夏のLA 2016

# 連長圧縮に基づく漸増的最長共通部分列アルゴリズム

稲永 俊介 (九州大学)
Heikki Hyyrö (タンペレ大学, フィンランド)

# 背景


Genkai Island
玄界島
Hakata ~ Hitakatsu ~ Izuhara ~ Gonoura ~ Ashibe
Mt. Myoken
妙見山
Hakata ~ Hitakatsu ~ Izuha
Nokonoshima Island Park
のこのしま
アイランドパーク
Shima Seaside CC
志摩シーサイド
カンツリークラブ
Amagadake
天ケ岳
Bishamonyama
毘沙門山
Koshidake
柑子岳
Kazan
火山
Keya GC
芥屋ゴルフ倶楽部
Tateishisan
立石山
九州大学 伊都キャンパス
NISHI WARD
Kaya Hospital
伊都キャンパス
Nobezaki
Takasuzan
高祖山
Shiroyama
城山
Miyajidake
宮地岳
Fukuoka-Maebaru Road
The Queens Hill GC
ザ・クイーンズ
Itoshima
Google
Heilongjiang
Harbin
哈尔滨
Changchun
长春
Jilin
Vladivostok
Владивосток
Sapporo
札幌
Liaoning
North Korea
Sea of Japan
Pyongyang
평양
Seoul
서울
Niigata
新潟
Sendai
仙台
South Korea
Japan
Gwangju
광주
Busan
부산
Hiroshima
広島
Osaka
大阪
Tokyo
東京
Hamamatsu
浜松
Fukuoka
福岡
九州大学
Jiangsu
Shanghai
上海
Kagoshima
鹿児島
Hangzhou
杭州
East China Sea
Zhejiang

# 背景


伊都キャンパス

糸島

String Island

文字列島

# 文字列 の編集距離計算

**問題 1 (編集距離)**

入力: 文字列 $A$ と文字列 $B$

出力: $A$ と $B$ の編集距離 $ed(A, B)$

- 編集距離 $ed(A, B)$ は，文字列 $A$ を $B$ に変換する編集操作（文字の挿入，削除，置換）の最小回数

# 動的計画法 (Dynamic Programming)

- $m = |A|$, $n = |B|$ とし，$(m+1) \times (n+1)$ の表 $D$ を $D[i, j] = ed(A[1..i], B[1..j])$ と定義する.
- 以下の式によって $D[i, j]$ を順に埋めていくことで, $D[m, n] = ed(A, B)$ を $O(mn)$ 時間で計算できる.
  - $D[i, 0] = i$ for $1 \leq i \leq m$,
  - $D[0, j] = j$ for $1 \leq j \leq n$,
  - $D[i, j] = \min\{ D[i, j\text{-}1]+1, D[i\text{-}1, j]+1, D[i\text{-}1, j\text{-}1] + \delta(A[i], B[j]) \}$,

    ただし $\delta(A[i], B[j]) = 1$ if $A[i] \neq B[j]$,
    $\delta(A[i], B[j]) = 0$ if $A[i] = B[j]$.

# 動的計画法 (Dynamic Programming)

| $D$ | | | $B$ | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | a | t | c | c | g | a | t |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| t | 1 | | | | | | | |
| g | 2 | | | | | | | |
| c | 3 | | | | | | | |
| a | 4 | | | | | | | |
| t | 5 | | | | | | | |
| a | 6 | | | | | | | |
| t | 7 | | | | | | | |

$A$

$A = \texttt{tgcatat}$

$B = \texttt{atccgat}$

$D[i, 0] = i$ for $1 \leq i \leq m$

$D[0, j] = j$ for $1 \leq j \leq n$

# 動的計画法 (Dynamic Programming)

| $D$ | | | $B$ | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | a | t | c | c | g | a | t |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| t | 1 | 1 | 1 | 2 | 3 | 4 | 5 | 6 |
| g | 2 | 2 | 2 | 2 | 3 | 3 | 4 | 5 |
| c | 3 | 3 | 3 | 2 | 2 | 3 | 4 | 5 |
| a | 4 | 3 | 4 | 3 | 3 | 3 | 3 | 4 |
| t | 5 | 4 | 3 | 4 | 4 | 4 | 4 | 3 |
| a | 6 | 5 | 4 | 4 | 5 | 5 | 4 | |
| t | 7 | 6 | 5 | 5 | 5 | 6 | 5 | |

$A$

$A = \texttt{tgcatat}$

$B = \texttt{atccgat}$

$$D[i,j] = \min\{\, D[i,j\text{-}1]+1,\ D[i\text{-}1,j]+1,\ D[i\text{-}1,j\text{-}1]+1 \}$$

# 動的計画法 (Dynamic Programming)

| $D$ | | | $B$ | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | a | t | c | c | g | a | t |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| t | 1 | 1 | 1 | 2 | 3 | 4 | 5 | 6 |
| g | 2 | 2 | 2 | 2 | 3 | 3 | 4 | 5 |
| c | 3 | 3 | 3 | 2 | 2 | 3 | 4 | 5 |
| a | 4 | 3 | 4 | 3 | 3 | 3 | 3 | 4 |
| t | 5 | 4 | 3 | 4 | 4 | 4 | 4 | 3 |
| a | 6 | 5 | 4 | 4 | 5 | 5 | 4 | 4 |
| t | 7 | 6 | 5 | 5 | 5 | 6 | 5 | |

$A$

$A = \texttt{tgcatat}$

$B = \texttt{atccgat}$

$$D[i,j] = \min\{\, D[i,j\text{-}1]+1,\ D[i\text{-}1,j]+1,\ D[i\text{-}1,j\text{-}1]+1 \}$$

# 動的計画法 (Dynamic Programming)

$D$ $B$

| | | a | t | c | c | g | a | t |
|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| t | 1 | 1 | 1 | 2 | 3 | 4 | 5 | 6 |
| g | 2 | 2 | 2 | 2 | 3 | 3 | 4 | 5 |
| c | 3 | 3 | 3 | 2 | 2 | 3 | 4 | 5 |
| a | 4 | 3 | 4 | 3 | 3 | 3 | 3 | 4 |
| t | 5 | 4 | 3 | 4 | 4 | 4 | 4 | 3 |
| a | 6 | 5 | 4 | 4 | 5 | 5 | 4 | 4 |
| t | 7 | 6 | 5 | 5 | 5 | 6 | 5 | |

$A$

$A = \texttt{tgcatat}$

$B = \texttt{atccgat}$

$$D[i,j] = \min\{\, D[i,j\text{-}1]+1,\ D[i\text{-}1,j]+1,\ D[i\text{-}1,j\text{-}1]\,\}$$

# 動的計画法 (Dynamic Programming)

| $D$ | | | $B$ | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | a | t | c | c | g | a | t |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| t | 1 | 1 | 1 | 2 | 3 | 4 | 5 | 6 |
| g | 2 | 2 | 2 | 2 | 3 | 3 | 4 | 5 |
| $A$ c | 3 | 3 | 3 | 2 | 2 | 3 | 4 | 5 |
| a | 4 | 3 | 4 | 3 | 3 | 3 | 3 | 4 |
| t | 5 | 4 | 3 | 4 | 4 | 4 | 4 | 3 |
| a | 6 | 5 | 4 | 4 | 5 | 5 | 4 | 4 |
| t | 7 | 6 | 5 | 5 | 5 | 6 | 5 | 4 |

$A = \texttt{tgcatat}$

$B = \texttt{atccgat}$

$$D[i,j] = \min\{\, D[i,j\text{-}1]+1,\ D[i\text{-}1,j]+1,\ D[i\text{-}1,j\text{-}1] \,\}$$

合計 $O(mn)$ 時間

# 循環文字列

- $1 \leq j \leq n$ に対して, $B_j = B[j..n]B[1..j\text{-}1]$ とする.
  すなわち, $B_j$ は $B$ の $j$ 番目の循環文字列である.

# 循環文字列

- $1 \leq j \leq n$ に対して, $B_j = B[j..n]B[1..j\text{-}1]$ とする.
  すなわち, $B_j$ は $B$ の $j$ 番目の循環文字列である.
- 例) 文字列 $B$ = “なつのえるえー” に対して,
  - $B_1$ = なつのえるえー
  - $B_2$ = つのえるえーな
  - $B_3$ = のえるえーなつ
  - $B_4$ = えるえーなつの
  - $B_5$ = るえーなつのえ
  - $B_6$ = えーなつのえる
  - $B_7$ = ーなつのえるえ

# 循環文字列比較

**問題 2 (循環文字列の編集距離)**

入力: 文字列 $A$ と文字列 $B$

出力: $A$ と $B$ のすべての循環文字列 $B_1, \ldots, B_n$ の編集距離 $ed(A, B_j)$.

- 動機 : 生物学的配列の比較など.
- 素朴な方法では, 各循環文字列 $B_j$ に対して $O(mn)$ 時間 ⇒ 合計 $O(mn^2)$ 時間かかる.
- より効率のよい方法はあるだろうか？

# 右端の文字追加は簡単

$B[1..5]$

| | | c | a | g | t | a |
|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 |
| a | 1 | 1 | 1 | 2 | 3 | 4 |
| g | 2 | 2 | 2 | 1 | 2 | 3 |
| c | 3 | 2 | 3 | 2 | 2 | 3 |
| t | 4 | 3 | 3 | 3 | 2 | 3 |
| a | 5 | 4 | 3 | 4 | 3 | 2 |

$A$

$B[1..5]B[1]$

| | | c | a | g | t | a | c |
|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| a | 1 | 1 | 1 | 2 | 3 | 4 | 5 |
| g | 2 | 2 | 2 | 1 | 2 | 3 | 4 |
| c | 3 | 2 | 3 | 2 | 2 | 3 | 3 |
| t | 4 | 3 | 3 | 3 | 2 | 3 | 4 |
| a | 5 | 4 | 3 | 4 | 3 | 2 | 3 |

$A$

- 最後の列の値だけが変化する
  ⇒ 右端の文字追加は $O(m)$ 時間でできる.

# 左端の文字削除は簡単ではない

$B[1..5]B[1]$

| $A$ | | | c | a | g | t | a | c |
|---|---|---|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| a | 1 | 1 | 1 | 2 | 3 | 4 | 5 | |
| g | 2 | 2 | 2 | 1 | 2 | 3 | 4 | |
| c | 3 | 2 | 3 | 2 | 2 | 3 | 3 | |
| t | 4 | 3 | 3 | 3 | 2 | 3 | 4 | |
| a | 5 | 4 | 3 | 4 | 3 | 2 | 3 | |

$B[2..5]B[1]$

| $A$ | | | a | g | t | a | c |
|---|---|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 | 5 |
| a | 1 | 0 | 1 | 2 | 3 | 4 | |
| g | 2 | 1 | 0 | 1 | 2 | 3 | |
| c | 3 | 2 | 1 | 1 | 2 | 2 | |
| t | 4 | 3 | 2 | 1 | 2 | 3 | |
| a | 5 | 4 | 3 | 2 | 1 | 2 | |

- 左端の文字を削除すると，最悪の場合には DP 表の <u>すべての列</u> に影響が及んでしまう！

# 既存手法

| アルゴリズム | 左端削除の時間 | 領域 |
|---|---|---|
| Landau et al. (1998) | $O(m+n)$ | $O(mn)$ |
| Schmidt (1998) | $O(m+n)$ | $O(mn)$ |
| Kim & Park (2004) | $O(m+n)$ | $O(mn)$ |
| Hyyrö et al. (2015) | $O(m+n)$ | $O(mn)$ |

- 前述のとおり，DP 表をそのまま保持すると $O(mn)$ 時間かかってしまう.
- そこで，既存研究では DP 表の非明示的表現（差分表現）を保持し，それを上手く更新している.

# 提案手法

| アルゴリズム | 左端削除の時間 | 領域 |
|---|---|---|
| Landau et al. (1998) | $O(m+n)$ | $O(mn)$ |
| Schmidt (1998) | $O(m+n)$ | $O(mn)$ |
| Kim & Park (2004) | $O(m+n)$ | $O(mn)$ |
| Hyyrö et al. (2016) | $O(m+n)$ | $O(mn)$ |
| **提案手法** | $\boldsymbol{O(m+n)}$ | $\boldsymbol{O(ml+nk)}$ |

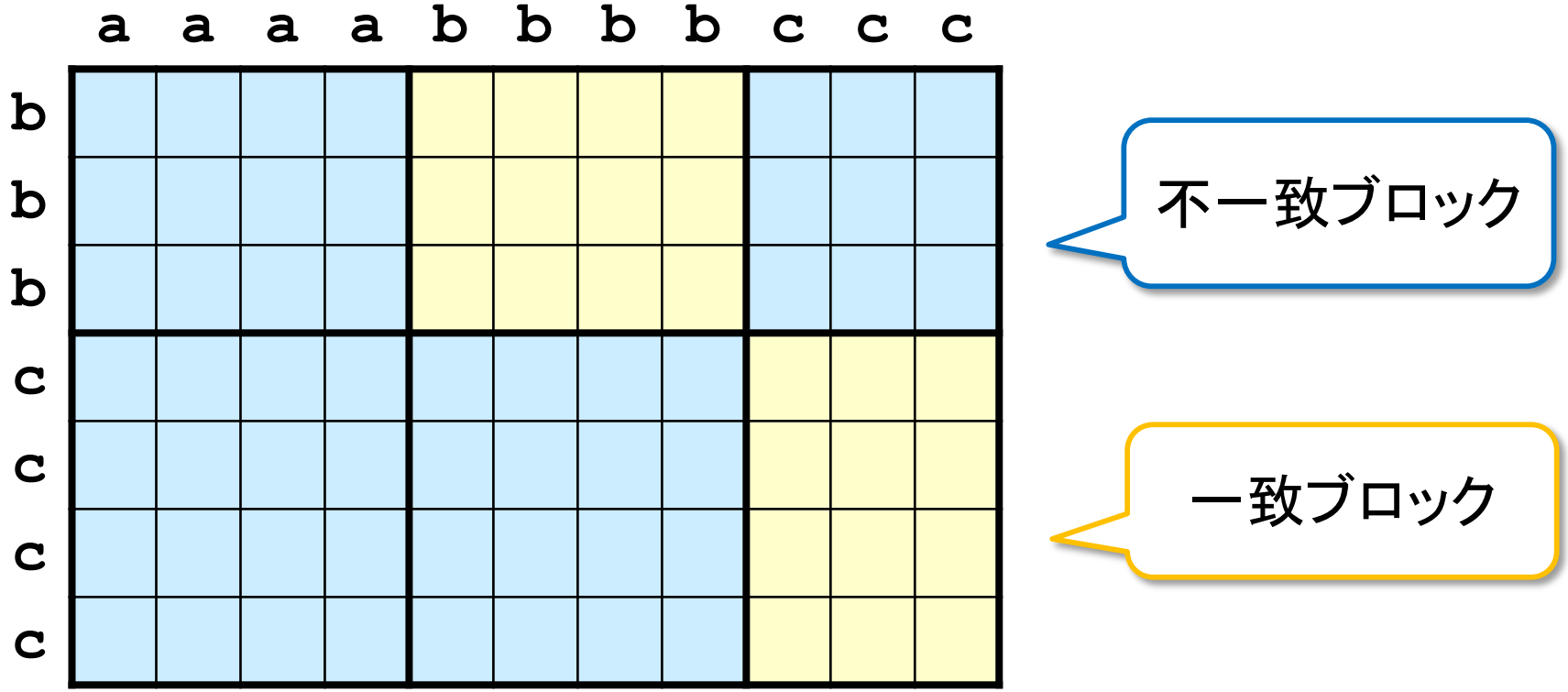
- より省領域な左端削除アルゴリズムを提案する.
- ここで, $k\ (\leq m)$ と $l\ (\leq n)$ はそれぞれ,
  文字列 $A$ と $B$ の 連長圧縮 のサイズ.

# 文字列の連長圧縮 (*RLE*)

- 文字列 $A$ 中の同一文字の連続を，その文字の連続長で表現する圧縮法 (Run Length Encoding, RLE).
  - 例) $RLE(\mathtt{aaabbccccccbb}) = \mathtt{a}^3\mathtt{b}^2\mathtt{c}^5\mathtt{b}^2$
- $RLE(A)$ 中の同一文字の連の個数を，$RLE(A)$ の サイズ という.
  - 上の例では，サイズは 4.
- $m$ を文字列 $A$ の長さ，$k$ を $RLE(A)$ のサイズとすると，必ず $k \leq m$ が成り立つ.

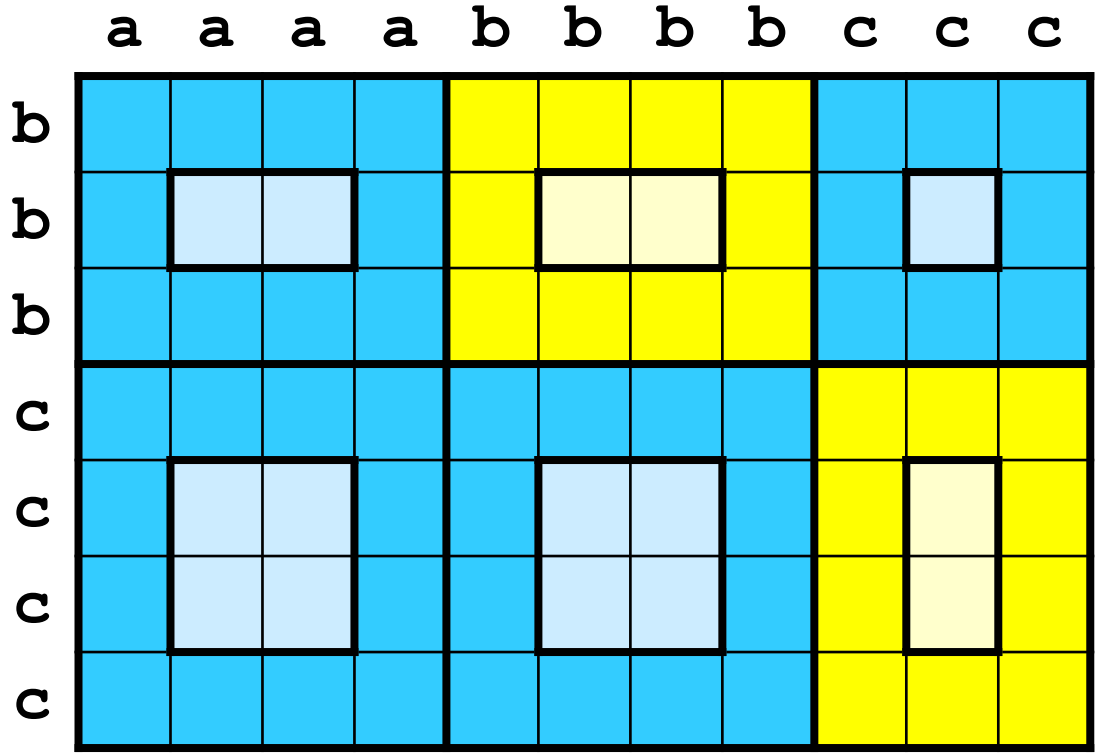
# 連長圧縮文字列の編集距離計算

- $ed(RLE(A), RLE(B))$ の動的計画法の表 $D$ を $kl$ 個のブロックに分割する [Arbel et al. 2002].



# 連長圧縮文字列の編集距離計算

- ブロックの境界の値だけを陽に保持する
  $\Rightarrow O(ml + nk)$ 領域で済む.
- ブロック内の値は，逐次計算できる.




境界に含まれる
セルの個数は
$O(ml + nk)$.

# Key Lemmas

動的計画表の差分表現を $DR$ とし,
左端文字の削除後の差分表現を $DR'$ とする.

**補題 1 [Hyyrö et al., 2016]**

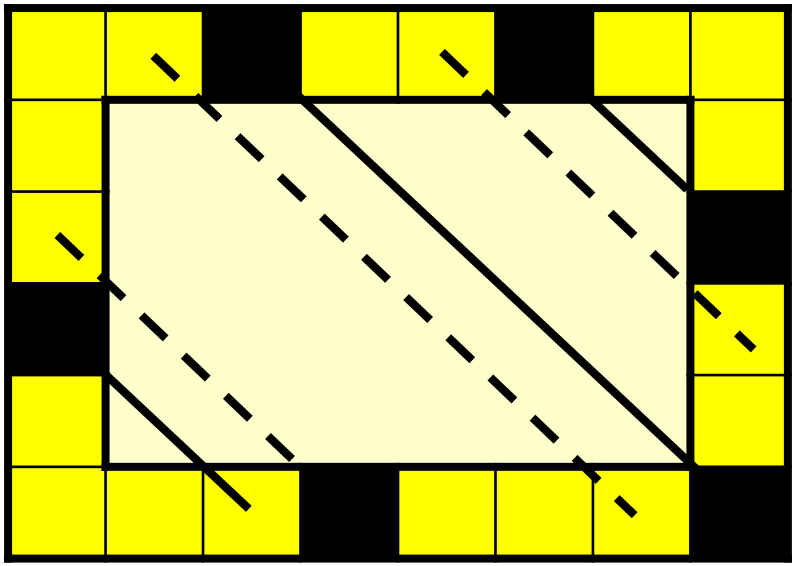
$DR'[i,j] \neq DR[i,j]$ を満たすセルは
表全体で $O(m+n)$ 個しかない.

**補題 2**

各ブロックの上下左右のそれぞれの境界中には,
$DR'[i,j] \neq DR[i,j]$ なるセルが $O(1)$ 個しかない.

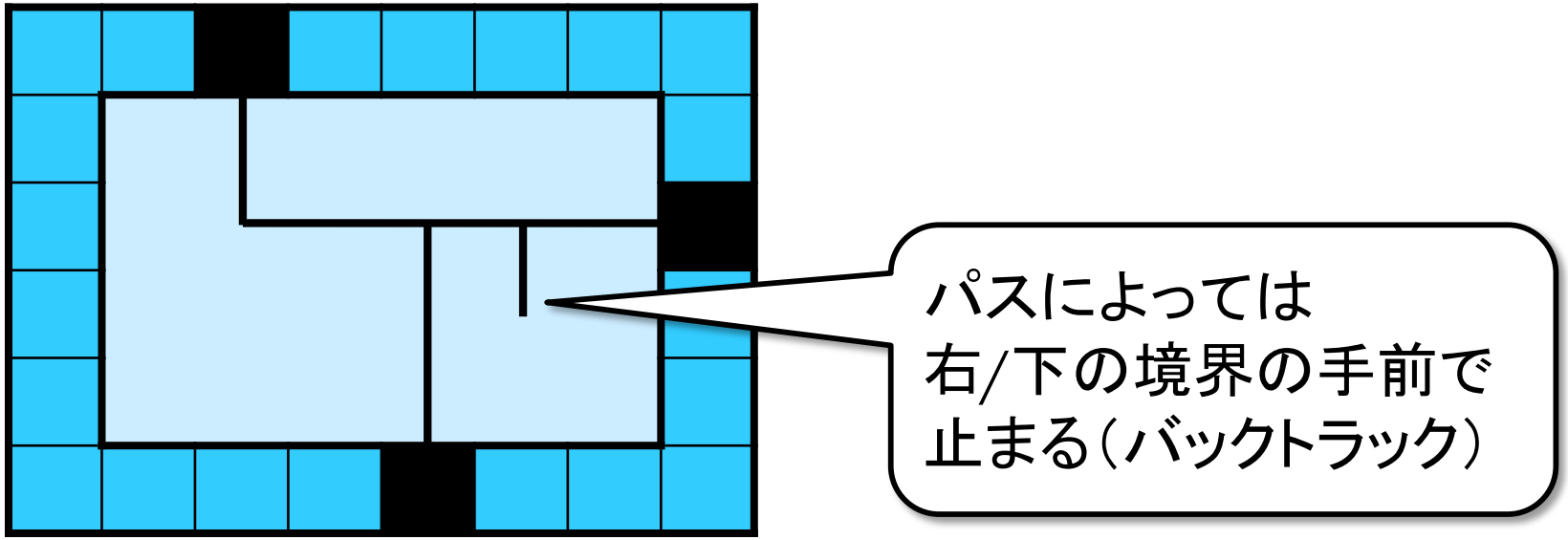
# 一致ブロックの処理

- 動的計画法の表の一致ブロック内では，
  値は対角線に伝搬する．
- よって，その差分表現の一致ブロック内でも，
  $DR'[i,j] \neq DR[i,j]$ なる値は，左／上の境界から
  右／下の境界へと伝搬する．



# 不一致ブロックの処理

- 不一致ブロック内では，$DR'[i,j] \neq DR[i,j]$ を満たすセルのパスは枝分かれする可能性がある.
- 左／上の境界中の $O(1)$ 個の各始点から，深さ優先探索で $DR'[i,j] \neq DR[i,j]$ を満たすパスをトレースする.



# 主結果

**定理**

文字列 $A$ と 文字列 $B$ に対して，以下を満たす編集距離 $ed(A, B)$ の DP 表の差分表現が存在する.

- $O(ml + nk)$ 領域
- 文字列 $B$ の左端文字の削除に対して，$O(m + n)$ 時間で更新可能

証明）前述のアルゴリズムと補題 1, 2 より.

- $m = |A|$
- $n = |B|$
- $k = |RLE(A)|$
- $l = |RLE(B)|$