北海道大学
Hokkaido University

情報知識ネットワーク特論
「情報検索とパターン照合」

情報科学研究科 コンピュータサイエンス専攻
情報知識ネットワーク研究室
喜田拓也

第4回
近似文字列照合

近似文字列照合問題とは？
動的計画法によるアルゴリズム
NFAに基づくアルゴリズム
ビットパラレル手法
フィルタリング手法

こんなときどうする？

バロック時代のあの画家、
なんて言ったっけ？

光と影のコントラストが
大胆な絵で有名な・・・

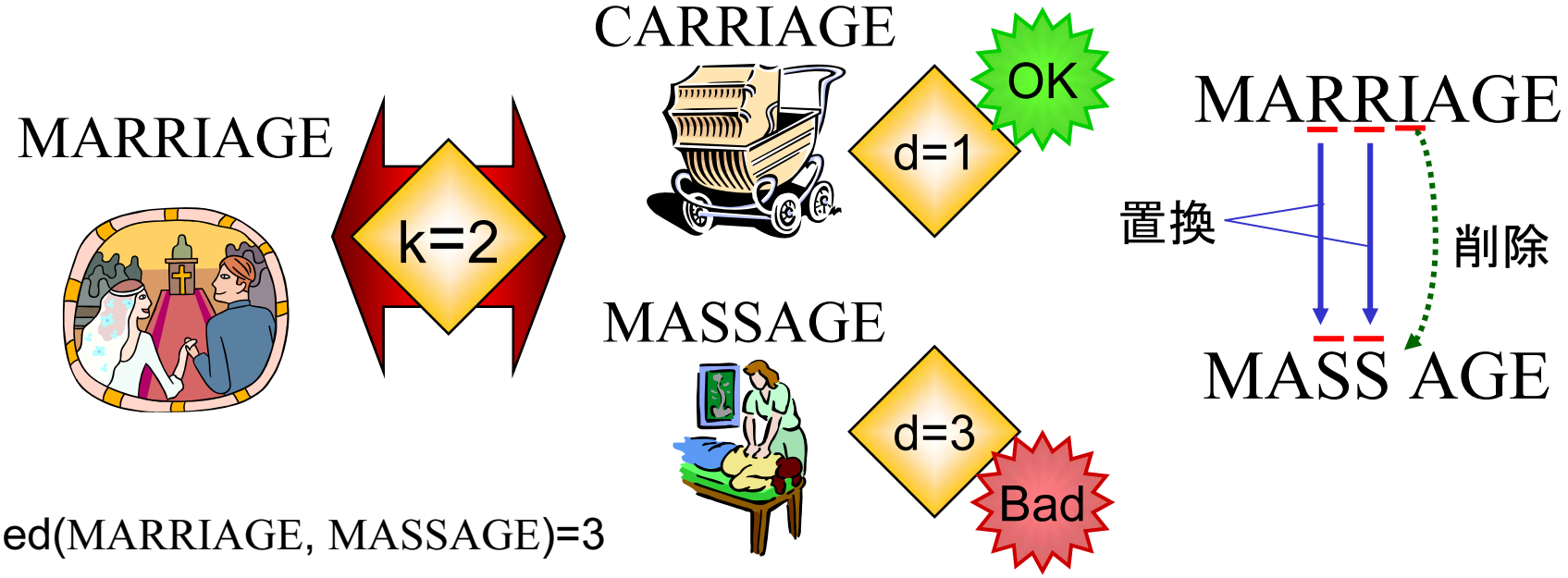
えーっと・・・
確か、カルパッチョ？
みたいな？

カラヴァッジョ
のこと？

近似文字列照合問題とは？

- テキスト中から、パターンとの編集距離が k以内の部分文字列の位置を求める問題
- 編集距離 ed(x, y)
  - 文字列 x に文字の挿入・削除・置換の操作を施して文字列 y へ変換するために要する最小のコスト d

0<k<m



編集距離（Edit distance）

- 二つの文字列がどのくらい似ているか？
  - 類似度 ⇔ 文字列間の編集距離（非類似度）

以降の話は、すべて
Levenshtein距離

- 編集距離の種類
  - Levenshtein 距離　：挿入・削除・置換のコストがすべて 1
  - Hamming 距離　：置換だけを許した編集距離
  - Weighted-cost 編集距離
    ：変換操作ごとにコストが違う
  - Unrestricted-cost 編集距離
    ：文字対ごとに操作コストが違う
  - Damerau 距離　：挿入・削除・置換の操作に加え、
    前後の文字の入れ替えも許す
  - （参考）Indel 距離　：挿入と削除だけを許した距離
    insertion+deletionでindelらしい
    （Heikki Hyyröの[SOFSEM2005]の論文より）

応用例

- DNA配列間の類似度の計算
- スペル・チェッカー
- あいまい検索
  - 表記ゆれを補助： カラヴァッジョ ⇔ カラバッジョ

シソーラスを用いた検索は別もの

- 類似文章の検索
  - 自然言語処理との組み合わせで高度な検索を実現する
  - 文章＝形態素の並び

各形態素をメタ文字と考える

- 類似音楽検索
  - MIDIデータを使って、似ているフレーズを探す
  - 鼻歌検索
- OCRデータに対する検索
  - 対象データ自体に誤りが含まれている！
- 実データのマイニングへの応用
  - 近似文字列照合アルゴリズムを用いたウェブマイニング手法の研究（九州大学 中藤哲也先生のグループ）

動的計画法によるアルゴリズム

- 動的計画法(Dynamic programming)による編集距離の求め方
  - 動的計画法に基づく解法は1960年代にはすでに知られていた
  - 現在知られているパターン照合用のアルゴリズムはSellersによる
    - P. H. Sellers, The theory and computation of evolutionary distances: Pattern recognition. *Journal of Algorithms*, 1(4):359–373,1980.

- まずは、二つの文字列 x, y の編集距離の求め方からはじめる
  - $M_{i,j} = ed(x_{1\cdots i}, y_{1\cdots j})$ とすると

    x[1..i], y[1..j] の意

    $M_{0,0} \leftarrow 0$

    $M_{i,j} \leftarrow \min(M_{i-1,j-1} + \delta(x_i, y_j), M_{i-1,j} + 1, M_{i,j-1} + 1)$

    すなわち $M_{|x|,|y|} = ed(x, y)$

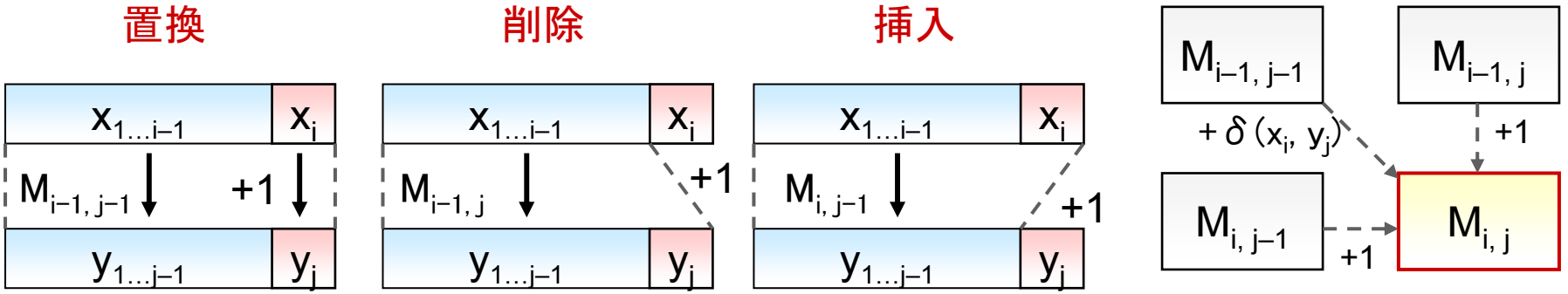
    で計算できる。ここで、$\delta(a,b)$は a=b なら0、そうでなければ1をとる関数。
  - 同等の計算をするより効率の良い再帰式

$$M_{i,0} \leftarrow i, \qquad M_{0,j} \leftarrow j$$

$$M_{i,j} \leftarrow \begin{cases} M_{i-1,j-1} & (x_i = y_j \text{ のとき}) \\ 1 + \min(M_{i-1,j-1}, M_{i-1,j}, M_{i,j-1}) & (\text{それ以外}) \end{cases}$$

どうしてそれで計算できるのか？

- 帰納法で証明する
  - $M_{0,0}$ は二つの空語 ε の編集距離
  - いま、二つの文字列$x_{1\cdots i}$,$y_{1\cdots j}$ の編集距離ed($x_{1\cdots i}$,$y_{1\cdots j}$) を求めたい
  - $x_{1\cdots i}$,$y_{1\cdots j}$ それぞれについて、より短い文字列(prefix)に対する編集距離は既に計算済みであると仮定する。このとき、$x_{1\cdots i}$ から $y_{1\cdots j}$ への変換について考える
  - $x_i = y_j$ のとき、$x_{1\cdots i-1}$ から $y_{1\cdots j-1}$ へコスト $M_{i-1,j-1}$ で変換するのが最小
  - $x_i \neq y_j$ のとき、以下の3通りが考えられる
    - xi を yi で置き換え、$x_{1\cdots i-1}$ を $y_{1\cdots j-1}$ へ変換する → コスト $M_{i-1,j-1}+1$
    - xi を削除し、$x_{1\cdots i-1}$ を $y_{1\cdots j}$ へ変換する → コスト $M_{i-1,j}+1$
    - yi を $x_{1\cdots i}$ の後ろに挿入し、$x_{1\cdots i}$ を $y_{1\cdots j-1}$ へ変換する → コスト $M_{i,j-1}+1$
  - 以上の中で最小のものをとればよい



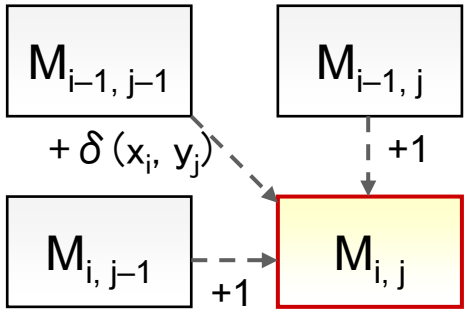
テキスト中の出現を検出するには？

ed(annual, annealing) の $M_{i,j}$

$M_{0,0}$

		a	n	n	e	a	l	i	n	g
	0	1	2	3	4	5	6	7	8	9
a	1	0	1	2	3	4	5	6	7	8
n	2	1	0	1	2	3	4	5	6	7
n	3	2	1	0	1	2	3	4	5	6
u	4	3	2	1	1	2	3	4	5	6
a	5	4	3	2	2	1	2	3	4	5
l	6	5	4	3	3	2	1	2	3	4

$M_{|x|,|y|}$ = ed(annual, annealing) = 4



P=annual, T=annealing, k=2
のときの近似文字列照合

		a	n	n	e	a	l	i	n	g
		0	0	0	0	0	0	0	0	0
a		0	1	1	1	0	1	1	1	1
n		1	0	1	2	1	1	2	1	2
n		2	1	0	1	2	2	2	2	2
u		3	2	1	1	2	3	3	3	3
a		4	3	2	2	1	2	3	4	4
l		5	4	3	3	2	1	2	3	4

任意のj = 0…n について、
$M_{0,j}$ = 0 と置くだけでよい！

空文字列 ε は、テキストの任意の位置に誤り0でマッチする！

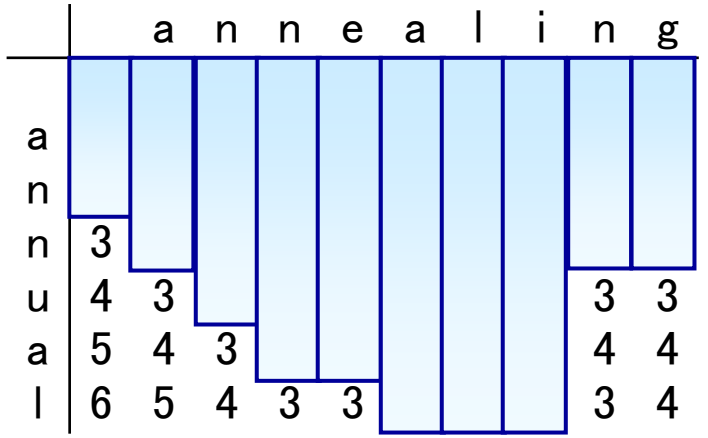
O(mn)時間、O(m)領域

平均時の計算量を改善する方法

- パターンはテキスト中にあまり出現しないと考える
  - 各縦列の上から下へ進む計算過程において、たいていの場合、すぐ k+1 に達してしまう(つまりミスマッチする)
  - 各セルが k+1 より大きいものは、その実際の値は照合結果と無関係
  - k よりも小さい値が入るセルを active と呼び、一番下方にあるactiveなセルまでの領域を計算することで、平均時の計算量をO(kn)へ落とすことができる(DPアルゴリズムと名づける by 黄色本)
    - E. Ukkonen. Finding approximate patterns in strings. Journal of Algorithms, 6(1–3):132–137, 1985.

P=annual, T=annealing, k=2
のときの近似文字列照合

最悪時O(mn)時間、
平均時O(kn)時間



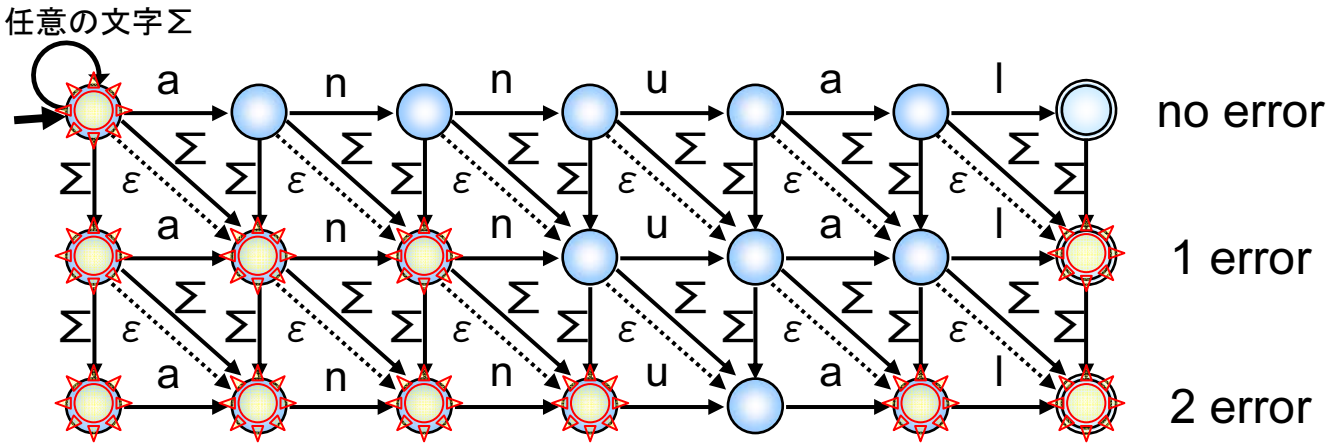
DPアルゴリズムの擬似コード

```
DP (P=p1p2…pm, T=t1t2…tn, k)
1    Preprocessing:
2        For i∈0…m Do Ci ← i
3        lact ← k + 1  /* last active cell */
4    Searching:
5        For pos∈1...n Do
6            pC ← 0, nC ← 0
7            For i ∈ 1…lact Do
8                If pi = tpos Then nC ← pC
9                Else
10                   If pC < nC Then nC ← pC
11                   If Ci < nC Then nC ← Ci
12                   nC ← nC + 1
13               End of if
14               pC ← Ci, Ci ← nC
15           End of for
16           While Clact > k Do lact ← lact – 1
17           If lact = m Then report an occurrence at pos
18           Else lact ← lact + 1
19       End of for
```

オートマトンに基づくアルゴリズム

E. Ukkonen. Finding approximate patterns in strings. *Journal of Algorithms*, 6(1-3):132-137, 1985.

パターン P=annual を誤り2以下で受理する非決定性有限オートマトン

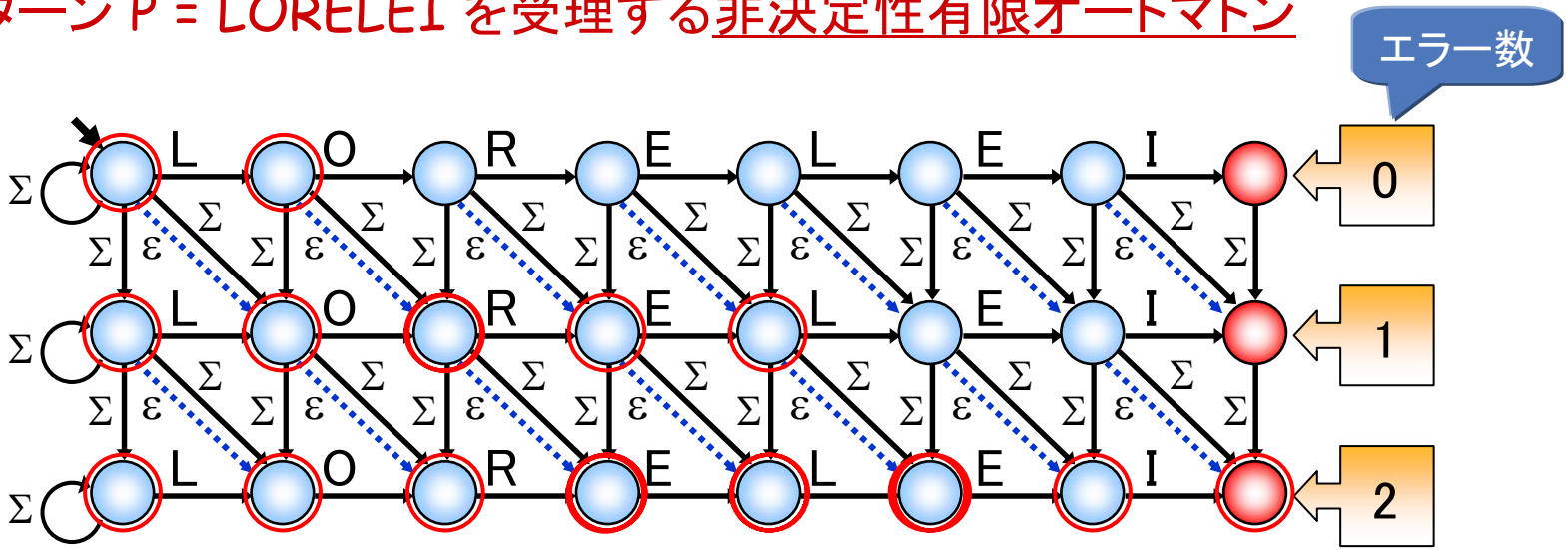


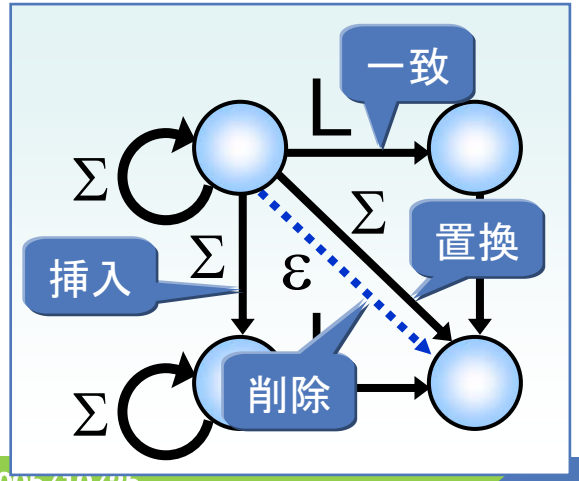
T= anneal を読み込んだときの状態

- このNFAをDFAに変換して、照合を行う
  - オリジナルはUkkonen[1985]、後にいくつかの改良が提案された
    - 古典的な方法で変換するため、状態数が$O(\min(3^m, m(2m|\Sigma|)^k))$ 個に膨れ上がる
    - パターン長が20より大きくなると、現実的ではない

NFAの動作

パターン P = LORELEI を受理する非決定性有限オートマトン



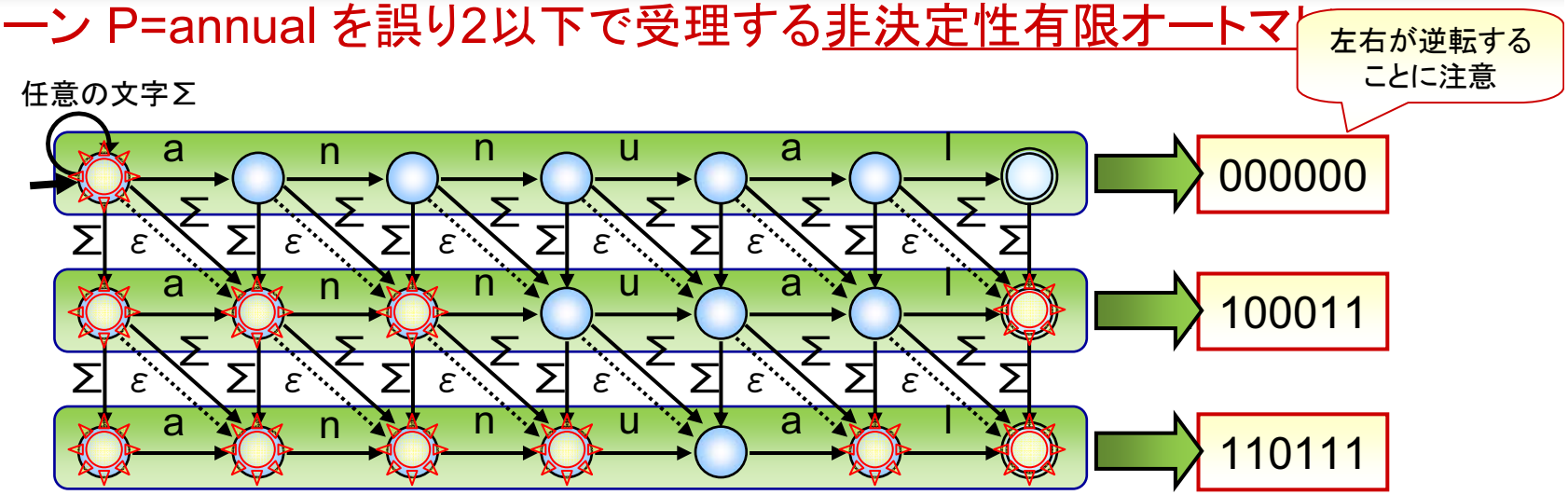


テキスト T = Ｍ Ｏ Ｒ Ｅ ＿ Ｌ Ｅ Ｉ

NFAのRow-wiseビットパラレル (BPR)

S. Wu and U. Manber. Fast text searching allowing errors. Communications of the ACM, 35(10): 83-91,1992.

パターン P=annual を誤り2以下で受理する非決定性有限オートマ



T= anneal を読み込んだときの状態

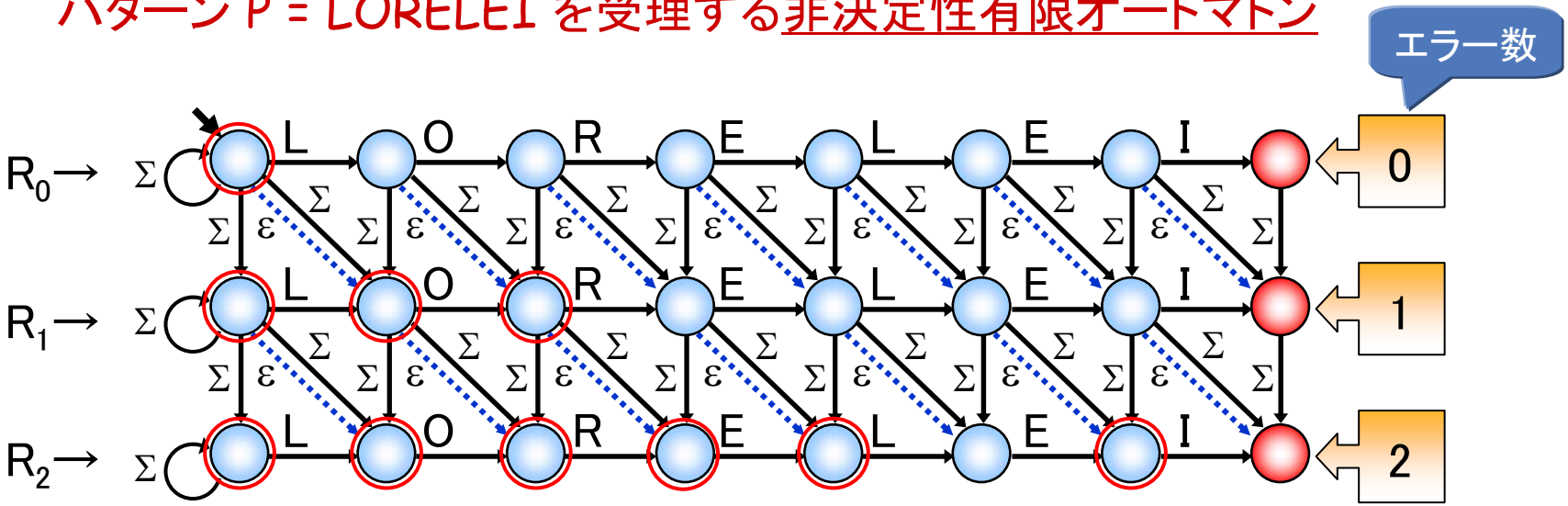
- 横一列の状態を一つのビット列で表現(activeな状態を1、非activeを0)し、ビットパラレル手法で全体の状態遷移をシミュレートする
  - ビット長mのビットマスクが k+1 個必要
  - i番目の段の状態R_i を新しい状態R'_i へ更新する式
    - $R'_0 \leftarrow ((R_0 << 1) \mid 0^{m-1}1) \,\&\, B[t_j]$
    - $R'_i \leftarrow ((R_i << 1) \,\&\, B[t_j]) \mid R_{i-1} \mid (R_{i-1} << 1) \mid (R'_{i-1} << 1)$

O(k⌈m/w⌉n)時間
m≦wの時はO(kn)時間

複数列をまとめる
ことはできない！

BPRの動作

パターン P = LORELEI を受理する非決定性有限オートマトン



状態R_0 = 0 0 0 0 0 0 0 1
状態R_1 = 0 0 0 0 0 1 1 1
状態R_2 = 0 1 0 1 1 1 1 1

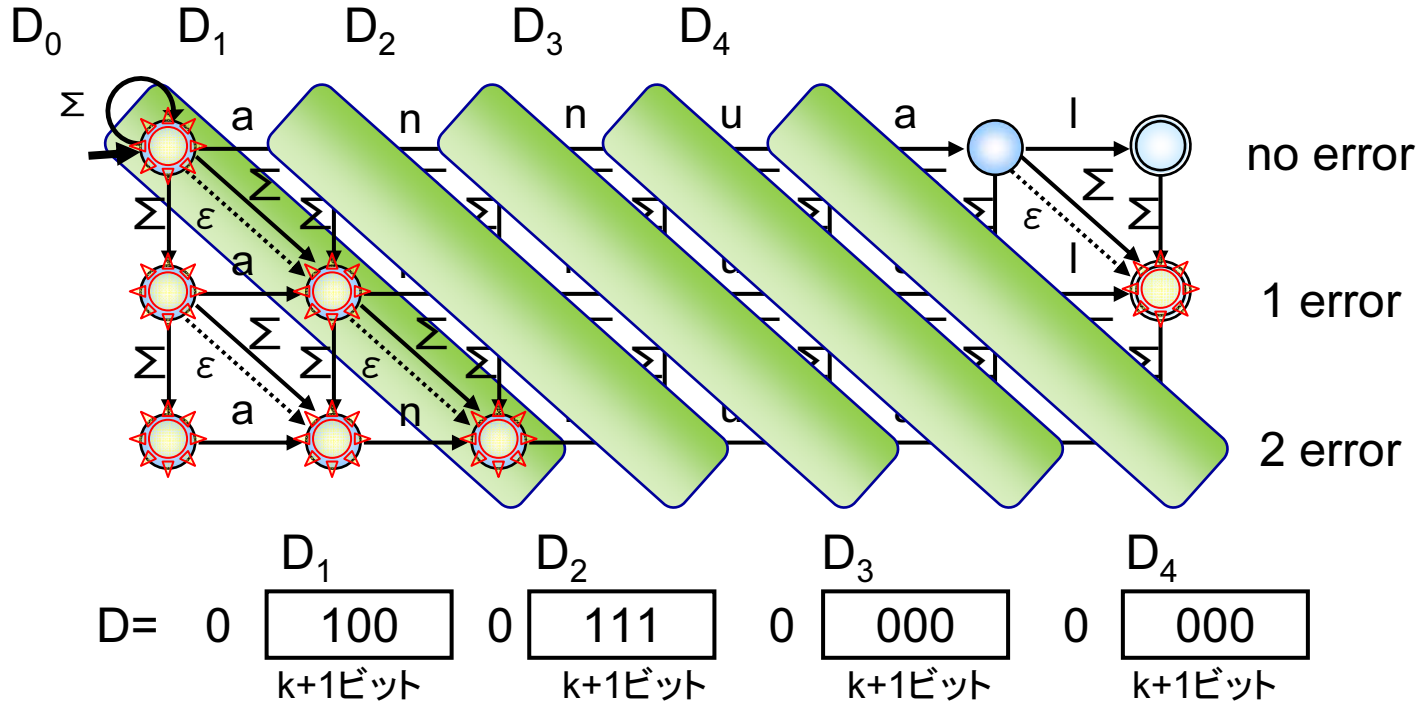
- $R'_0 \leftarrow ((R_0 << 1) \mid 0^{m-1}1) \,\&\, B[t_j]$
  - $R'_i \leftarrow ((R_i << 1) \,\&\, B[t_j]) \mid R_{i-1} \mid (R_{i-1} << 1) \mid (R'_{i-1} << 1)$

擬似コード

BPR ($P=p_1p_2\ldots p_m$, $T=t_1t_2\ldots t_n$, k)
1 Preprocessing:
2 **For** $c\in\Sigma$ **Do** $B[c] \leftarrow 0^m$
3 **For** $j \in 1\ldots m$ **Do** $B[p_j] \leftarrow B[p_j] \mid 0^{m-j}10^{j-1}$
4 Searching:
5 **For** $i \in 0\ldots k$ **Do** $Ri \leftarrow 0^{m-i}1^i$
6 **For** $pos \in 1\ldots n$ **Do**
7 $oldR \leftarrow R_0$
8 $newR \leftarrow ((oldR << 1) \mid 0^{m-1}1) \& B[t_{pos}]$
9 $R_0 \leftarrow newR$
10 **For** $i \in 1\ldots k$ **Do**
11 $newR \leftarrow ((R_i << 1) \& B[t_{pos}]) \mid oldR \mid ((oldR \mid newR) << 1)$
12 $oldR \leftarrow R_i$, $R_i \leftarrow newR$
13 **End of for**
14 **If** $newR \& 10^{m-1} \neq 0^m$ **Then** report an occurrence at pos
15 **End of for**

NFAのDiagonal-wiseビットパラレル (BPD)

R. A. Baeza-Yates and G. Navarro. Faster approximate string matching. *Algorithmica*, 23(2):127-158, 1999.



- 斜めの列(Diagonal)を基準に、各列においてactiveな状態の深さD_i をunary で保持し(k+1ビット必要)、それらを連結したもので全体を表す
  - 境界に0 をはさむため、全体では(m-k)(k+2)ビット必要
  - i番目のテキスト t_j を読み込んだときの新しいD'iの値:
    - $D'_i \leftarrow \min(D_i + 1, D_{i+1}+1, g(i-1, t_j))$
    - $g(i, c) = \min(\{k+1\} \cup \{r \mid r \geqq D_i \text{ and } p_{i+1+r} = c\})$

Shift-Orライクな
ビットマスク

擬似コード

BPD ($P=p_1p_2\ldots p_m$, $T=t_1t_2\ldots t_n$, k)

1 Preprocessing:
2 **For** $c\in\Sigma$ **Do** $B[c] \leftarrow 1^m$
3 **For** $j \in 1\ldots m$ **Do** $B[p_j] \leftarrow B[p_j] \,\&\, 1^{m-j}01^{j-1}$
4 **For** $c\in\Sigma$ **Do**
5 $BB[c] \leftarrow 0\ s_{k+1}(B[c],0)\ 0\ s_{k+1}(B[c],1)\ldots\ 0\ s_{k+1}(B[c],m-k-1)$
6 **End of for**
7 Searching:
8 $D \leftarrow (01^{k+1})^{m-k}$
9 **For** $pos \in 1\ldots n$ **Do**
10 $x \leftarrow (D >> (k+2)) \mid BB[t_{pos}]$
11 $D \leftarrow ((D << 1) \mid (0^{k+1}1)^{m-k})$
12 $\&\ ((D << (k+3)) \mid (0^{k+1}1)^{m-k-1}01^{k+1})$
13 $\&\ (((x + (0^{k+1}1)^{m-k}) \wedge x) >> 1) \,\&\, (01^{k+1})^{m-k}$
14 **If** $D \,\&\, 0^{(m-k-1)(k+2)}010^k = 0^{(m-k)(k+2)}$ **Then**
15 Report an occurrence at pos
16 $D \leftarrow D \mid 0^{(m-k-1)(k+2)}01^{k+1}$
17 **End of If**
18 **End of for**

D_i +1

D_{i+1} +1

clean up

$g(i-1, T_{pos})$

ちょっと、ひといき・・・

- ここまでのまとめ
  - 近似文字列照合とは、編集距離がk以下の部分文字列を見つける問題
  - 動的計画法によるアルゴリズム
    - O(mn)時間、O(m)領域　→　平均時O(kn)時間に改良するアルゴリズム：DP
  - オートマトンに基づくアルゴリズム
    - K以下の誤りを許してパターンを受理するNFAを構築　→　DFAに変換して計算
  - ビットパラレル手法によるNFAシミュレーション
    - Row-wise と Diagonal-wise

ジェンツーペンギン（旭山動物園にて '05.8.12）

～トリビア～

レジスタ内の最右の1のビットをオフにする。

$$x \leftarrow x \,\&\, (x - 1)$$

レジスタ内の最右の1のビットだけを分離する。

$$x \leftarrow x \,\&\, (-x)$$

出典：「ハッカーのたのしみ」

擬似コード

BPM ($P=p_1p_2\ldots p_m$, $T=t_1t_2\ldots t_n$, k)
1 Preprocessing:
2 **For** $c \in \Sigma$ **Do** $B[c] \leftarrow 0^m$
3 **For** $j \in 1\ldots m$ **Do** $B[p_j] \leftarrow B[p_j] \mid 0^{m-j}10^{j-1}$
4 $VP \leftarrow 1^m$, $VN \leftarrow 0^m$
5 err ← m
6 Searching:
7 **For** pos ∈ 1…n **Do**
8 $X \leftarrow B[t_{pos}] \mid VN$
9 $D0 \leftarrow ((VP + (X \,\&\, VP)) \wedge VP) \mid X$
10 $HN \leftarrow VP \,\&\, D0$
11 $HP \leftarrow VN \mid \sim(VP \mid D0)$
12 $X \leftarrow HP << 1$
13 $VN \leftarrow X \,\&\, D0$
14 $VP \leftarrow (HN << 1) \mid \sim(X \mid D0)$
15 **If** $HP \,\&\, 10^{m-1} \neq 0^m$ **Then** err ← err + 1
16 **Else If** $HN \,\&\, 10^{m-1} \neq 0^m$ **Then** err ← err – 1
17 **If** err ≦ k **Then** report an occurrence at pos
18 **End of for**

フィルタリング手法：パターン分割方式

S. Wu and U. Manber. Fast text searching allowing errors. Communications of the ACM, 35(10): 83-91,1992.

- フィルタリング手法の考え方：
  - 「テキストのある位置がパターンの出現位置である」ということよりも、「出現位置ではない」ことを調べるほうが簡単
  - 近似文字列照合のアルゴリズムは（比較的）複雑なので、遅い
    → 出現位置の候補を高速に見つけてから、詳細に検査する！
  - フィルタリング手法は平均時の計算量を改善する
  - 実際、エラー率（α=k/m）が小さいときはうまくいく
- パターン分割方式
  - パタンを k+1 個の小片に分割
  - 各小片を複数パターン照合アルゴリズムで高速に照合
  - 小片が出現した近辺を、通常用いられる近似文字列照合アルゴリズムを用いて検査する

Multiple Shift-And
や
Set Horspool

$k = 2$ の場合

パターン: TAAATCACGGCATACT

分割後の小片: TAAAT, CACGG, CATACT

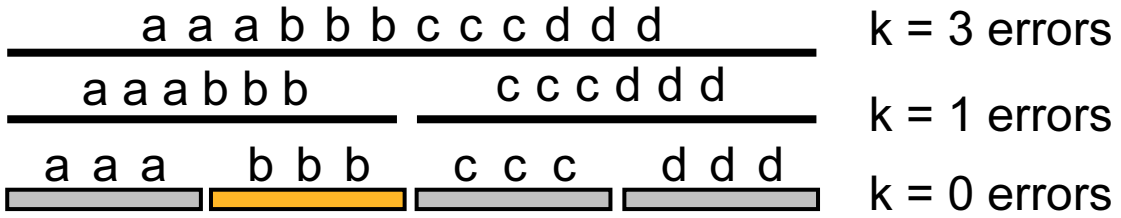
テキスト: ACCCTGTTTAGATCACGGCACTACTGTAAAC

階層的検証による高速化

G. Navarro and R. Baeza-Yates. Very fast and simple approximate string matching. Information Processing Letters, 72:65-70, 1999.

- 検査を階層的に行うことで、正しくない候補を少ない労力で見分けられる
  - $j = k+1 = 2^r$ と仮定する
  - パターンを半分に分割、それぞれ$\lfloor k/2 \rfloor$個の誤りを許すパターンとする。これを再帰的に誤りが0になるまで繰り返す。
  - 分割したパターンを複数パターン照合アルゴリズムで照合
  - 出現の候補を階層的に検査する

2r でない場合、
バランスする木をつくる



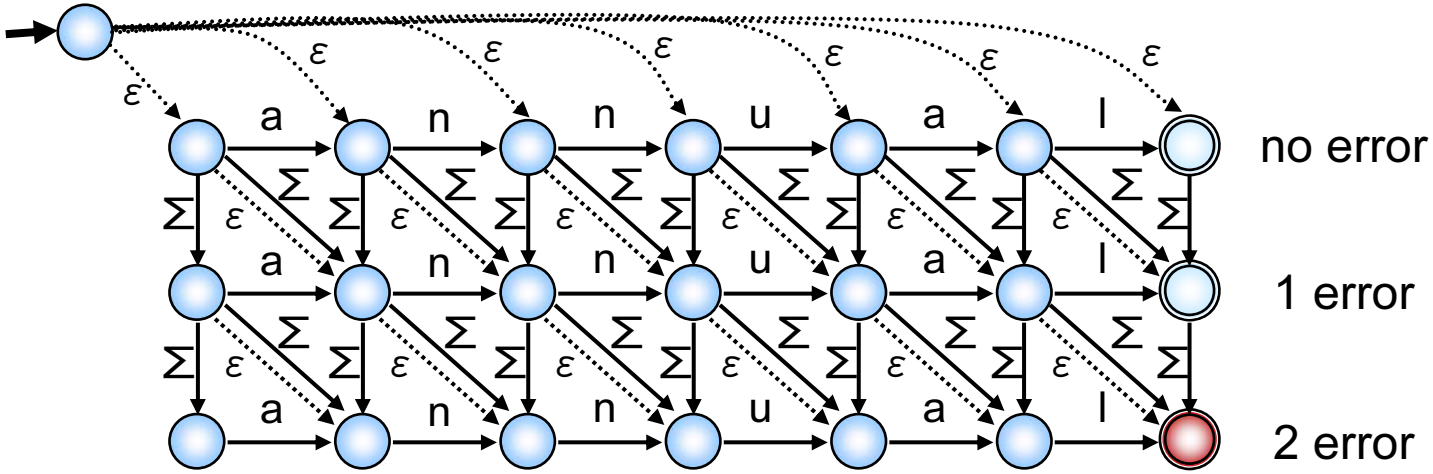
CreateTree ($P=p_1p_2\ldots p_m$, k, myParent, idx, plen)
1 Create new node
2 from(node) ← i
3 to(node) ← j
4 left ← $\lceil (k+1)/2 \rceil$
5 parent(node) ← myParent
6 err(node) ← k
7 **If** k = 0 Then leafidx ← node
8 **Else**
9 CreateTree($p_{i\ldots i+left\cdot plen-1}$, (left・k)/(k+1), node, idx, plen)
10 CreateTree($p_{i+left\cdot plen\ldots j}$, $\lfloor ((k+1-left)\cdot k)/(k+1) \rfloor$, node, idx + left, plen)
11 **End of If**

擬似コード

PEX ($P=p_1p_2\ldots p_m$, $T=t_1t_2\ldots t_n$, k)
1 Preprocessing:
2 **CreateTree**(p, k, θ , 0, $\lfloor m/(k+1) \rfloor$)
3 Preprocess multipattern search for
4 $\{p_{from(node)}\ldots p_{to(node)} \mid node = leaf_i , i \in \{0\ldots k\} \}$
5 Searching:
6 **For** (pos, i) $\in$ output of multipattern search **Do**
7 node $\leftarrow$ $leaf_i$
8 in $\leftarrow$ from(node)
9 node $\leftarrow$ parent(node)
10 cand $\leftarrow$ TRUE
11 **While** cand = TRUE and node $\neq \theta$ **Do**
12 $p_1 \leftarrow$ pos – (in – from(node)) – err(node)
13 $p_2 \leftarrow$ pos + (to(node) – in + 1) + err(node)
14 Verify text area $T_{p1\ldots p2}$ for pattern piece $p_{from(node)\ldots to(node)}$
15 allowing err(node) errors
16 **If** pattern piece was not found **Then** cand $\leftarrow$ FALSE
17 **Else** node $\leftarrow$ parent(node)
18 **End of while**
19 **If** cand = TRUE **Then**
20 Report the positions where the whole p was found
21 **End of If**
22 **End of for**

フィルタリング手法：BNDM方式

G. Navarro and R. Baeza-Yates. Very fast and simple approximate string matching. Information Processing Letters, 72:65-70, 1999.



- パターンPに対して、P^Rの任意のfactorにk以下の誤りを許した文字列を受理するNFAを構築 → BNDMの拡張
  - このNFAは、P^Rの誤りk以下のprefixを認識できる
  - BNDMはアルファベットサイズが小さいとき、Boyer-Mooreよりも高速
    - このNFAで出現の候補を特定する
    - BNDMと同じく、テキストを飛ばし読みしながら動作する
  - DNAのようなテキストに対しては、PEXよりもうまく動作する

擬似コード

```
ABNDM (P=p1p2…pm, T=t1t2…tn, k)
1    Preprocessing:
2        For c∈Σ Do B[c] ← 0^m
3        For j ∈1…m Do B[pj] ← B[pj] | 0^(m-j) 10^(j-1)
4    Searching:
5        pos ← 0
6        While pos ≦ n – (m – k) Do
7            j ← m – k – 1, last ← m – k – 1
8            R0 ← B[t(pos+m–k)]
9            newR ← 1^m
10           For i ∈1…k Do Ri ← newR
11           While newR ≠ 0^m and j ≠ 0 Do
12               oldR ← R0
13               newR ← (oldR << 1) & B[t(pos+j)]
14               R0 ← newR
15               For i ∈1…k Do
16                   newR ← ((Ri << 1) & B[t(pos+j)]) | oldR | ((oldR | newR) << 1)
17                   oldR ← Ri, Ri ← newR
18               End of for
19               j ← j – 1
20               If newR & 10^(m-1) ≠ 0^m Then /* prefix recognized */
21                   If j > 0 Then last ← j
22                   Else check a possible occurrence starting at pos + 1
23               End of if
24           End of while
25           pos ← pos + last
26       End of while
```

第4回 まとめ

- 近似文字列照合とは？
  - 編集距離がk以下の部分文字列を見つける問題
- 動的計画法によるアルゴリズム
  - O(mn)時間、O(m)領域 → 平均時O(kn)時間に改良するアルゴリズム(DP)
- オートマトンに基づくアルゴリズム
  - K以下の誤りを許してパターンを受理するNFAを構築 → DFAに変換して計算
- ビットパラレル手法
  - Row-wiseビットパラレル(BPR): $O(k\lceil m/w \rceil n)$時間
  - Diagonal-wiseビットパラレル(BPD): $O(\lceil k(m-k)/w \rceil n)$時間
  - DP表をビットパラレル(BPM): $O(\lceil m/w \rceil n)$時間
- フィルタリング手法
  - テキストの大部分を調べずに済ませる
  - パターン分割方式(PEX)、BNDM方式(ABNDM)
- 次回のテーマ
  - 正規表現の照合: 柔軟・便利な検索キーワードの記述ができる

DP表のビットパラレル

各ビットマスクの更新手順

各ビットマスクの更新例