなるアイデア



緑の回文が黄色の回文の外に出ている場合

黒を中心とした回文の半径の最大値と、緑の回文の半径が同じとは限らない

→今わかっている範囲で伸ばせるところまでは伸ばして、残りは愚直に求める

# ベースとなるアイデア

左側から順番に半径の最大値を求めていくことを考える。

赤を中心とした回文は何なのか？

→すでに見つけた回文の中で、右側が最も右にあるものを覚えておけば良い。



上のようになったら黒を中心とした回文をベースとする。

# ソースコード



```
vector<int> Manacher(string S){  
    int c = 0, n = S.length();  
    vector<int> R(n,1);  
    for(int i = 0; i < n; ++i){  
        int l = c - (i - c);  
        if(i + R[l] < c + R[c]){  
            R[i] = R[l];  
        }else{  
            int j = c + R[c] - i;  
            while(i - j >= 0 and i + j < n and S[i-j] == S[i+j]) ++j;  
            R[i] = j;  
            c = i;  
        }  
    }  
    return R;  
}
```

# 計算量の解析

while 文中の  $++j$  が何回実行されるかを考える

$j = R[c] + c - i$  より

$i + j = R[c] + c$

while 文を抜けた後,

$R[i] = j, c = i$  としているので,  $i + j = R[c] + c$  が成り立つ

$i + j < |S|$  なので,  $++j$  された回数は全体で  $O(|S|)$  である

従って for 文と合わせて全体でも  $O(|S|)$  である

# Manacher's algorithm

偶数長の回文は検出できないの？

# Manacher's algorithm

偶数長の回文は検出できないの？

→できます！



# Manacher's algorithm

偶数長の回文は検出できないの？

→できます！

$S = \text{"aabbaa"}$  の時、  $S' = \text{"$a$a$b$b$a$a$"}$  として  
 $S'$  に対して Manacher's algorithm を適用すればよい

# Z-algorithm

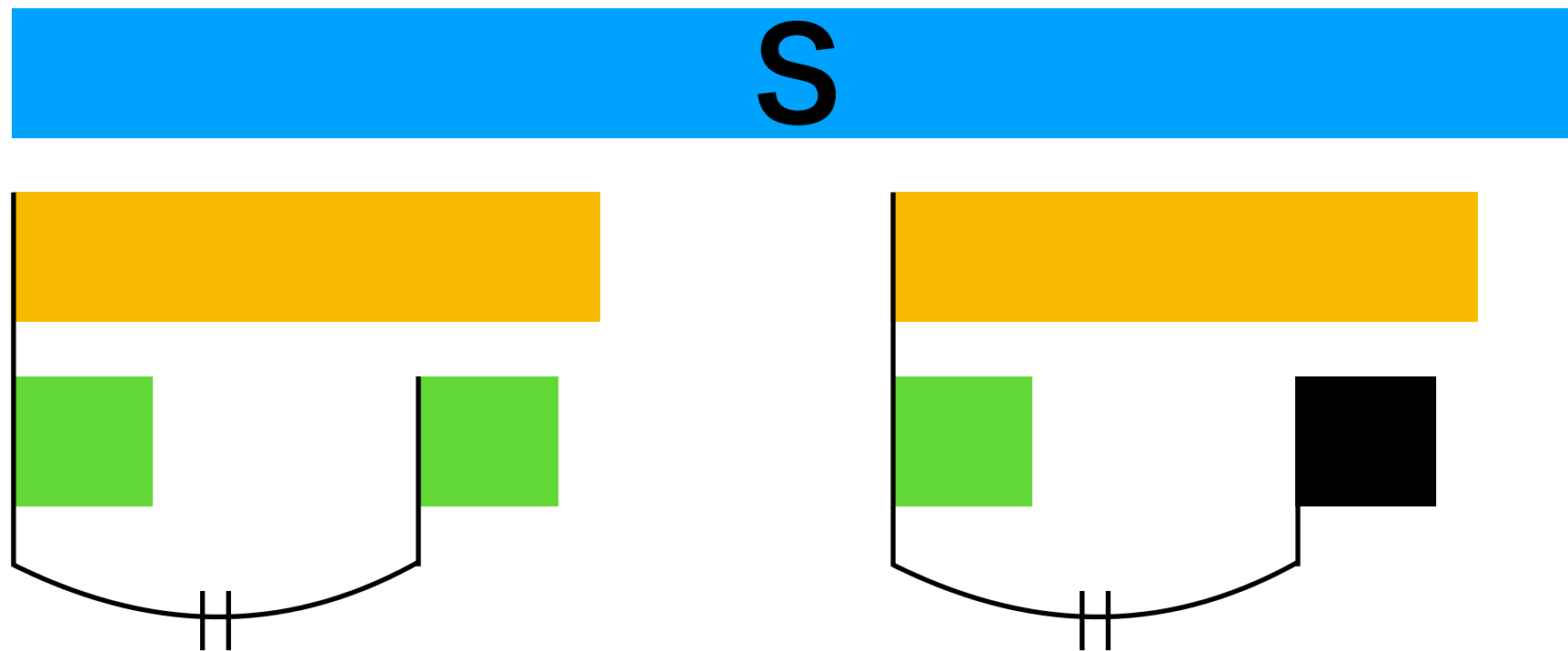
文字列  $S$  が与えられた時、各  $i$  に対して、 $S$  と  $S[i, |S|-1]$  の最長共通接頭辞の長さを  $O(|S|)$  で求めるアルゴリズム

	0	1	2	3	4	5	6	7	8
S	a	a	b	a	a	b	a	a	a
A	9	1	0	5	1	0	2	2	1

$S = \text{"aabaabaaa"}$  と  $S[3,8] = \text{"aabaaa"}$  の最長共通接頭辞は  $\text{"aabaa"}$

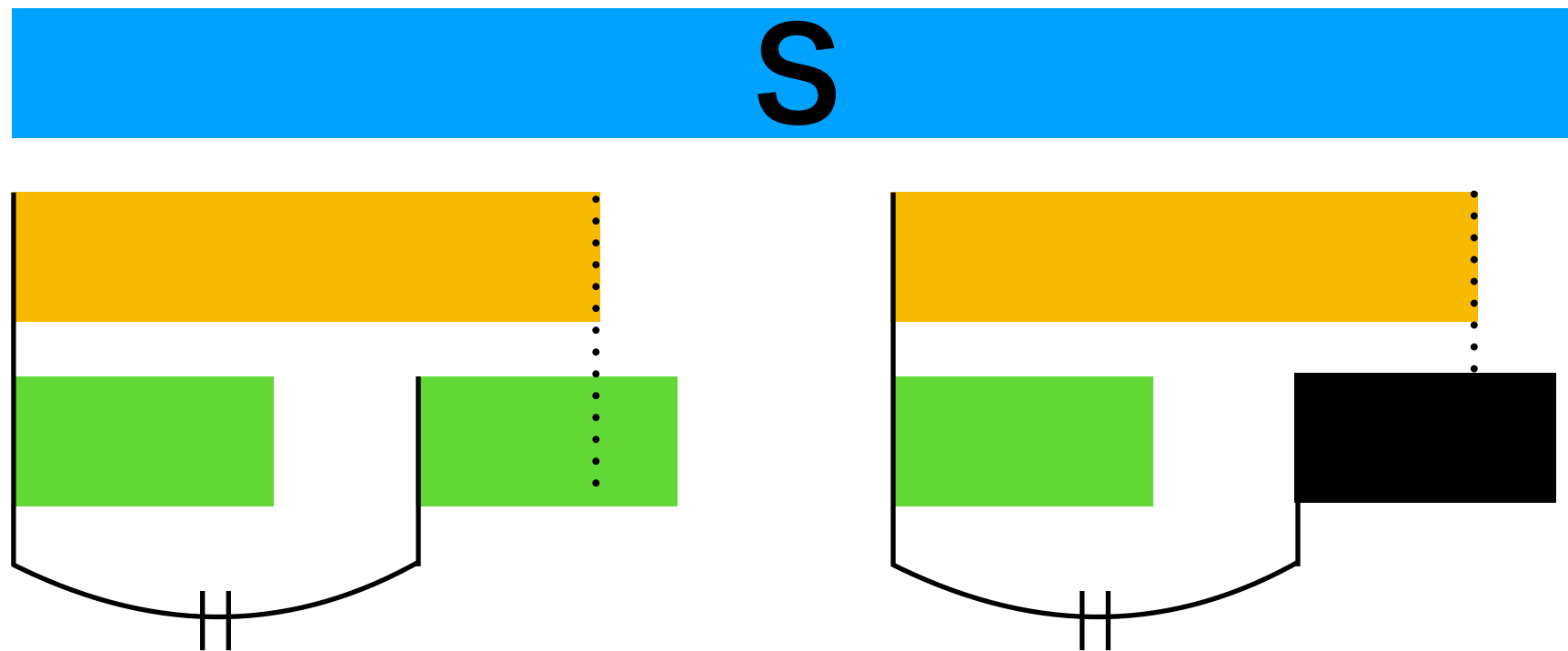
従って  $A[3] = |\text{"aabaa"}| = 5$

# ベースとなるアイデア



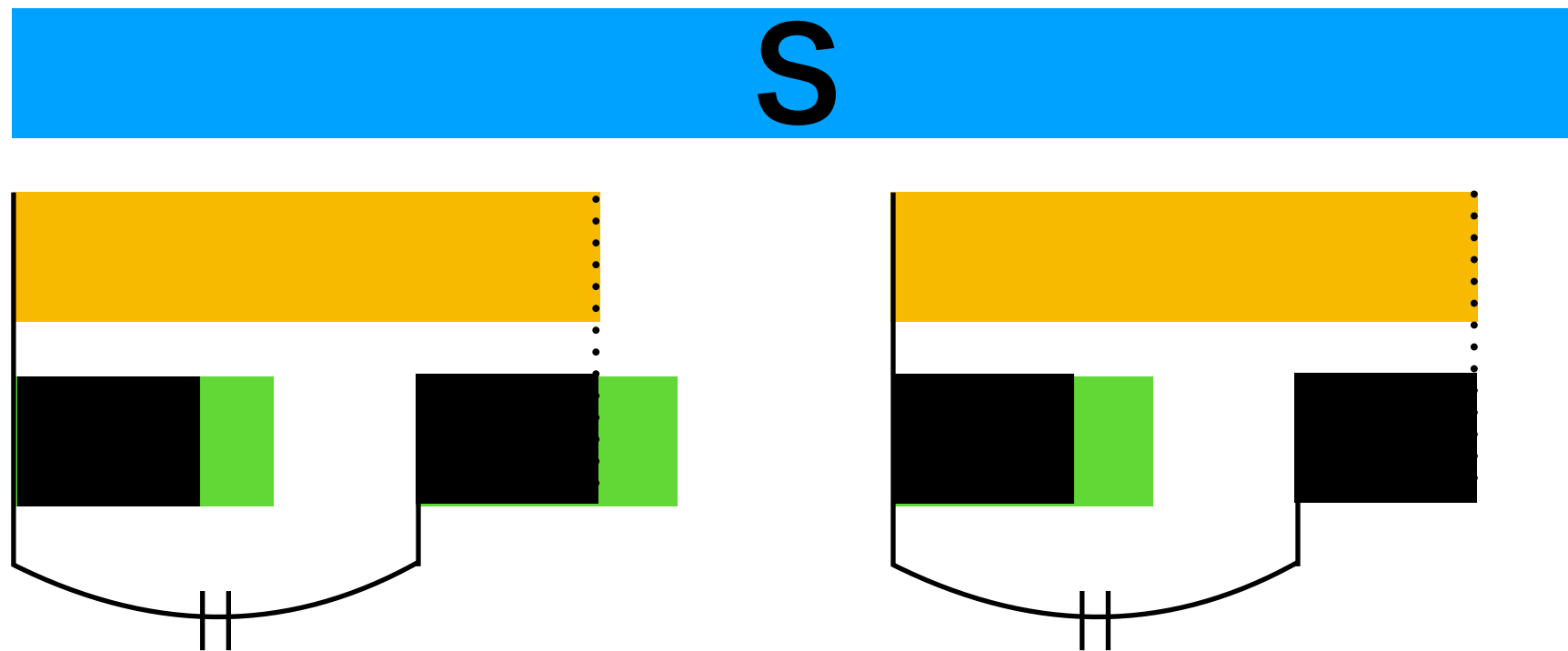
黄色同士が同じ文字列の時、黒と緑は同じ文字列

# ベースとなるアイデア



緑の文字列が黄色の文字列から飛び出している場合、緑と黒が同じとは言えない。  
→黄色の文字列に収まっている部分については同じであると言える。  
飛び出している部分は愚直に比較する。

# ベースとなるアイデア



緑の文字列が黄色の文字列から飛び出している場合、緑と黒が同じとは言えない。  
→黄色の文字列に収まっている部分については同じであると言える。  
飛び出している部分は愚直に比較する。

# ソースコード

```
vector<int> Z_algorithm(string S){  
    int c = 0, n = S.length();  
    vector<int> Z(n,0);  
    for(int i = 1; i < n; ++i){  
        int l = i - c;  
        if(i + Z[l] < c + Z[c]){  
            Z[i] = Z[l];  
        }else{  
            int j = max(0, c + Z[c] - i);  
            while(i + j < n && S[j] == S[i+j]) ++j;  
            Z[i] = j;  
            c = i;  
        }  
    }  
    Z[0] = n;  
    return Z;  
}
```

# Z-algorithm 使用例

求めた配列は結局何に使えるの？

# Z-algorithm 使用例

求めた配列は結局何に使えるの？

→例えば文字列検索に使えます



# Z-algorithm 使用例

求めた配列は結局何に使えるの？

→例えば文字列検索に使えます

文字列検索：テキスト文字列から、パターン文字列を探すこと

# Z-algorithm 使用例

求めた配列は結局何に使えるの？

→例えば文字列検索に使えます

文字列検索：テキスト文字列から、パターン文字列を探すこと

どうやるの？

# Z-algorithm 使用例

求めた配列は結局何に使えるの？

→例えば文字列検索に使えます

文字列検索：テキスト文字列から、パターン文字列を探すこと

どうやるの？

テキスト文字列を  $S$  パターン文字列を  $T$  とすると、

$T + \text{'\#'} + S$  に対して Z-algorithm で配列を求めればよい。

$A[i] = |T|$  となっている  $i$  を見つければ良い。

# MP 法

$A[i] := S[0, i-1]$  の接頭辞と接尾辞が最大何文字一致しているか  
(ただし、 $|S[0, i-1]|$  未満のもののみを考える)  
(この配列  $A$  は最長ボーダーというらしい)

最長ボーダー  $A$  を  $O(|S|)$  で求めるアルゴリズム。

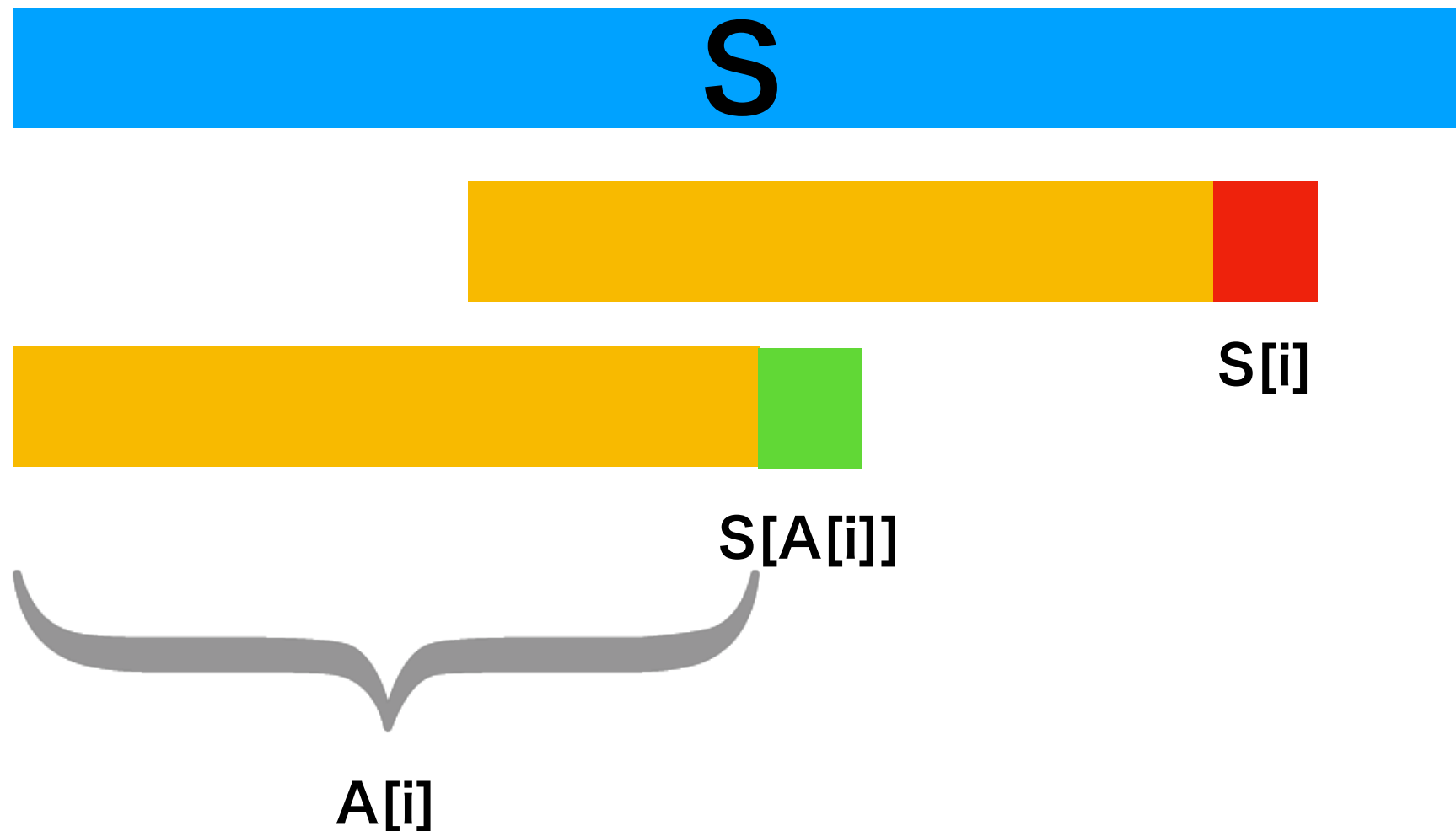
S	a	a	b	a	a	b	a	a	a	
A	-1	0	1	0	1	2	3	4	5	2

$S[0, 5] = \text{aabaab}$

$S[0, 2] = \text{aab} = S[3, 5]$

従って、 $A[6] = 3$

# ベースとなるアイデア



赤と緑が一致している場合  
→  $A[i+1] = A[i] + 1$  とすれば良い

# ベースとなるアイデア



$S[i]$

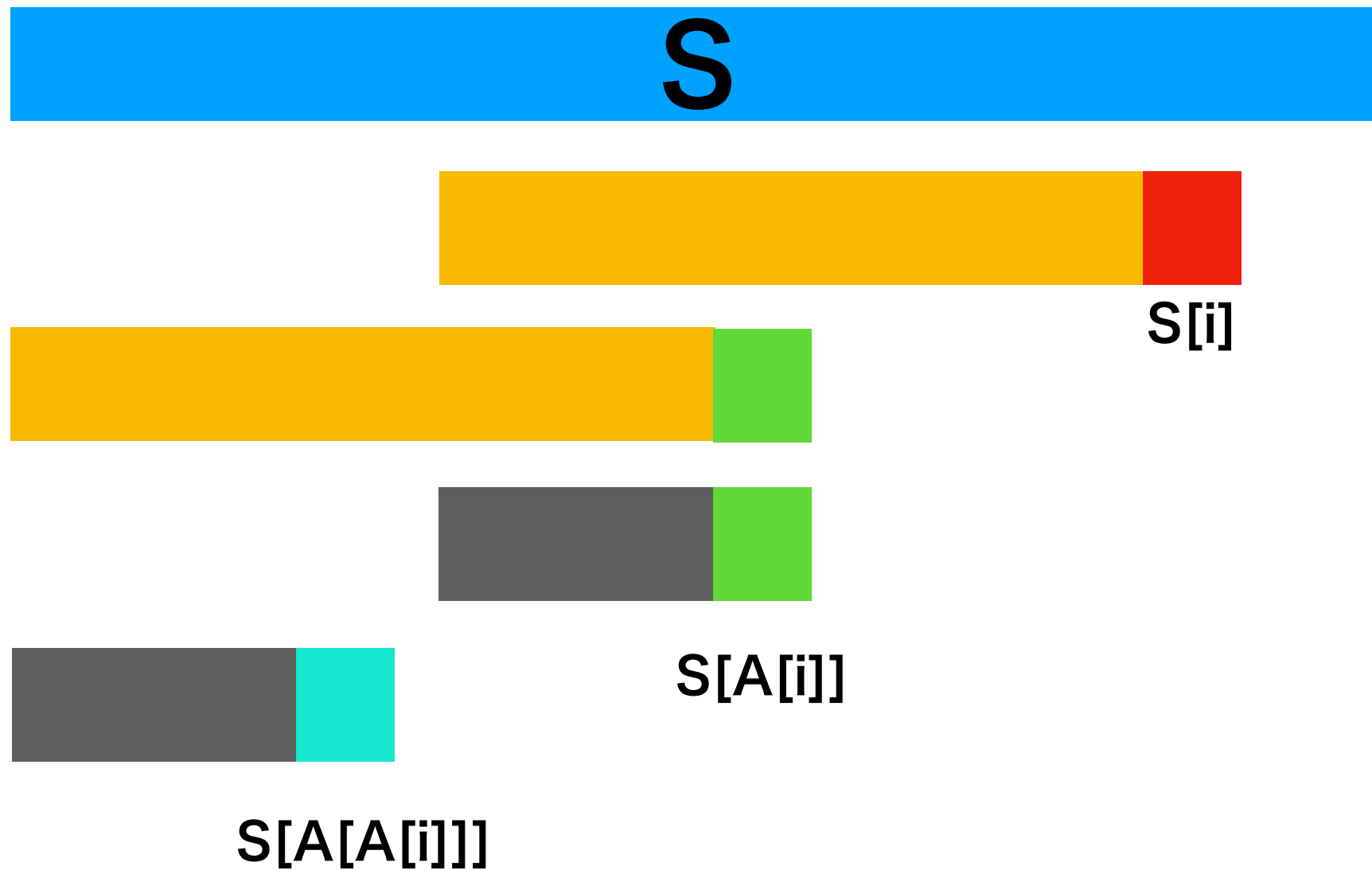


$S[A[i]]$

$S[A[A[i]]]$

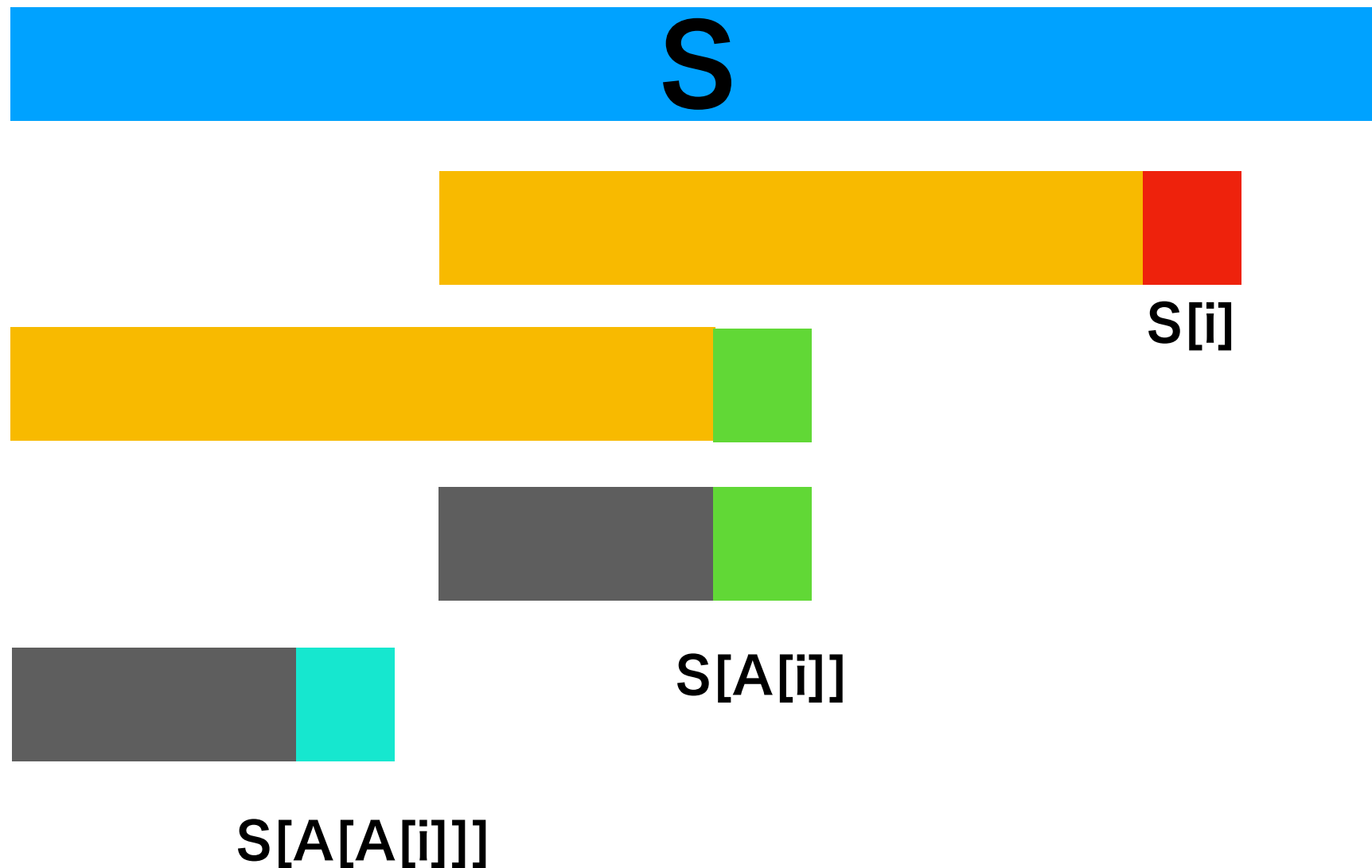
赤と緑が違う場合

# ベースとなるアイデア



赤と緑が違う場合

# ベースとなるアイデア




赤と緑が違う場合

→赤と水色を比較する



# ソースコード



```
vector<int> MP(string S){  
    int l = S.length();  
    vector<int> A(l+1);  
    A[0] = -1;  
    int j = -1;  
    for(int i = 0; i < l; ++i){  
        while(j >= 0 && S[i] != S[j]) j = A[j];  
        ++j;  
        A[i+1] = j;  
    }  
    return A;  
}
```

# 計算量の解析

while 文が何回実行されるかを考える

while 文が実行されると,  $j$  の値は減る  
(-1 より小さくならない)

従って  $j$  の増加量  $\geq j$  の減少量  $\geq$  while 文の実行回数

$j$  が増えるのは  $++j$  しかないので,  $j$  の増加量  $= |S|$   
従って while 文が  $|S|$  より多く実行されることはない  
従って全体で  $O(|S|)$  である

# MP 法

最長ボーダーって結局何に使えるの？

→最長ボーダーを使うことで解ける競プロの問題が色々ある

一般的に有名なのは文字列検索

文字列の周期性の判定も行える

(<http://snuke.hatenablog.com/entry/2015/04/05/184819>)

# MP 法による文字列検索

パターン文字列  $T$  の  $i$  文字目で不一致

→  $i - A[i]$  文字, パターン文字列をずらす

$S = \text{"aaabaabaabaaa"} , T = \text{"aabaabaaa"}$

	0	1	2	3	4	5	6	7	8	9
T	a	a	b	a	a	b	a	a	a	
A	-1	0	1	0	1	2	3	4	5	2

a a a b a a b a a b a a a

a a b a a b a a a

# MP 法による文字列検索

パターン文字列  $T$  の  $i$  文字目で不一致

→  $i - A[i]$  文字, パターン文字列をずらす

$S = \text{"aaabaabaabaaa"} , T = \text{"aabaabaaa"}$

	0	1	2	3	4	5	6	7	8	9
T	a	a	b	a	a	b	a	a	a	
A	-1	0	1	0	1	2	3	4	5	2

一致

a	a	a	b	a	a	b	a	a	b	a	a	a
a	a	b	a	a	b	a	a	a				

# MP 法による文字列検索

パターン文字列  $T$  の  $i$  文字目で不一致

→  $i - A[i]$  文字, パターン文字列をずらす

$S = \text{"aaabaabaabaaa"} , T = \text{"aabaabaaa"}$

	0	1	2	3	4	5	6	7	8	9
T	a	a	b	a	a	b	a	a	a	
A	-1	0	1	0	1	2	3	4	5	2

a a a b a a b a a b a a a

a a b a a b a a a

# MP 法による文字列検索

パターン文字列  $T$  の  $i$  文字目で不一致

→  $i - A[i]$  文字, パターン文字列をずらす

$S = \text{"aaabaabaabaaa"} , T = \text{"aabaabaaa"}$

	0	1	2	3	4	5	6	7	8	9
T	a	a	b	a	a	b	a	a	a	
A	-1	0	1	0	1	2	3	4	5	2

一致

a	a	a	b	a	a	b	a	a	b	a	a	a
a	a	b	a	a	b	a	a	a				

# MP 法による文字列検索

パターン文字列  $T$  の  $i$  文字目で不一致

→  $i - A[i]$  文字, パターン文字列をずらす

$S = \text{"aaabaabaabaaa"} , T = \text{"aabaabaaa"}$

	0	1	2	3	4	5	6	7	8	9
T	a	a	b	a	a	b	a	a	a	
A	-1	0	1	0	1	2	3	4	5	2

a a a b a a b a a b a a a

a a b a a b a a a



# MP 法による文字列検索

パターン文字列  $T$  の  $i$  文字目で不一致

→  $i - A[i]$  文字, パターン文字列をずらす

$S = \text{"aaabaabaabaaa"} , T = \text{"aabaabaaa"}$

	0	1	2	3	4	5	6	7	8	9
T	a	a	b	a	a	b	a	a	a	
A	-1	0	1	0	1	2	3	4	5	2

不一致

a	a	a	b	a	a	b	a	a	b	a	a	a
a	a	b	a	a	b	a	a	a				

$2 - A[2] = 1$  文字ずらす

# MP 法による文字列検索

パターン文字列  $T$  の  $i$  文字目で不一致

→  $i - A[i]$  文字, パターン文字列をずらす

$S = \text{"aaabaabaabaaa"} , T = \text{"aabaabaaa"}$

	0	1	2	3	4	5	6	7	8	9
T	a	a	b	a	a	b	a	a	a	
A	-1	0	1	0	1	2	3	4	5	2

a a a b a a b a a b a a a

a a b a a b a a a

# MP 法による文字列検索

パターン文字列  $T$  の  $i$  文字目で不一致

→  $i - A[i]$  文字, パターン文字列をずらす

$S = \text{"aaabaabaabaaa"} , T = \text{"aabaabaaa"}$

	0	1	2	3	4	5	6	7	8	9
T	a	a	b	a	a	b	a	a	a	
A	-1	0	1	0	1	2	3	4	5	2

a a a b a a b a a b a a a

a a b a a b a a a

# MP 法による文字列検索

パターン文字列  $T$  の  $i$  文字目で不一致

→  $i - A[i]$  文字, パターン文字列をずらす

$S = \text{"aaabaabaabaaa"} , T = \text{"aabaabaaa"}$

	0	1	2	3	4	5	6	7	8	9
T	a	a	b	a	a	b	a	a	a	
A	-1	0	1	0	1	2	3	4	5	2

一致

a	a	a	b	a	a	b	a	a	b	a	a	a
a	a	a	b	a	a	b	a	a	a			

# MP 法による文字列検索

パターン文字列  $T$  の  $i$  文字目で不一致

→  $i - A[i]$  文字, パターン文字列をずらす

$S = \text{"aaabaabaabaaa"} , T = \text{"aabaabaaa"}$

	0	1	2	3	4	5	6	7	8	9
T	a	a	b	a	a	b	a	a	a	
A	-1	0	1	0	1	2	3	4	5	2

a a a b a a b a a b a a a

a a b a a b a a a

# MP 法による文字列検索

パターン文字列  $T$  の  $i$  文字目で不一致

→  $i - A[i]$  文字, パターン文字列をずらす

S = "aaabaabaabaaa", T = "aabaabaaa"

	0	1	2	3	4	5	6	7	8	9
T	a	a	b	a	a	b	a	a	a	
A	-1	0	1	0	1	2	3	4	5	2

# 一致

Diagram illustrating a sequence of characters (a, b) arranged in two rows. The first row contains 13 characters: a, a, a, b, a, a, b, a, a, b, a, a, a. The second row contains 9 characters: a, a, b, a, a, b, a, a, a. A vertical rectangle highlights the third column, which contains 'a' in the first row and 'b' in the second row.

# MP 法による文字列検索

パターン文字列  $T$  の  $i$  文字目で不一致

→  $i - A[i]$  文字, パターン文字列をずらす

$S = \text{"aaabaabaabaaa"} , T = \text{"aabaabaaa"}$

	0	1	2	3	4	5	6	7	8	9
T	a	a	b	a	a	b	a	a	a	
A	-1	0	1	0	1	2	3	4	5	2

a a a b a a b a a b a a a

a a b a a b a a a

# MP 法による文字列検索

パターン文字列  $T$  の  $i$  文字目で不一致

→  $i - A[i]$  文字, パターン文字列をずらす

$S = \text{"aaabaabaabaaa"} , T = \text{"aabaabaaa"}$

	0	1	2	3	4	5	6	7	8	9
T	a	a	b	a	a	b	a	a	a	
A	-1	0	1	0	1	2	3	4	5	2

一致

a	a	a	b	a	a	b	a	a	b	a	a	a
				a	a	b	a	a	a			



# MP 法による文字列検索

パターン文字列  $T$  の  $i$  文字目で不一致

→  $i - A[i]$  文字, パターン文字列をずらす

$S = \text{"aaabaabaabaaa"} , T = \text{"aabaabaaa"}$

	0	1	2	3	4	5	6	7	8	9
T	a	a	b	a	a	b	a	a	a	
A	-1	0	1	0	1	2	3	4	5	2

a a a b a a b a a b a a a

a a b a a b a a a

# MP 法による文字列検索

パターン文字列  $T$  の  $i$  文字目で不一致

→  $i - A[i]$  文字, パターン文字列をずらす

$S = \text{"aaabaabaabaaa"} , T = \text{"aabaabaaa"}$

	0	1	2	3	4	5	6	7	8	9
T	a	a	b	a	a	b	a	a	a	
A	-1	0	1	0	1	2	3	4	5	2

一致

a	a	a	b	a	a	b	a	a	b	a	a	a
					a	b	a	a	a			
			a	a	b	a	a	a				

# MP 法による文字列検索

パターン文字列  $T$  の  $i$  文字目で不一致

→  $i - A[i]$  文字, パターン文字列をずらす

$S = \text{"aaabaabaabaaa"} , T = \text{"aabaabaaa"}$

	0	1	2	3	4	5	6	7	8	9
T	a	a	b	a	a	b	a	a	a	
A	-1	0	1	0	1	2	3	4	5	2

a a a b a a b a a b a a a

a a b a a b a a a

# MP 法による文字列検索

パターン文字列  $T$  の  $i$  文字目で不一致

→  $i - A[i]$  文字, パターン文字列をずらす

$S = \text{"aaabaabaabaaa"} , T = \text{"aabaabaaa"}$

	0	1	2	3	4	5	6	7	8	9
T	a	a	b	a	a	b	a	a	a	
A	-1	0	1	0	1	2	3	4	5	2

一致

a	a	a	b	a	a	b	a	a	b	a	a	a
						b						
			a	a	b	a	a	a				

# MP 法による文字列検索

パターン文字列  $T$  の  $i$  文字目で不一致

→  $i - A[i]$  文字, パターン文字列をずらす

$S = \text{"aaabaabaabaaa"} , T = \text{"aabaabaaa"}$

	0	1	2	3	4	5	6	7	8	9
T	a	a	b	a	a	b	a	a	a	
A	-1	0	1	0	1	2	3	4	5	2

a a a b a a b a a b a a a

a a b a a b a a a

# MP 法による文字列検索

パターン文字列  $T$  の  $i$  文字目で不一致

→  $i - A[i]$  文字, パターン文字列をずらす

$S = \text{"aaabaabaabaaa"} , T = \text{"aabaabaaa"}$

	0	1	2	3	4	5	6	7	8	9
T	a	a	b	a	a	b	a	a	a	
A	-1	0	1	0	1	2	3	4	5	2

一致

a	a	a	b	a	a	b	a	a	b	a	a	a
							a	a	a			

# MP 法による文字列検索

パターン文字列  $T$  の  $i$  文字目で不一致

→  $i - A[i]$  文字, パターン文字列をずらす

$S = \text{"aaabaabaabaaa"} , T = \text{"aabaabaaa"}$

	0	1	2	3	4	5	6	7	8	9
T	a	a	b	a	a	b	a	a	a	
A	-1	0	1	0	1	2	3	4	5	2

a a a b a a b a a b a a a

a a b a a b a a a

# MP 法による文字列検索

パターン文字列  $T$  の  $i$  文字目で不一致

→  $i - A[i]$  文字, パターン文字列をずらす

S = "aaabaabaabaaa", T = "aabaabaaa"

	0	1	2	3	4	5	6	7	8	9
T	a	a	b	a	a	b	a	a	a	
A	-1	0	1	0	1	2	3	4	5	2

# 一致

a a a b a a b a a b a a a  
a a b a a b a a a



# MP 法による文字列検索

パターン文字列  $T$  の  $i$  文字目で不一致

→  $i - A[i]$  文字, パターン文字列をずらす

$S = \text{"aaabaabaabaaa"} , T = \text{"aabaabaaa"}$

	0	1	2	3	4	5	6	7	8	9
T	a	a	b	a	a	b	a	a	a	
A	-1	0	1	0	1	2	3	4	5	2

a a a b a a b a a b a a a

a a b a a b a a a

# MP 法による文字列検索

パターン文字列  $T$  の  $i$  文字目で不一致

→  $i - A[i]$  文字, パターン文字列をずらす

$S = \text{"aaabaabaabaaa"} , T = \text{"aabaabaaa"}$

	0	1	2	3	4	5	6	7	8	9
T	a	a	b	a	a	b	a	a	a	
A	-1	0	1	0	1	2	3	4	5	2

不一致

a	a	a	b	a	a	b	a	a	b	a	a	a
									a			

$8 - A[8] = 3$  文字ずらす

# MP 法による文字列検索

パターン文字列  $T$  の  $i$  文字目で不一致

→  $i - A[i]$  文字, パターン文字列をずらす

$S = \text{"aaabaabaabaaa"} , T = \text{"aabaabaaa"}$

	0	1	2	3	4	5	6	7	8	9
T	a	a	b	a	a	b	a	a	a	
A	-1	0	1	0	1	2	3	4	5	2

a a a b a a b a a b a a a

a a b a a b a a a

# MP 法による文字列検索

パターン文字列  $T$  の  $i$  文字目で不一致

→  $i - A[i]$  文字, パターン文字列をずらす

$S = \text{"aaabaabaabaaa"} , T = \text{"aabaabaaa"}$

	0	1	2	3	4	5	6	7	8	9
T	a	a	b	a	a	b	a	a	a	
A	-1	0	1	0	1	2	3	4	5	2

a a a b a a b a a b a a a

a a b a a b a a a

# MP 法による文字列検索

パターン文字列  $T$  の  $i$  文字目で不一致

→  $i - A[i]$  文字, パターン文字列をずらす

$S = \text{"aaabaabaabaaa"} , T = \text{"aabaabaaa"}$

	0	1	2	3	4	5	6	7	8	9
T	a	a	b	a	a	b	a	a	a	
A	-1	0	1	0	1	2	3	4	5	2

一致

a	a	a	b	a	a	b	a	a	b	a	a	a
									b	a	a	a

# MP 法による文字列検索

パターン文字列  $T$  の  $i$  文字目で不一致

→  $i - A[i]$  文字, パターン文字列をずらす

$S = \text{"aaabaabaabaaa"} , T = \text{"aabaabaaa"}$

	0	1	2	3	4	5	6	7	8	9
T	a	a	b	a	a	b	a	a	a	
A	-1	0	1	0	1	2	3	4	5	2

a a a b a a b a a b a a a

a a b a a b a a a

# MP 法による文字列検索

パターン文字列  $T$  の  $i$  文字目で不一致

→  $i - A[i]$  文字, パターン文字列をずらす

S = "aaabaabaabaaa", T = "aabaabaaa"

	0	1	2	3	4	5	6	7	8	9
T	a	a	b	a	a	b	a	a	a	
A	-1	0	1	0	1	2	3	4	5	2

# 一致

a a a b a a b a a b a a a  
a a b a a b a a a

# MP 法による文字列検索

パターン文字列  $T$  の  $i$  文字目で不一致

→  $i - A[i]$  文字, パターン文字列をずらす

$S = \text{"aaabaabaabaaa"} , T = \text{"aabaabaaa"}$

	0	1	2	3	4	5	6	7	8	9
T	a	a	b	a	a	b	a	a	a	
A	-1	0	1	0	1	2	3	4	5	2

a a a b a a b a a b a a a

a a b a a b a a a



# MP 法による文字列検索

パターン文字列  $T$  の  $i$  文字目で不一致

→  $i - A[i]$  文字, パターン文字列をずらす

S = "aaabaabaabaaa", T = "aabaabaaa"

	0	1	2	3	4	5	6	7	8	9
T	a	a	b	a	a	b	a	a	a	
A	-1	0	1	0	1	2	3	4	5	2

# 一致

a a a b a a b a a b a a a  
a a b a a b a a a

# MP 法による文字列検索

パターン文字列  $T$  の  $i$  文字目で不一致

→  $i - A[i]$  文字, パターン文字列をずらす

$S = \text{"aaabaabaabaaa"} , T = \text{"aabaabaaa"}$

	0	1	2	3	4	5	6	7	8	9
T	a	a	b	a	a	b	a	a	a	
A	-1	0	1	0	1	2	3	4	5	2

a a a b a a b a a b a a a

a a b a a b a a a

# MP 法による文字列検索

パターン文字列  $T$  の  $i$  文字目で不一致

→  $i - A[i]$  文字, パターン文字列をずらす

S = “aaabaabaabaaa”, T = “aabaabaaa”

	0	1	2	3	4	5	6	7	8	9
T	a	a	b	a	a	b	a	a	a	
A	-1	0	1	0	1	2	3	4	5	2

一致

a	a	a	b	a	a	b	a	a	b	a	a	a
				a	a	b	a	a	b	a	a	a

# MP 法による文字列検索

パターン文字列  $T$  の  $i$  文字目で不一致

→  $i - A[i]$  文字, パターン文字列をずらす

$S = \text{"aaabaabaabaaa"} , T = \text{"aabaabaaa"}$

	0	1	2	3	4	5	6	7	8	9
T	a	a	b	a	a	b	a	a	a	
A	-1	0	1	0	1	2	3	4	5	2

a a a b a a b a a b a a a

a a b a a b a a a

# MP 法による文字列検索

パターン文字列  $T$  の  $i$  文字目で不一致

→  $i - A[i]$  文字, パターン文字列をずらす

$S = \text{"aaabaabaabaaa"} , T = \text{"aabaabaaa"}$

	0	1	2	3	4	5	6	7	8	9
T	a	a	b	a	a	b	a	a	a	
A	-1	0	1	0	1	2	3	4	5	2

a a a b a a b a a b a a a

# MP 法による文字列検索

テキスト文字列中の見ている場所が後ろに下がることはない  
→全体で  $O(|S| + |T|)$

# KMP 法

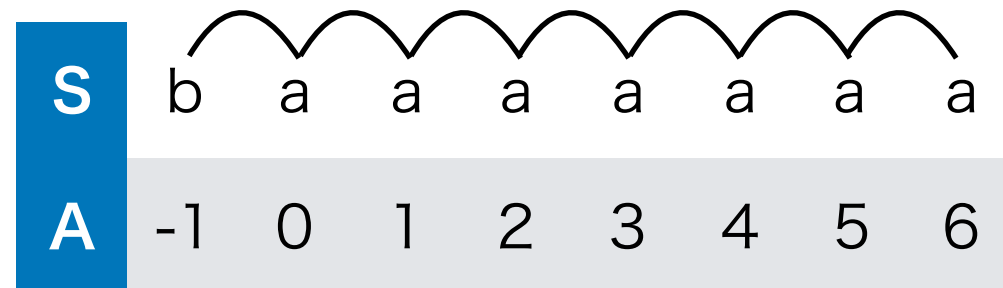
Knuth + MP で KMP

MP を高速化したもの。

パターン文字列をずらす時に賢くずらす。

最悪時のオーダーは MP と変わらず  $O(|S| + |T|)$

# MP との違い

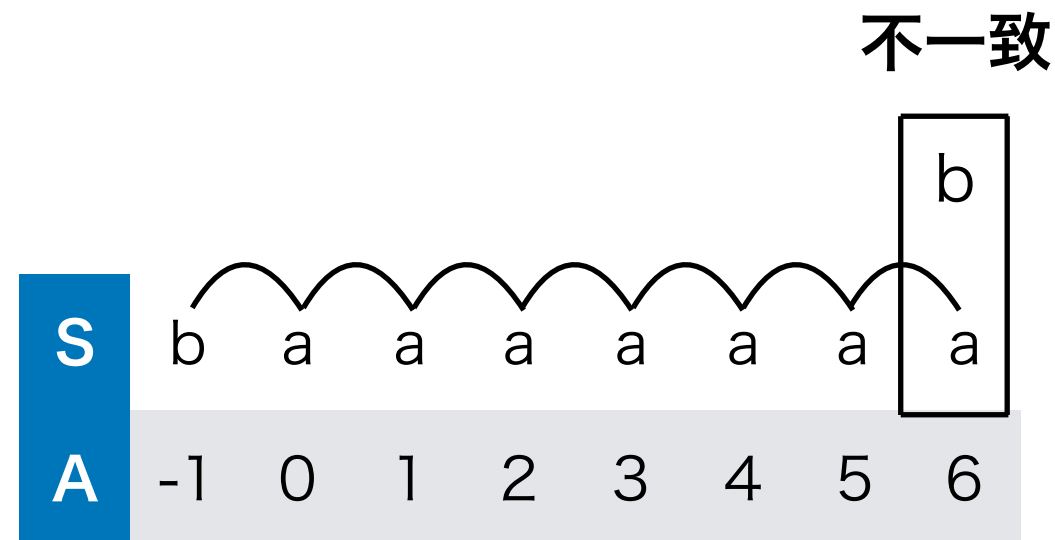


MP を用いた文字列検索で、 $i$  文字目で不一致になった時、次に比較する場所に黒い線が伸びています

a ではないことがわかっているのに、a と比較するのは無駄  
a 以外の文字が出てくるところまで、飛ばしてしまって良い  
7 文字目で不一致になったら次は 0 文字目を見るようにする



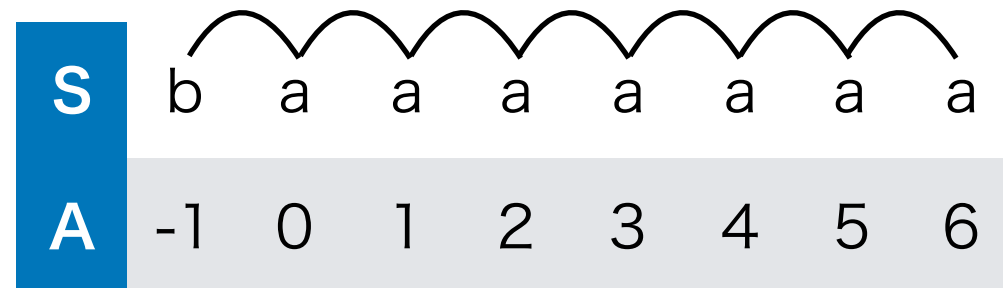
# MP との違い



MP を用いた文字列検索で、 $i$  文字目で不一致になった時、次に比較する場所に黒い線が伸びています

a ではないことがわかっているのに、a と比較するのは無駄  
a 以外の文字が出てくるところまで、飛ばしてしまって良い  
7 文字目で不一致になったら次は 0 文字目を見るようにする

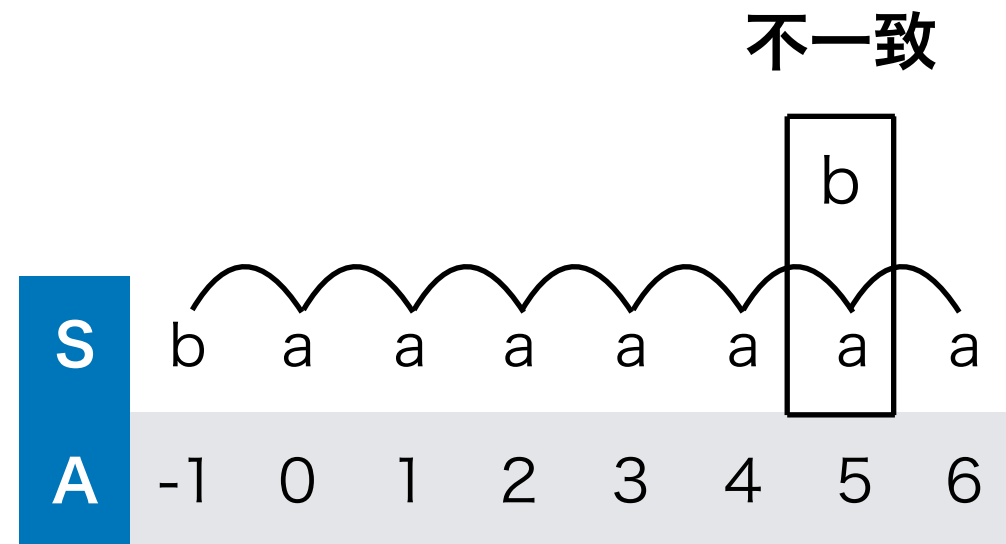
# MP との違い



MP を用いた文字列検索で、 $i$  文字目で不一致になった時、次に比較する場所に黒い線が伸びています

a ではないことがわかっているのに、a と比較するのは無駄  
a 以外の文字が出てくるところまで、飛ばしてしまって良い  
7 文字目で不一致になったら次は 0 文字目を見るようにする

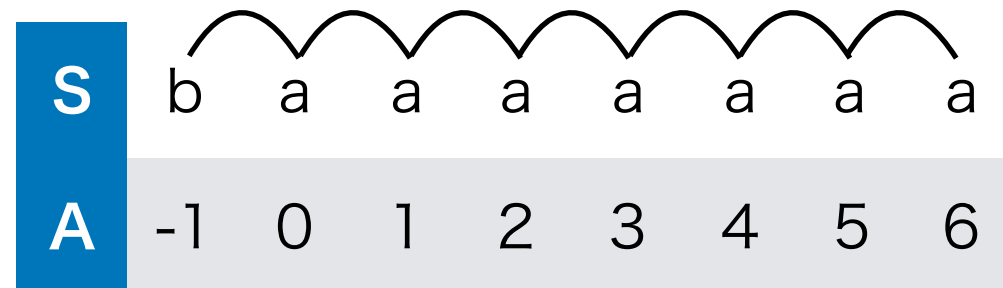
# MP との違い



MP を用いた文字列検索で、 $i$  文字目で不一致になった時、次に比較する場所に黒い線が伸びています

a ではないことがわかっているのに、a と比較するのは無駄  
a 以外の文字が出てくるところまで、飛ばしてしまって良い  
7 文字目で不一致になったら次は 0 文字目を見るようにする

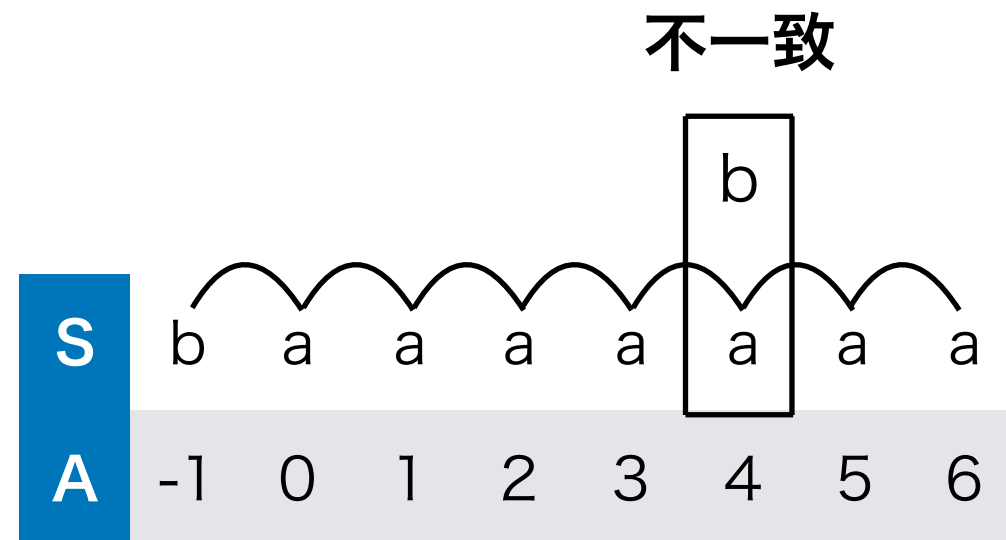
# MP との違い



MP を用いた文字列検索で、 $i$  文字目で不一致になった時、次に比較する場所に黒い線が伸びています

a ではないことがわかっているのに、a と比較するのは無駄  
a 以外の文字が出てくるところまで、飛ばしてしまって良い  
7 文字目で不一致になったら次は 0 文字目を見るようにする

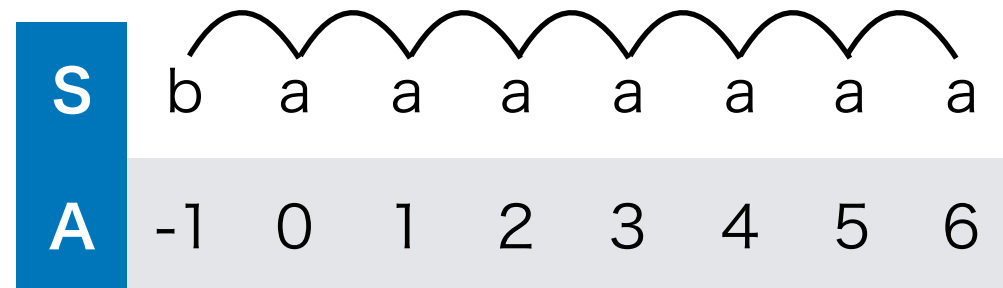
# MP との違い



MP を用いた文字列検索で、 $i$  文字目で不一致になった時、次に比較する場所に黒い線が伸びています

$a$  ではないことがわかっているのに、 $a$  と比較するのは無駄  
 $a$  以外の文字が出てくるところまで、飛ばしてしまって良い  
7 文字目で不一致になったら次は 0 文字目を見るようにする

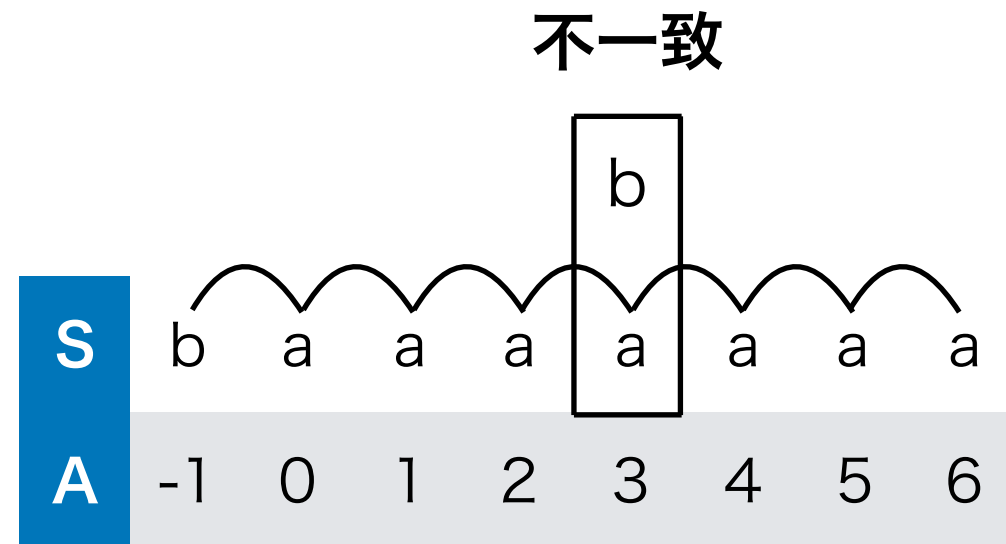
# MP との違い



MP を用いた文字列検索で、 $i$  文字目で不一致になった時、次に比較する場所に黒い線が伸びています

a ではないことがわかっているのに、a と比較するのは無駄  
a 以外の文字が出てくるところまで、飛ばしてしまって良い  
7 文字目で不一致になったら次は 0 文字目を見るようにする

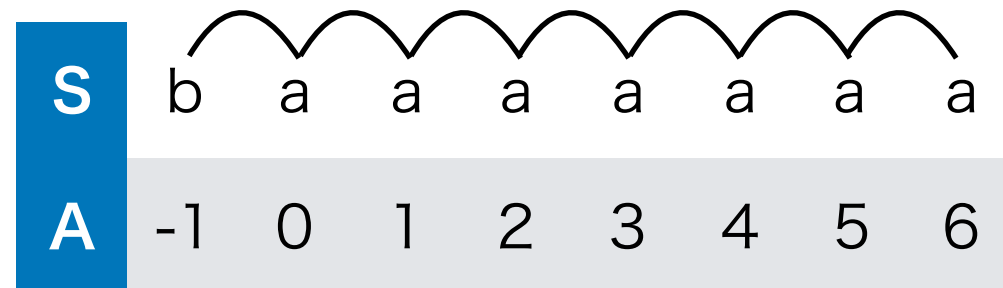
# MP との違い



MP を用いた文字列検索で、 $i$  文字目で不一致になった時、次に比較する場所に黒い線が伸びています

$a$  ではないことがわかっているのに、 $a$  と比較するのは無駄  
 $a$  以外の文字が出てくるところまで、飛ばしてしまって良い  
7 文字目で不一致になったら次は 0 文字目を見るようにする

# MP との違い

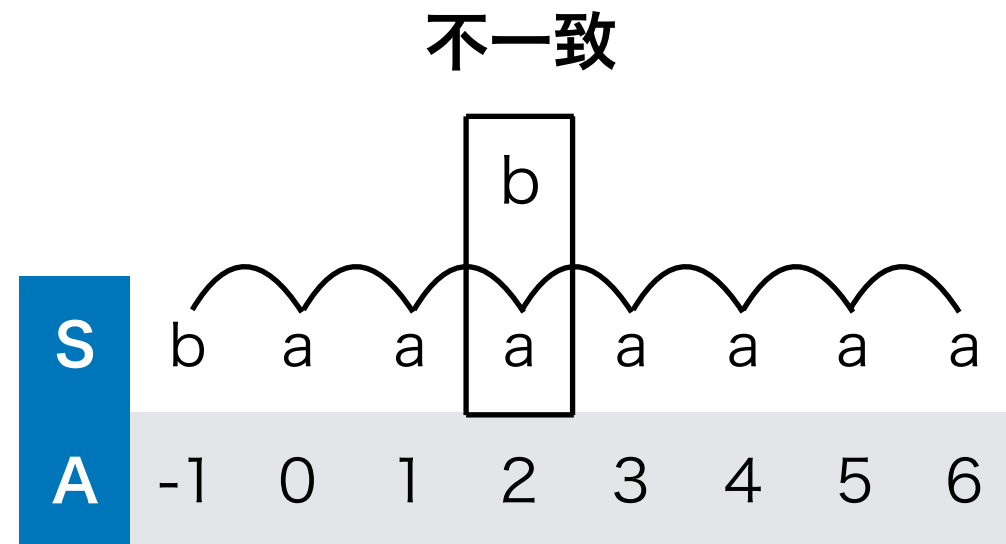


MP を用いた文字列検索で、 $i$  文字目で不一致になった時、次に比較する場所に黒い線が伸びています

a ではないことがわかっているのに、a と比較するのは無駄  
a 以外の文字が出てくるところまで、飛ばしてしまって良い  
7 文字目で不一致になったら次は 0 文字目を見るようにする



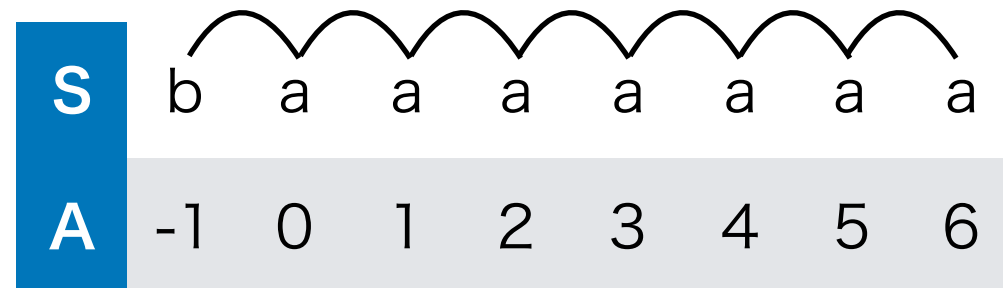
# MP との違い



MP を用いた文字列検索で、 $i$  文字目で不一致になった時、次に比較する場所に黒い線が伸びています

a ではないことがわかっているのに、a と比較するのは無駄  
a 以外の文字が出てくるところまで、飛ばしてしまって良い  
7 文字目で不一致になったら次は 0 文字目を見るようにする

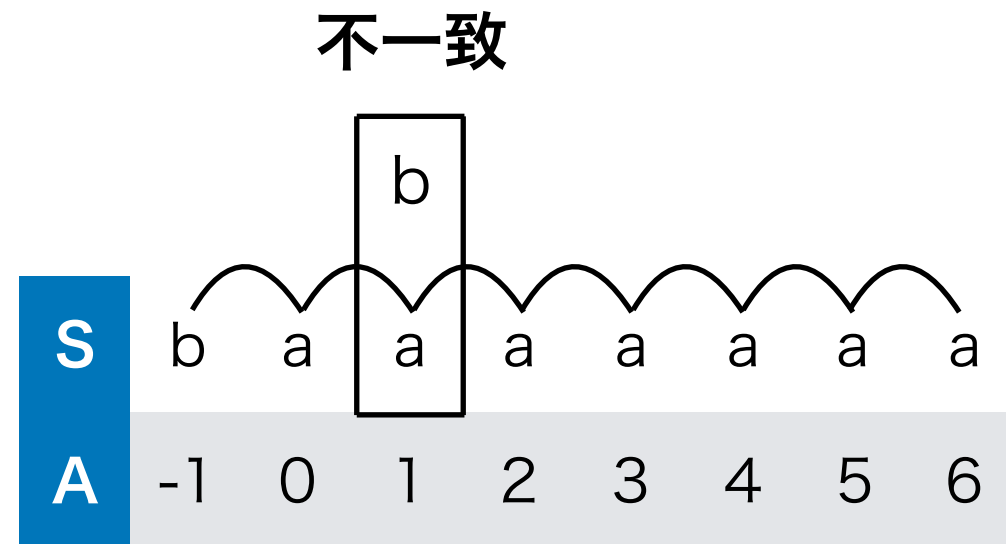
# MP との違い



MP を用いた文字列検索で、 $i$  文字目で不一致になった時、次に比較する場所に黒い線が伸びています

a ではないことがわかっているのに、a と比較するのは無駄  
a 以外の文字が出てくるところまで、飛ばしてしまって良い  
7 文字目で不一致になったら次は 0 文字目を見るようにする

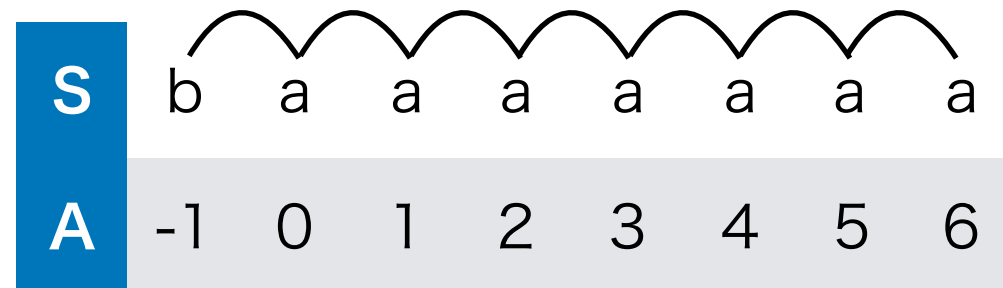
# MP との違い



MP を用いた文字列検索で、 $i$  文字目で不一致になった時、次に比較する場所に黒い線が伸びています

a ではないことがわかっているのに、a と比較するのは無駄  
a 以外の文字が出てくるところまで、飛ばしてしまって良い  
7 文字目で不一致になったら次は 0 文字目を見るようにする

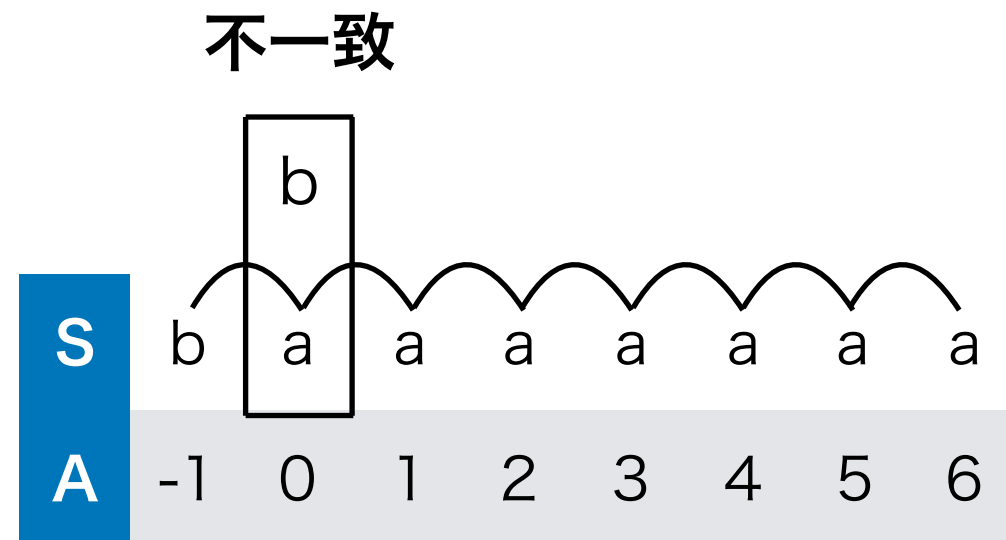
# MP との違い



MP を用いた文字列検索で、 $i$  文字目で不一致になった時、次に比較する場所に黒い線が伸びています

a ではないことがわかっているのに、a と比較するのは無駄  
a 以外の文字が出てくるところまで、飛ばしてしまって良い  
7 文字目で不一致になったら次は 0 文字目を見るようにする

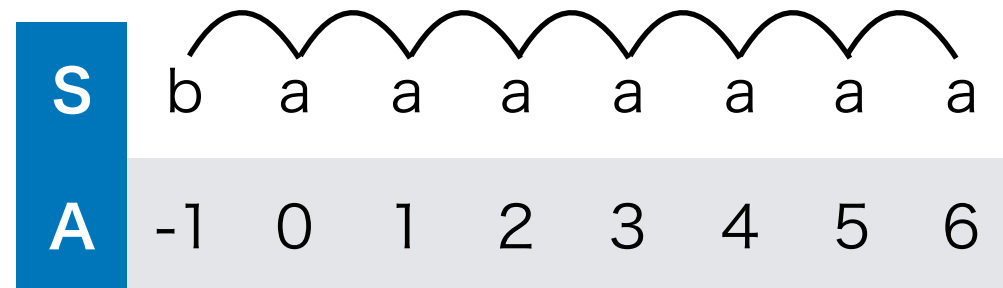
# MP との違い



MP を用いた文字列検索で、 $i$  文字目で不一致になった時、次に比較する場所に黒い線が伸びています

a ではないことがわかっているのに、a と比較するのは無駄  
a 以外の文字が出てくるところまで、飛ばしてしまって良い  
7 文字目で不一致になったら次は 0 文字目を見るようにする

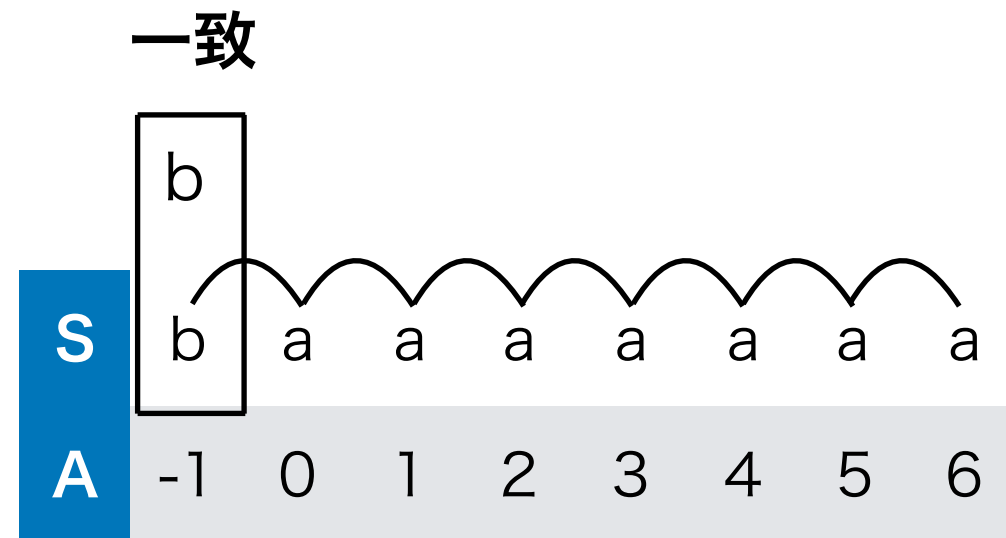
# MP との違い



MP を用いた文字列検索で、 $i$  文字目で不一致になった時、次に比較する場所に黒い線が伸びています

a ではないことがわかっているのに、a と比較するのは無駄  
a 以外の文字が出てくるところまで、飛ばしてしまって良い  
7 文字目で不一致になったら次は 0 文字目を見るようにする

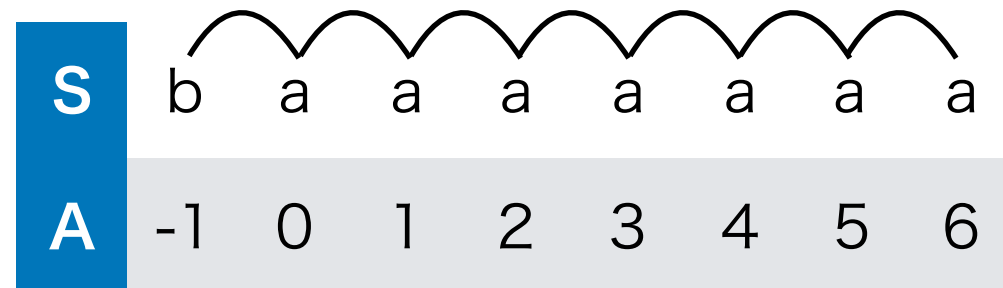
# MP との違い



MP を用いた文字列検索で、 $i$  文字目で不一致になった時、次に比較する場所に黒い線が伸びています

a ではないことがわかっているのに、a と比較するのは無駄  
a 以外の文字が出てくるところまで、飛ばしてしまって良い  
7 文字目で不一致になったら次は 0 文字目を見るようにする

# MP との違い

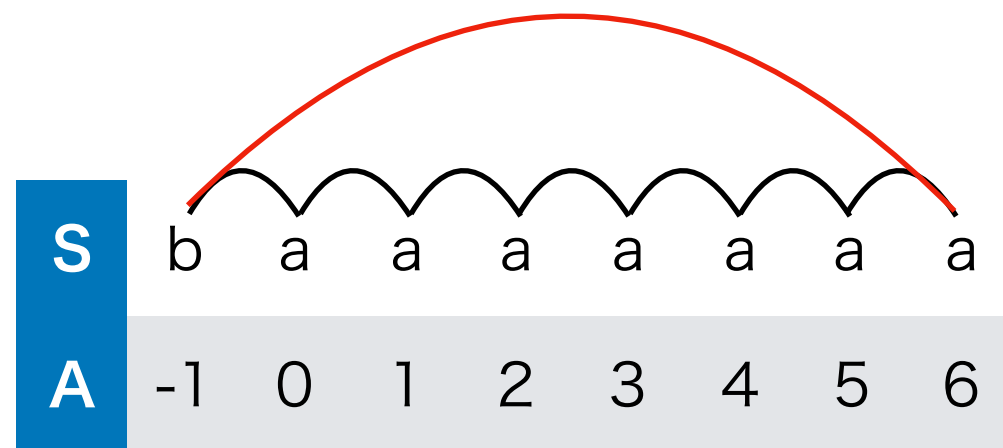


MP を用いた文字列検索で、 $i$  文字目で不一致になった時、次に比較する場所に黒い線が伸びています

a ではないことがわかっているのに、a と比較するのは無駄  
a 以外の文字が出てくるところまで、飛ばしてしまって良い  
7 文字目で不一致になったら次は 0 文字目を見るようにする




# MP との違い



MP を用いた文字列検索で、i 文字目で不一致になった時、次に比較する場所に黒い線が伸びています

a ではないことがわかっているのに、a と比較するのは無駄  
a 以外の文字が出てくるところまで、飛ばしてしまって良い  
7 文字目で不一致になったら次は 0 文字目を見るようにする

# ソースコード



```
vector<int> KMP(string S){
    int l = S.length();
    vector<int> A(l+1);
    A[0] = -1;
    int j = -1;
    for(int i = 0; i < l; ++i){
        while(j >= 0 && S[i] != S[j]) j = A[j];
        ++j;
        if(i < l && S[i+1] == S[j]) A[i+1] = A[j];
        else A[i+1] = j;
    }
    return A;
}
```

# 練習問題

- <http://codeforces.com/contest/914/problem/F>
- <http://codeforces.com/contest/808/problem/G>
- [https://beta.atcoder.jp/contests/jag2018summer-day2/tasks/jag2018summer\\_day2\\_h](https://beta.atcoder.jp/contests/jag2018summer-day2/tasks/jag2018summer_day2_h)
- [https://beta.atcoder.jp/contests/ukuku09/tasks/ukuku09\\_d](https://beta.atcoder.jp/contests/ukuku09/tasks/ukuku09_d)
- [https://beta.atcoder.jp/contests/arc058/tasks/arc058\\_d](https://beta.atcoder.jp/contests/arc058/tasks/arc058_d)