データ構造とアルゴリズム

第11回　文字列

小池 英樹 (koike@c.titech.ac.jp)

文字列照合

- 長さnのテキストTから，長さmの文字列Pを見つける
  - 一般に，n >> m

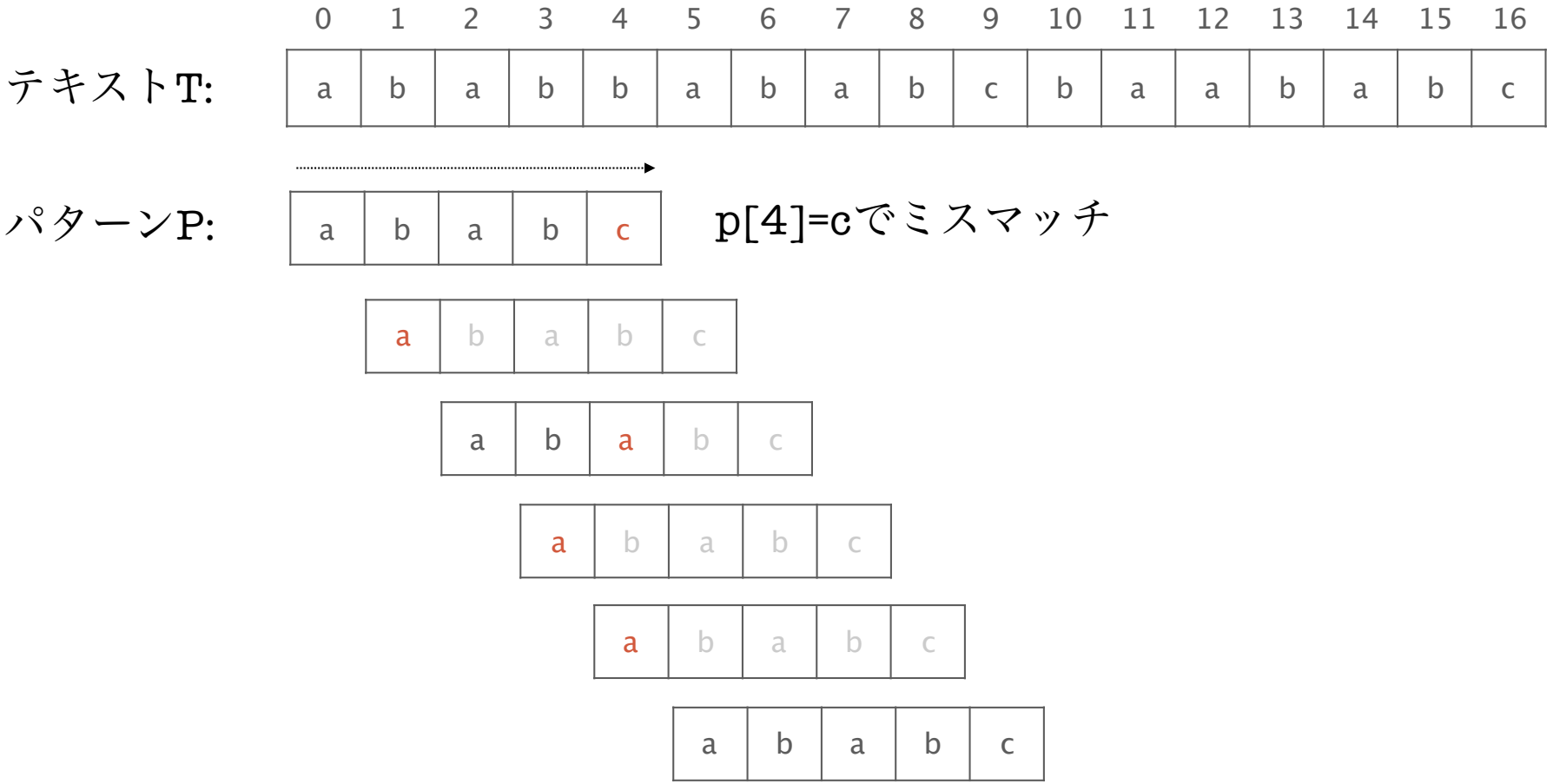
- 応用例：
  - テキストエディタでの文字列検索
  - メールのSPAM判定
  - DNAからのパターン検索
  - etc.

文字列照合：代表的アルゴリズム

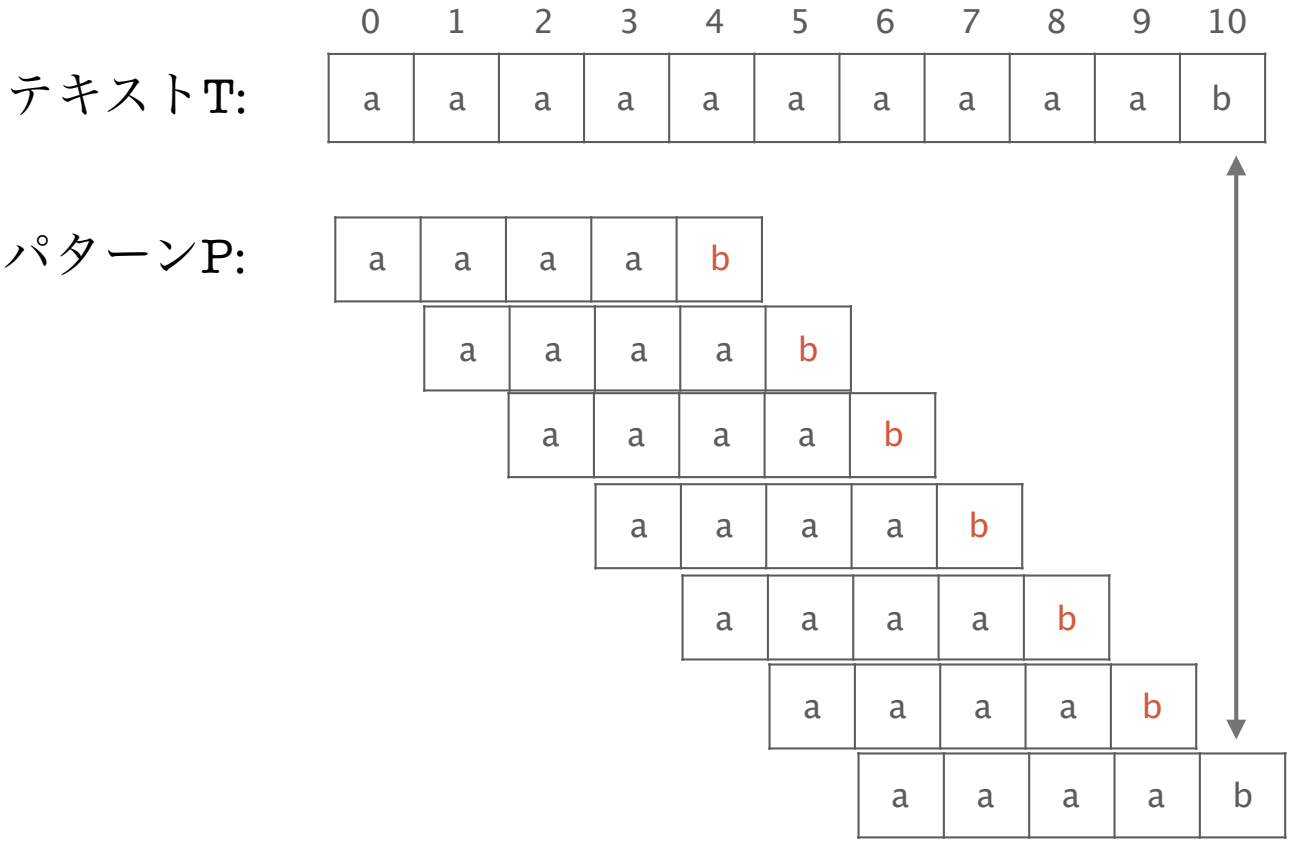
- Brute-forceアルゴリズム
- Knuth-Morris-Prattのアルゴリズム (KMP法)
- Boyer-Mooreのアルゴリズム (BM法)

BRUTE-FORCEアルゴリズム

先頭から順に調べ，一致しなければ先頭を 1 つずつずらしていく



BRUTE-FORCEアルゴリズム（最悪のケース）



最悪 O(m × n)

BRUTE-FORCEアルゴリズム

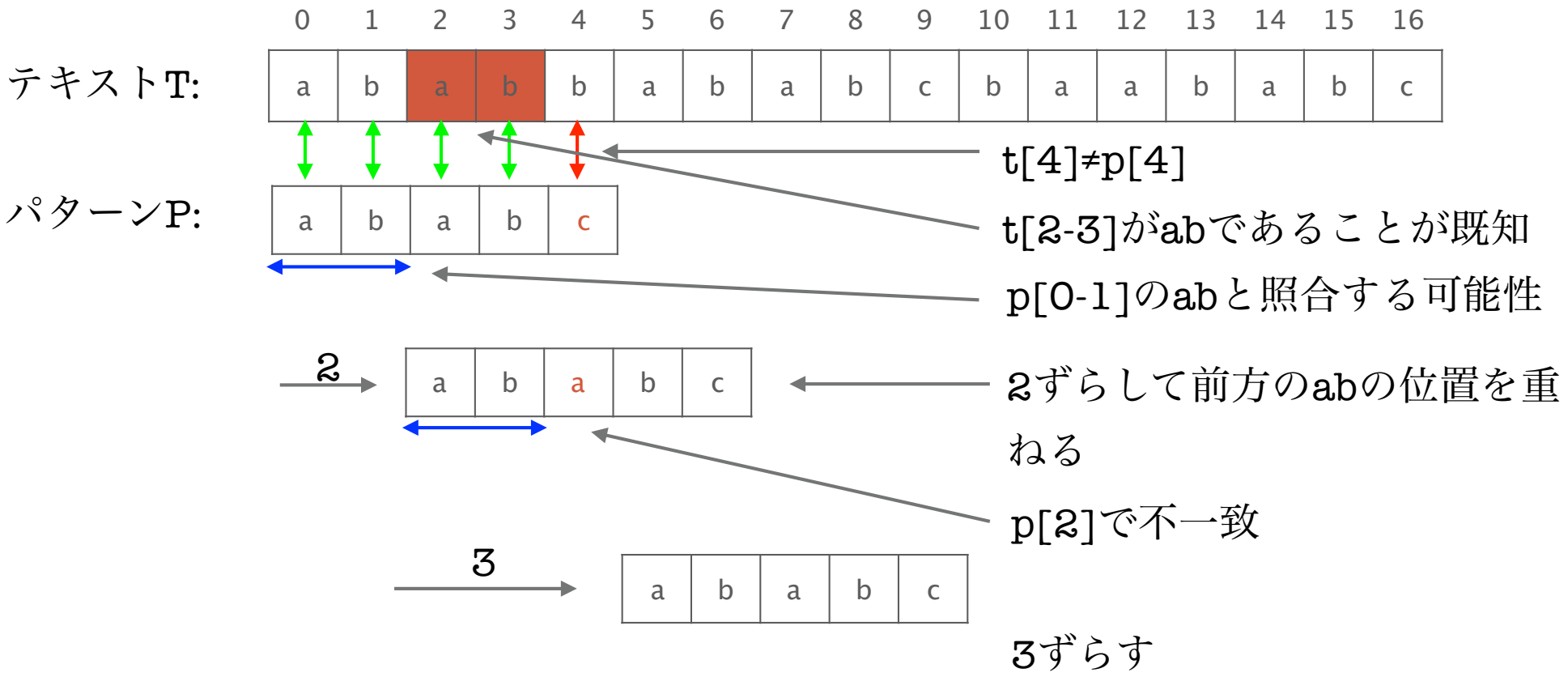
t: テキスト， p: 照合文字列, n: tの長さ, m: pの長さ

```
int bf(char *t, char *p, int n, int m) {
    int i, j;

    for (j=0; j <= n-m; j++) {
        for (i=0; i < m && p[i] == t[i+j]; i++)
            ;
        if (i >= m)
            return(j);
    }
    return(-1);
}
```

KNUTH-MORRIS-PRATTのアルゴリズム

brute-forceと同様，パターンの先頭から照合していくが，それまでに調べた情報（履歴を使用する）



KMP: 比較表の作成

配列kmpNext[]が，検索文字列pの何文字目かで不一致になったときに，何文字目から比較開始するかを保持する．

kmpNext[i] = -1だったら，テキスト側のポインタを 1 つ進めて，パターン側のポインタを0にすることを表す．

```
void preKmp(char *p, int m, int kmpNext[]) {
    int i, j;

    i = 0;
    j = kmpNext[0] = -1;
    while (i < m) {
        while (j > -1 && p[i] != p[j])
            j = kmpNext[j];
        i++;
        j++;
        if (p[i] == p[j])
            kmpNext[i] = kmpNext[j];
        else
            kmpNext[i] = j;
    }
}
```

KMP: 比較テーブルの例

- P=“ababc” -> kmpNext[] = {-1, 0, -1, 0, 2}
- P=“aaab” -> kmpNext[] = {-1, -1, -1, 2}

```
void preKmp(char *p, int m, int kmpNext[]) {
    int i, j;

    i = 0;
    j = kmpNext[0] = -1;
    while (i < m) {
        while (j > -1 && p[i] != p[j])
            j = kmpNext[j];
        i++;
        j++;
        if (p[i] == p[j])
            kmpNext[i] = kmpNext[j];
        else
            kmpNext[i] = j;
    }
}
```

KMP

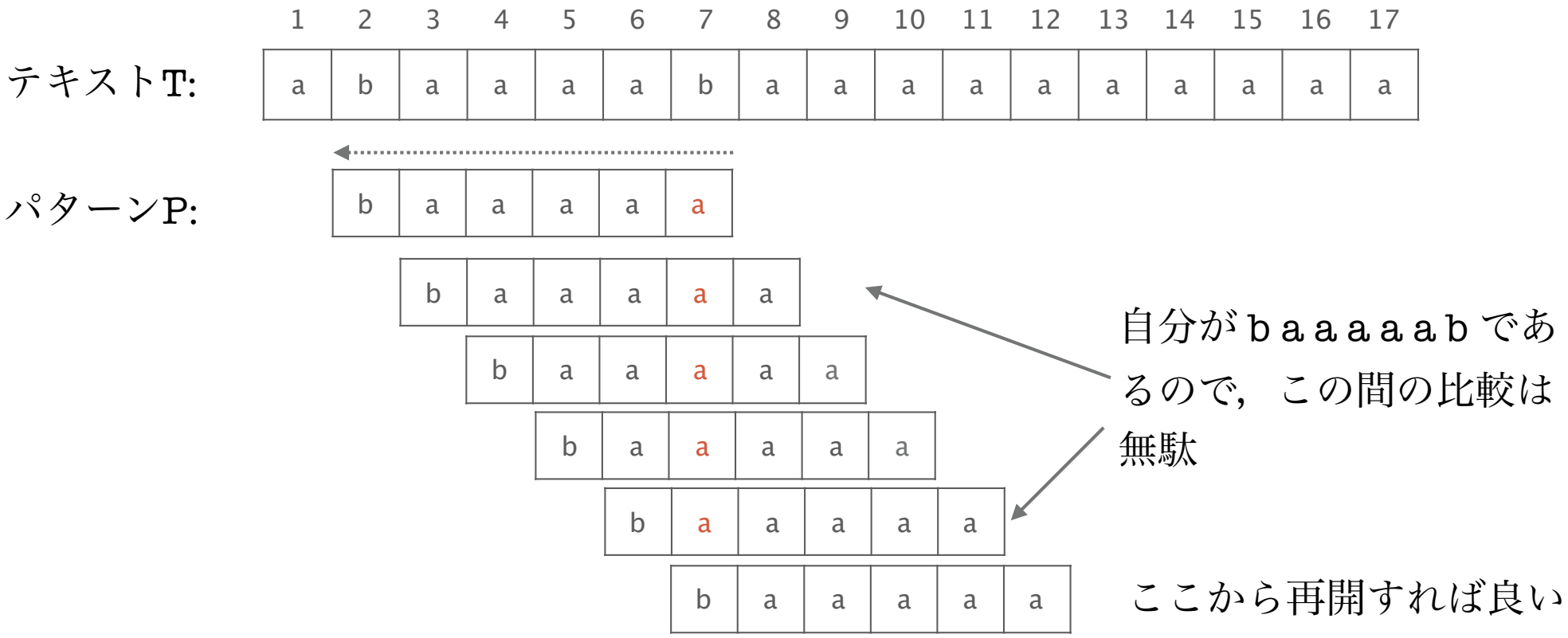
```
int kmp(char *t, char *p, int n, int m) {
    int i, j, kmpNext[ASIZE];       /* ASIZEはalpabetの文字数 */

    preKmp(p, m, kmpNext);

    i = j = 0;
    while (j < n) {
        while (i > -1 && p[i] != t[j])
            i = kmpNext[i];
        i++;
        j++;
        if (i >= m) {
            printf("%d \n", j-i);  /* 発見 */
            i = kmpNext[i];
        }
    }
    return(-1);
}
```

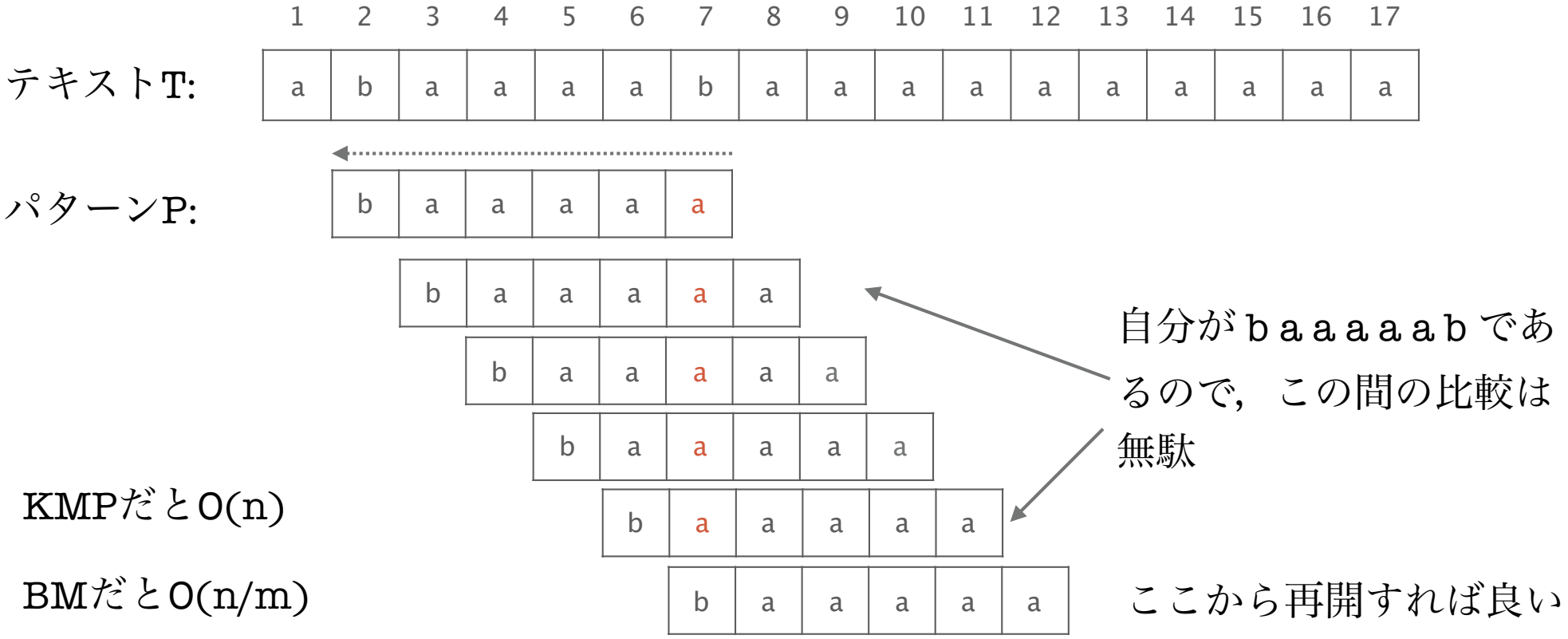
BOYER-MOOREのアルゴリズム

- 非常に効率的でかつ実用的 -> 各種エディタに実装されている
- 照合をパターンの後ろから開始



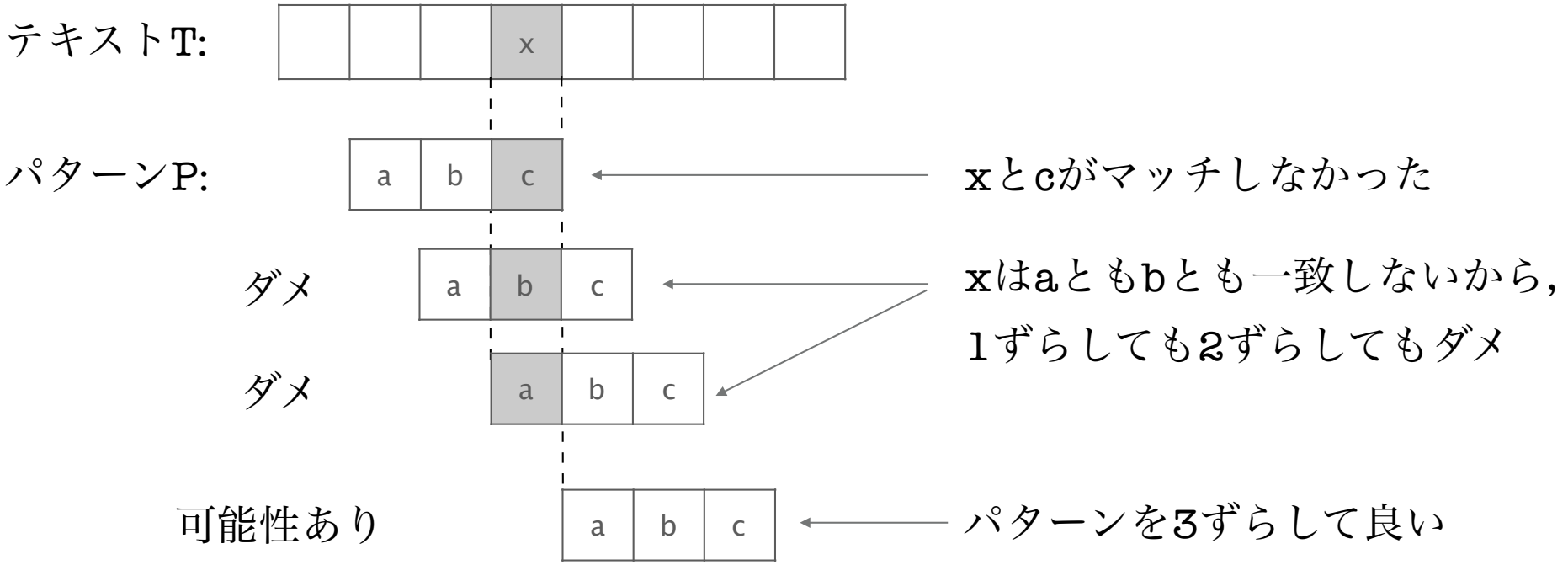
BOYER-MOOREのアルゴリズム

- 2つのヒューリスティックスを使用する
  - bad-character shift: 1文字の照合失敗に着目
  - good-suffix shift: 数文字の照合に着目



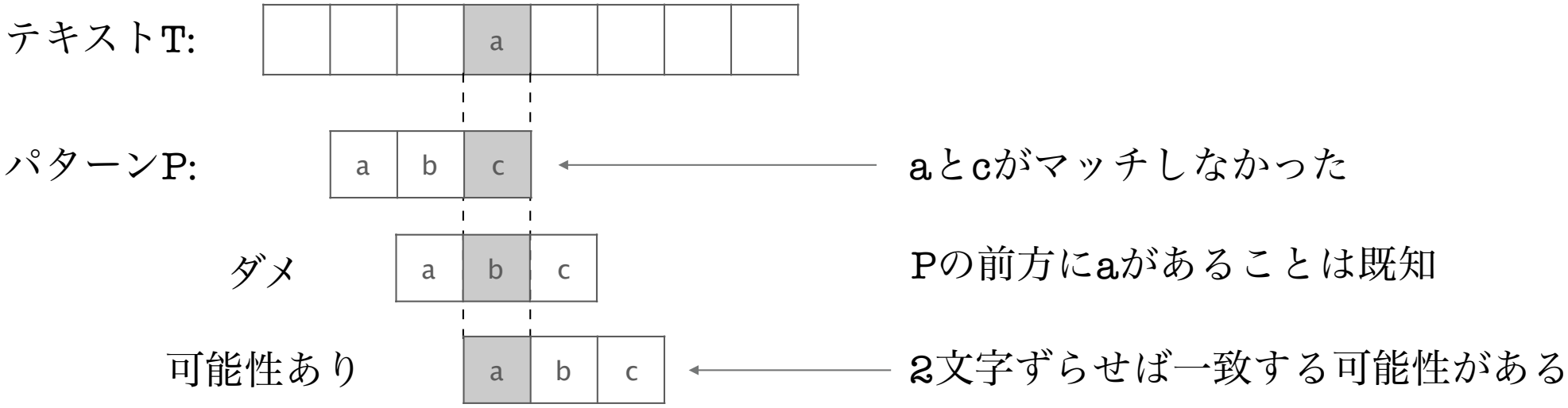
BAD-CHARACTER SHIFT

(1) 一致しなかった場所のTの文字がPに含まれていない場合



BAD-CHARACTER SHIFT

(2) 一致しなかった場所のTの文字がPに含まれている場合

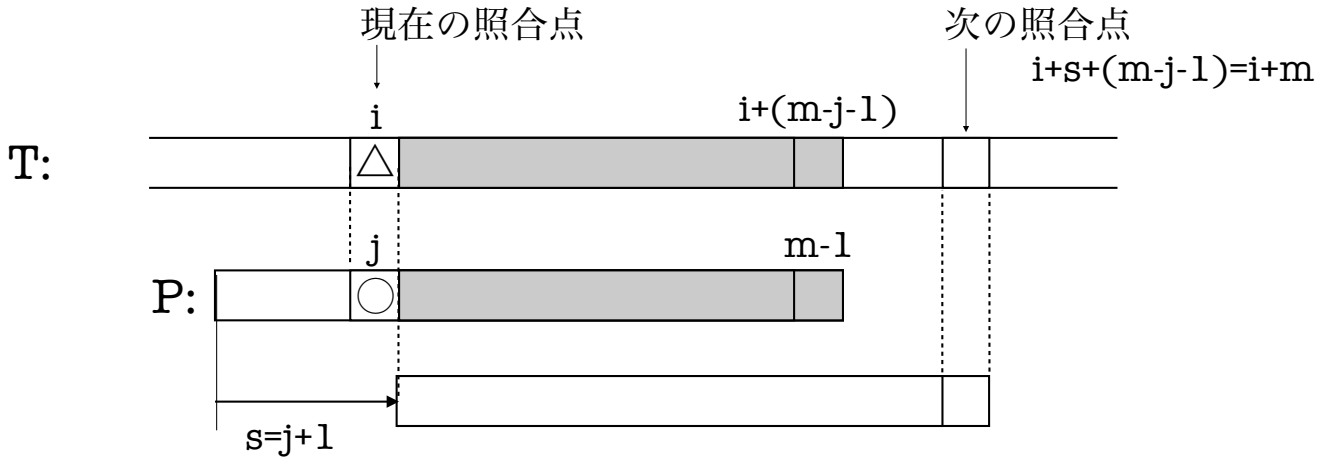


・つまり，不一致が判明した点のT側の文字がPの中にあるかどうか，あればどの位置にあるかで何文字ずらすかが決まる．

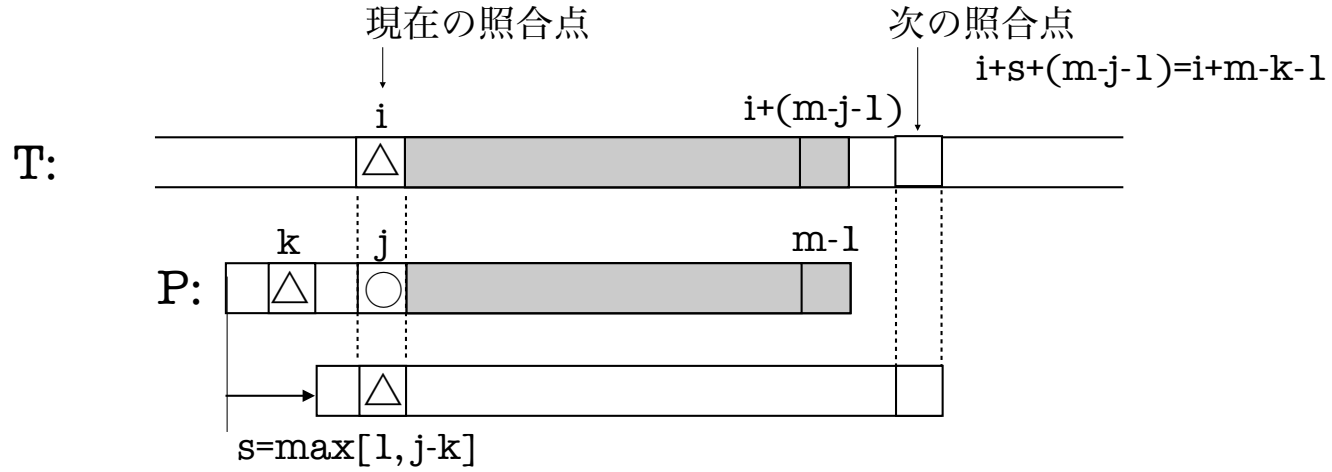
・Tに現れうる全ての文字についてあらかじめ計算して表にしておくことができる．

BAD-CHARACTER SHIFT

(1) 文字△がPにない場合



(2) 文字△がPにある場合



BAD-CHARACTER SHIFT: 表の作成

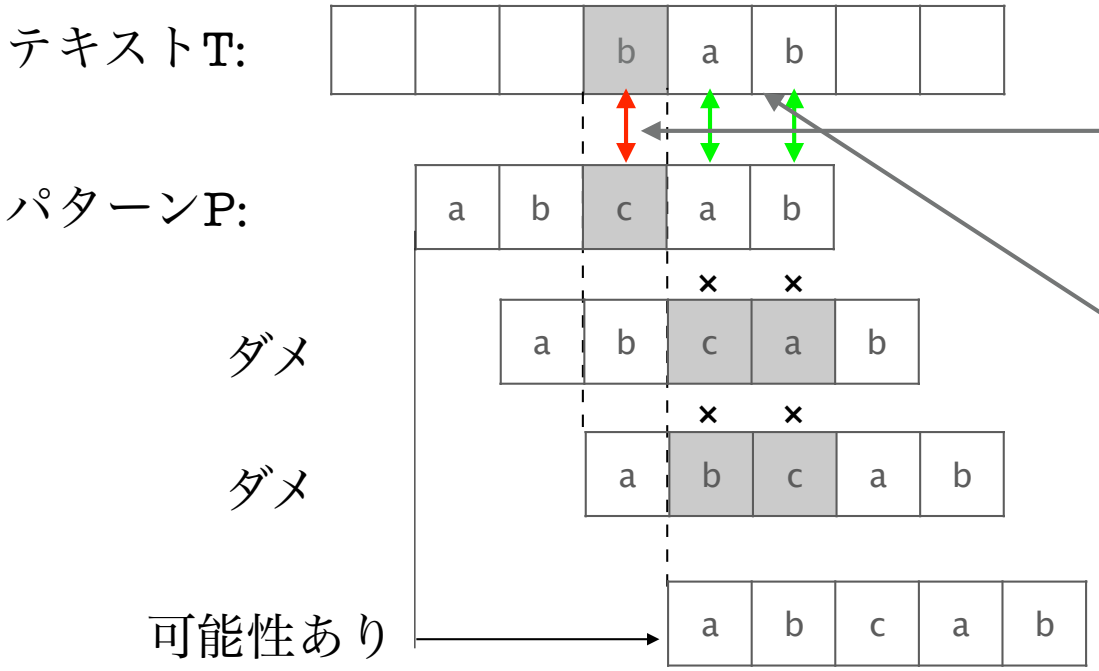
- シフトする文字数
  - Pに含まれる文字(末尾を除く)：末尾からの順番
  - Pに含まれない文字：Pの長さ
- 例：”never” -> n->4, e->1, v->2, r->5

```
void preBmBc(char *p, int m, int bmBc[]) {
    int i;

    for (i=0; i < ASIZE; i++)
        bmBc[i] = m;
    for (i=0; i < m-1; i++)
        bmBc[p[i]] = m-i-1;
}
```

GOOD-SUFFIX SHIFT

Pの後方が前方にもある場合



bとcが照合失敗

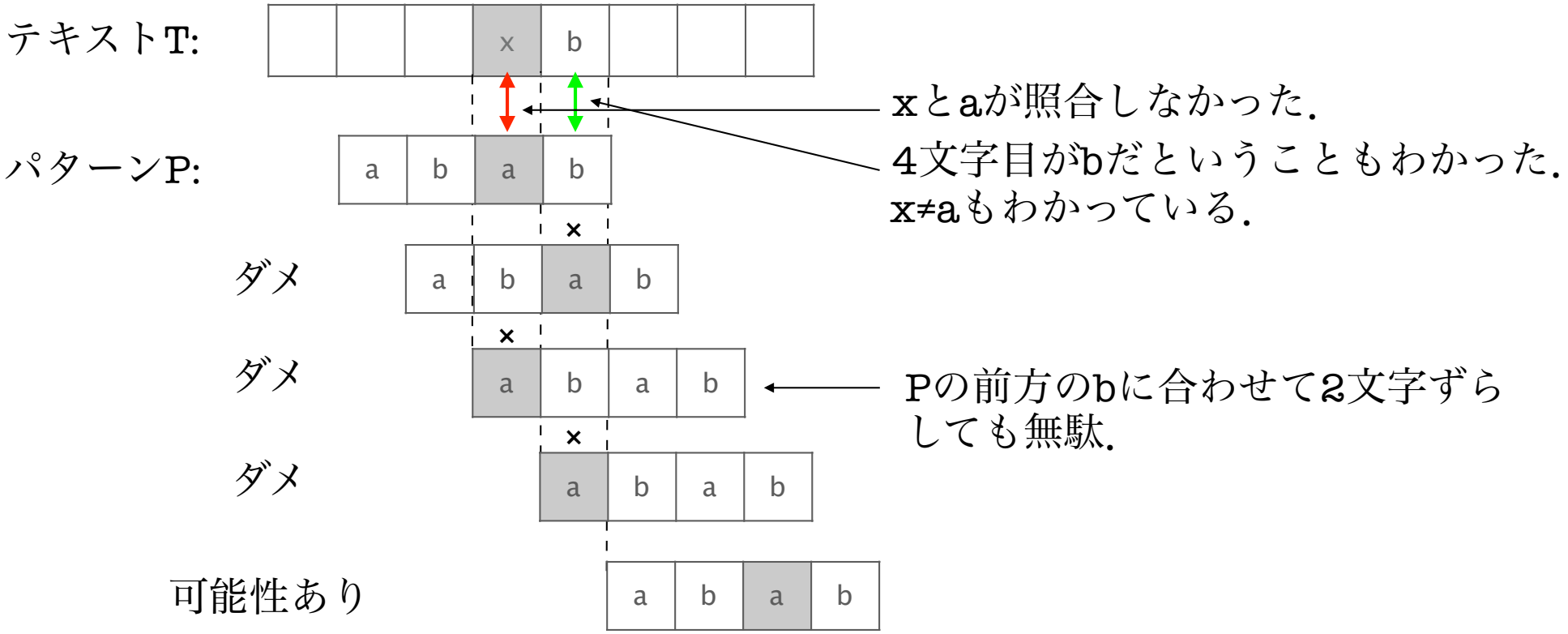
bad-char shiftだけなら1ずらすだけ

Tのこの部分がa, bであることは既にわかっている.

パターン前方にa, bがあることもわかっている.

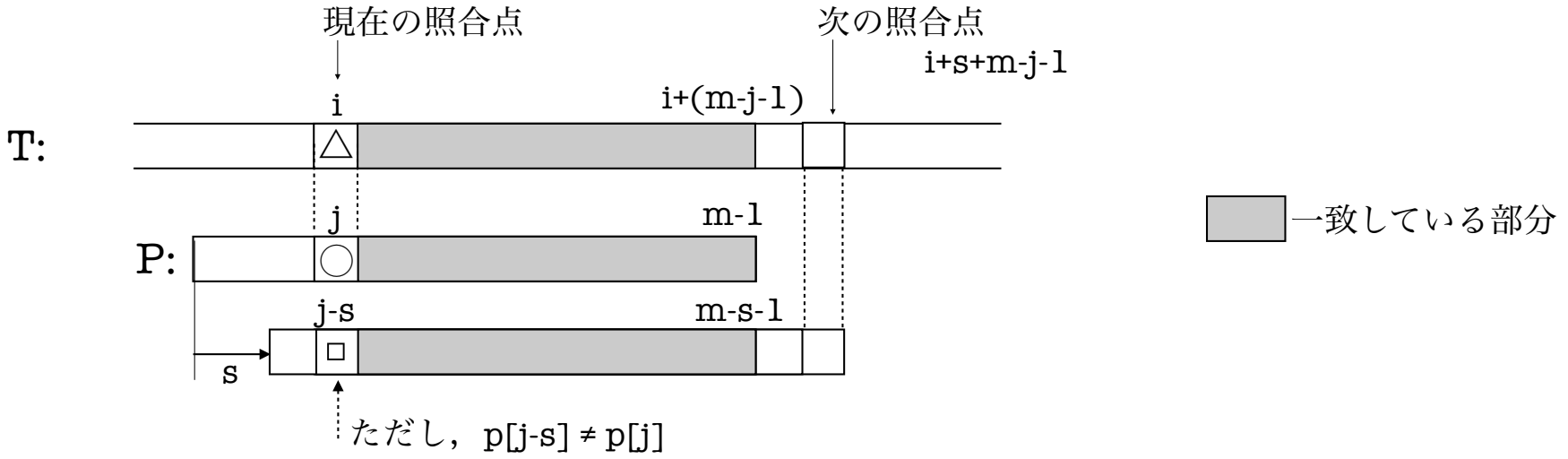
GOOD-SUFFIX SHIFT

不一致の情報も使う

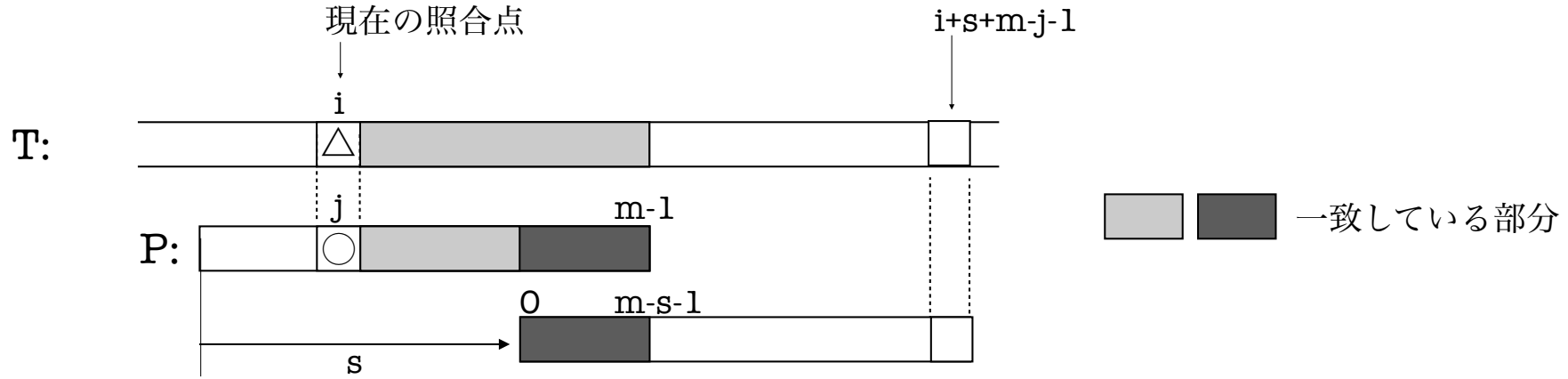


GOOD-SUFFIX SHIFT

(1) Pの後方が前方にもある場合



(2) Pの後方部分がPの先頭部分と一致する場合



GOOD-SUFFIX SHIFT

```
void suffixes(char *p, int m, int *suff) {
    int f, g, i;

    suff[m - 1] = m;
    g = m - 1;
    for (i = m-2; i >= 0; i--) {
        if (i > g && suff[i+m-1-f] < i-g)
            suff[i] = suff[i+m-1-f];
        else {
            if (i < g)
                g = i;
            f = i;
            while (g >= 0 && p[g] == p[g+m-1-f])
                --g;
            suff[i] = f-g;
        }
    }
}
```

GOOD-SUFFIX SHIFT

```
void preBmGs(char *p, int m, int bmGs[]) {
    int i, j, suff[ASIZE];

    suffixes(p, m, suff);

    for (i = 0; i < m; i++)
        bmGs[i] = m;
    j = 0;
    for (i = m-1; i >= 0; i--)
        if (suff[i] == i + 1)
            for (; j < m-1-i; j++)
                if (bmGs[j] == m)
                    bmGs[j] = m-1-i;
    for (i = 0; i <= m-2; i++)
        bmGs[m-1-suff[i]] = m-1-i;
}
```

BOYER-MOORE

```
int bm(char *t, char *p, int n, int m) {
    int i, j, bmGs[ASIZE], bmBc[ASIZE];

    preBmGs(p, m, bmGs);
    preBmBc(p, m, bmBc);

    j = 0;
    while (j <= n-m) {
        for (i = m-1; i >= 0 && p[i] == t[i+j]; i--);
        if (i < 0) {
            printf("%d ", j);    /* 照合した位置jの出力 */
            j += bmGs[0];
        }
        else
            j += max(bmGs[i], bmBc[t[i+j]]-m+1+i);
    }
    return(-1);
}
```

計算量

- BF: 最悪O(n×m)
- KMP: O(n)
- BM: O(n)，ただし最善でO(n/m)

- 実際には：
  - BFのO(n×m)は同じパターンの続く最悪の場合.
  - 現実的にはBFも十分実用的.

期末試験

- 期末試験
  - 日時：2/6 13:20-14:50
  - 場所：W621
  - 出題範囲：前半(hashまで)が30%, 後半が70%
    - プリント等，持ち込み不可
    - 用語やアルゴリズムを中心に問う（動作とか）