アルゴリズムとデータ構造III
12回目：1月6日（木）

全文検索アルゴリズム
（Aho-Corasick）
暗号：符号化：テキスト圧縮

授業資料　http://ir.cs.yamanashi.ac.jp/~ysuzuki/public/algorithm3/index.html

授業の予定（中間試験まで）

1	10/07	スタック（後置記法で書かれた式の計算）
2	10/14	チューリング機械，文脈自由文法
3	10/21	構文解析　CYK法
4	11/04	構文解析　CYK法
5	11/11	構文解析（チャート法）
6	11/18	構文解析（チャート法），グラフ（ダイクストラ法，DPマッチング）
7	11/19 4時限 B2-41	グラフ（A*アルゴリズム，DPマッチング）
8	11/25	グラフ（ DPマッチング），前半のまとめ
9	12/02	中間試験

授業の予定（中間試験以降）

10	12/09	全文検索アルゴリズム（simple search, KMP)
11	12/16	全文検索アルゴリズム（BM, Aho-Corasick)
12	01/06	全文検索アルゴリズム（Aho-Corasick），データ圧縮
13	01/13	暗号（黄金虫，踊る人形） 符号化（モールス信号，Zipfの法則，ハフマン符号）テキスト圧縮
14	01/20	テキスト圧縮　（zip）， 音声圧縮　（ADPCM，MP3，CELP）， 画像圧縮（JPEG）
15	02/03	**期末試験**

本日のメニュー

- 全文検索アルゴリズム
  - Aho-Corasickの続き
- 暗号
  - 黄金虫（The gold bug)
  - 踊る人形（The Adventure of the Dancing Men)
- 符号化
- テキスト圧縮

全文検索

- 文書中から，与えられた文字列と完全に一致する部分を探し出す．
- 全文検索の種類
  - **文字列照合による全文検索**
  - 索引を用いた全文検索

文字列照合タスク

- テキスト処理には不可欠
- テキスト文字列からキーワードとその出現位置を見つける
- 例
  - テキスト文字列：aabcdabdabbabcdabacade
  - キーワード：abcaba

a	b	c	a	b	c	a	b	a	b	c	a	b	a	b	x	a	b	c	a
			a	b	c	a	b	a											
								a	b	c	a	b	a						

文字列照合アルゴリズム

- Simple Search
- Knuth-Morris-Pratt法
- Boyer-Moore法
- Aho-Corasick法

文字列照合問題の単純な解決法 Simple Search

- Simple Searchの文字列照合手順
- Simple Searchのアルゴリズム
- Simple Searchの評価

単純な文字列照合アルゴリズム Simple Search

- テキストの1文字目からn文字目まで，2文字目からn+1文字目まで，・・・がキーワードと一致するかどうかをチェックする．

a	b	c	a	b	c	a	b	a	b	c	a	b	a	b	x	a	b	c	a
a	b	c	a	b	a														
	a	b	c	a	b	a													
		a	b	c	a	b	a												
			a	b	c	a	b	a											
				a	b	c	a	b	a										

Simple Search 同じ部分を何度も照合しなければならない

位置	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2
text	a	b	c	a	b	c	a	b	a	b	c	a	b	a	b	x	a	b	c	a	b	x
	a	b	c	a	b	a																
		a																				
			a																			
				a	b	c	a	b	a													
					a																	
						a												照合失敗				
							a	b	c													
								a											文字列照合成功			
									a	b	c	a	b	a								
										a												
											a											
												a	b	c								
照合回数	1	2	2	2	3	3	2	3	3	2	2	2	2	2								

Simple Searchのアルゴリズム

- 入力:テキスト text, キーワード key
- 出力:テキスト中のキーワードの位置
- m: テキストの長さ
- n: キーワードの長さ

```
Method
begin
        for i:=1 to m-n+1 do                    起点を決めて
        begin
                for j:=1 to n do                キーワードと1字ずつ照合
                        if text[i+j-1]≠key[j] then
                                goto 1;
                print i;
                1:
        end
end
```

Simple Search 最も効率の悪い場合

文字照合回数 (7-3+1)*3=15
(m-n+1)*n回
一般にm≫nなので O(mn)

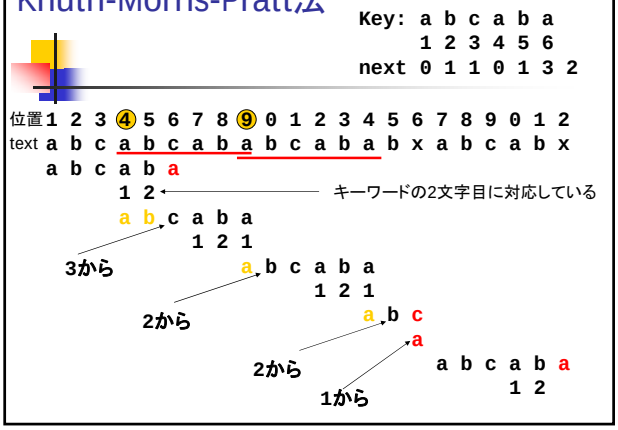
- key = aaa
- text = aaaaaaa

位置	1	2	3	4	5	6	7
text	a	a	a	a	a	a	a
	a	a	a				
		a	a	a			
			a	a	a		
				a	a	a	
					a	a	a
照合回数	1	2	3	3	3	2	1

Knuth-Morris-Pratt法 （KMP法）

- Simple Search
  - テキスト中の各文字がキーワードと複数回照合される → 冗長
- KMP法
  - 文字照合の実行中に次回の文字照合を考慮しつつ処理を進める
  - 文字照合中，バックトラックが必要ない

Knuth-Morris-Pratt法



KMP法 アルゴリズム

```
Method kmp
begin
     j:=1;
     for i:=1 to m do
     begin
          while j>0 and key[j] ≠text[i] do
               j:=next(j);
          if j=n then
               print i-n+1:
          j:=j+1;
     end
end
```

m :textの長さ
n :keywordの長さ
i: textの照合位置
J: keywordの照合位置

照合
つぎの照合位置
照合成功

キーワードの接頭辞文字列の出現位置

関数next：次回の照合でキーワードの何文字目を照合すべきか

テキスト中の照合に失敗した文字の直前の何文字がキーワードの接頭辞になっているかを調べる

位置	1	2	3	4	5	6	7				
キーワード	a	b	c	a	b	a					
				a	b	c	a	b	a		

6文字目で照合失敗した場合：直前文字列がabなので3文字目から照合開始

						a	b	c	a	b	a

照合に成功した場合：直前文字がaなので2文字目から照合開始

next関数値	0	1	1	0	1	3	2				

next関数

Keyword: abcabaのとき
123456

a:1 : keywordの一文字目のa
a : a以外の文字

1文字目のaで照合失敗 （直前の文字がa）
→ 照合失敗箇所の右隣とa:1を照合
→ 照合失敗箇所はキーワードの0文字目と照合→ next(1)=0

2文字目のbで照合失敗 （直前の文字がab）
→ 照合失敗箇所とa:1を照合 → next(2)=1

3文字目のcで照合失敗 （直前の文字がabc）
→ 照合失敗箇所とa:1を照合 → next(3)=1

next関数

Keyword: abcabaのとき
123456

a:1 : keywordの一文字目のa
a : a以外の文字

4文字目のaで照合失敗 （直前の文字がabca）
→ 照合失敗箇所の右隣とa:1を照合
→ 照合失敗箇所はキーワードの0文字目と照合→ next(4)=0

5文字目のbで照合失敗 （直前の文字がabcab）
→ 照合失敗箇所とa:1を照合 → next(5)=1

6文字目のaで照合失敗 （直前の文字がabcaba）
→ 照合失敗箇所とc:3を照合 → next(6)=3

6文字目のaで照合成功 （直前の文字がabcaba）
→ 照合失敗箇所（照合成功末尾の右隣）とb:2を照合 → next(7)=2

KMP法　アルゴリズム next関数
入力：キーワード key, 出力：next関数

```
Method next                    n : keyの長さ
begin                          j : keyの照合位置
                               t : keyのj文字目の直前の何文字がkeyの接頭辞になっているか
      t:=0;
      next(1):=0;
      for j:=1 to n do     keyの各文字に対してnext関数値を計算
      begin
            while t ≠ 0 and key[j] ≠ key[t] do
                  t:=next(t);   keyのj文字目までの文字列がkeyの
            t:=t+1;             接頭辞と一致しているか調べる
            if key[j+1]=key[t] then     keyの
                  next(j+1):=next(t);   j+1文字目の
            else                        next関数値を
                  next(j+1):=t;         決定
      end
end
```

KMP法の評価

- KMP法
  - 漸近的時間計算量　$O(m)$
  - next関数が必要　テキスト文字列の各文字に対して1回照合
- Simple Search法
  - 漸近的時間計算量　$O(mn)$

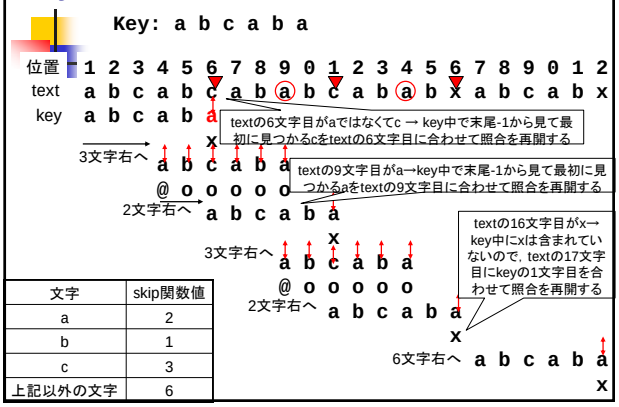
テキスト文字列の各文字に対して
キーワード文字数回照合

m: テキストの文字数
n: キーワードの文字数

Boyer-Moore法

- キーワードの末尾から照合を行う.
- キーワードの末尾と照合したテキストストリングの文字を覚えておく
- その文字とキーワードの文字が一致するまでキーワードをずらす

Boyer-Moore法



文字	skip関数値
a	2
b	1
c	3
上記以外の文字	6

skip関数

- テキスト文字列中の照合文字cが，キーワードの末尾から何文字目にあるか

```
?????a
  abcaba
 6543210

?????b
 abcaba
6543210

?????c
   abcaba
  6543210

?????x
      abcaba
     6543210
```

キーワード"a b c a b a"に対するskip関数

文字	skip関数値
a	2
b	1
c	3
上記以外の文字	6

BM法による文字列照合

m :textの長さ
n :keywordの長さ
J: keywordの照合位置
pos: text中の照合位置

```
Method BM
begin
      pos:=n;
      while pos<=m do
      begin
            if text[pos]=key[n] then
            begin
                  k:=pos-1;
                  j:=n-1;
                  while j>0 and text[k]=key[j] do
                  begin
                        k:=k-1;
                        j:=j-1;
                  end
                  if j=0 then
                        print k+1;
            end
            pos:=pos+skip(text[pos]);
      end
end
```

BM法による文字列照合 skip関数

入力:キーワード key
出力:skip関数

文字種:p～q
n: keyの長さ

```
Method skip
begin
     for i:=p to q do
          skip(i):=n;
     for i:=1 to n-1 do
          skip(key[i]):=n-i;
end
```

初期設定(全ての文字種でkeyの長さだけskip)

Keyに含まれる文字種の場合keyの先頭から末尾まで調べて最後に見つかった位置をkeyの長さから引いた数だけskipする

BM法の評価

- 最良の場合　m/n回の文字照合
  $\text{textの文字} \cap \text{keyの文字} = \phi$
- 最悪の場合　m*n回の文字照合
  $\text{textの文字} = \text{keyの文字} = \{a\}$
- キーワードが長いほど高速
  - keyに含まれない文字がtextに出現したときにkeyの長さだけスキップできる
- 文字種類数が少ないほど遅くなる
  - text中の文字がkey中に現れる確率が高くなる　→　遅くなる

Aho-Corasick法

- マシンAC
- AC法の文字列照合手順
- AC法の文字列照合アルゴリズム
- AC法の評価
- マシンACの構成方法

Aho-Corasick法

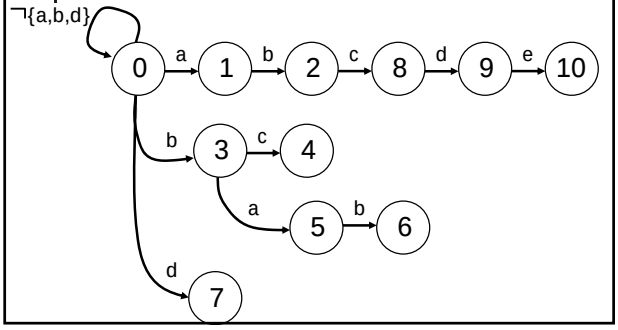
- 文書中から複数のキーワードを検索するための手法
- テキストをバックトラックすることなく1回走査するだけで，複数のキーワードを同時に検出することができる
- goto関数，failure関数，output関数により構成される

goto関数，failure関数，output関数

- goto関数
  - ある状態で文字xが入力されたときに遷移する状態
- failure関数
  - goto関数からfailが返された際の照合ポインタの移動先
- output関数
  - ある状態に遷移したときに検出できるキーワード

マシンAC　goto関数
キーワード {“ab”,”bc”,”bab”,”d”,”abcde”}

ある状態で文字xが入力されたときに遷移する状態

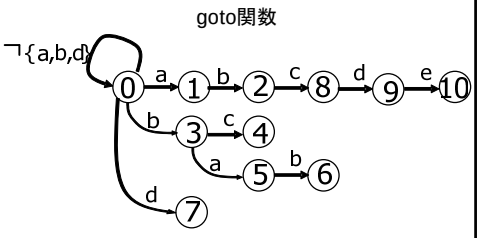


マシンAC　failure関数

goto関数からfailが返された際の照合ポインタの移動先

failure関数

s	f(s)
1	0
2	3
3	0
4	0
5	1
6	2
7	0
8	4
9	7
10	0

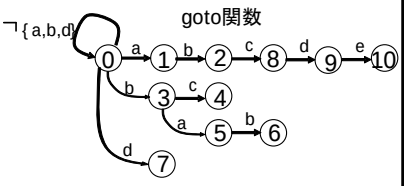


マシンAC　output関数

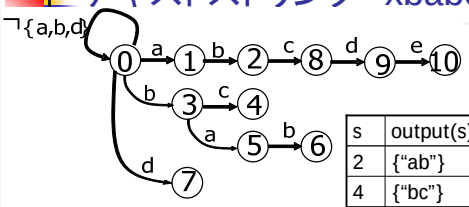
ある状態に遷移したときに検出できるキーワード

output関数

s	output(s)
2	{“ab”}
4	{“bc”}
6	{“bab”,”ab”}
7	{“d”}
8	{“bc”}
9	{“d”}
10	{“abcde”}



照合ポインタの遷移
テキストストリング ”xbabcdex”

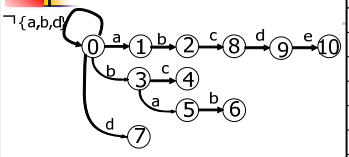


keyword
“ab”, ”bc”, ”bab”, ”d”, ”abcde”

s	f(s)
1	0
2	3
3	0
4	0
5	1
6	2
7	0
8	4
9	7
10	0

s	output(s)
2	{“ab”}
4	{“bc”}
6	{“bab”,”ab”}
7	{“d”}
8	{“bc”}
9	{“d”}
10	{“abcde”}

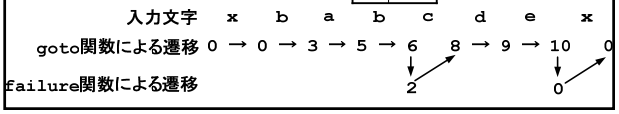
照合ポインタの遷移
テキストストリング ”xbabcdex”



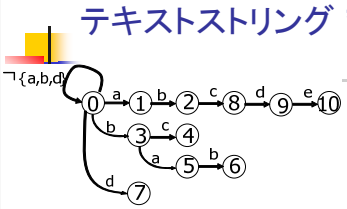
keyword
“ab”, ”bc”, ”bab”, ”d”, ”abcde”

s	f(s)
1	0
2	3
3	0
4	0
5	1
6	2
7	0
8	4
9	7
10	0

s	output(s)
2	{“ab”}
4	{“bc”}
6	{“bab”,”ab”}
7	{“d”}
8	{“bc”}
9	{“d”}
10	{“abcde”}



練習問題　照合ポインタの遷移
テキストストリング ”abcdbcba”



keyword
“ab”, ”bc”, ”bab”, ”d”, ”abcde”

s	f(s)
1	0
2	3
3	0
4	0
5	1
6	2
7	0
8	4
9	7
10	0

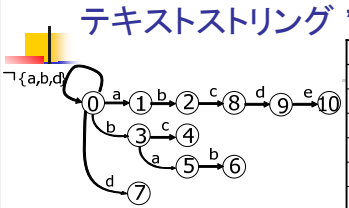
s	output(s)
2	{“ab”}
4	{“bc”}
6	{“bab”,”ab”}
7	{“d”}
8	{“bc”}
9	{“d”}
10	{“abcde”}

入力文字　a　b　c　d　b　c　b　a

goto関数による遷移

failure関数による遷移

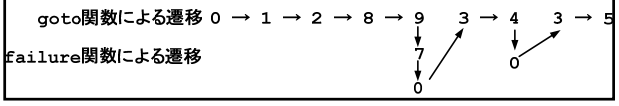
練習問題　照合ポインタの遷移
テキストストリング ”abcdbcba”



keyword
“ab”, ”bc”, ”bab”, ”d”, ”abcde”

s	output(s)
2	{“ab”}
4	{“bc”}
6	{“bab”,”ab”}
7	{“d”}
8	{“bc”}
9	{“d”}
10	{“abcde”}

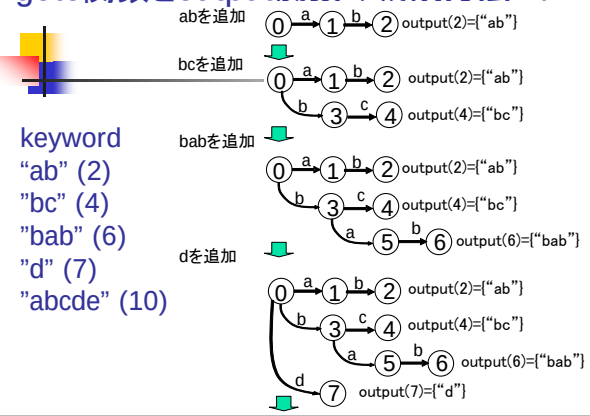
s	f(s)
1	0
2	3
3	0
4	0
5	1
6	2
7	0
8	4
9	7
10	0



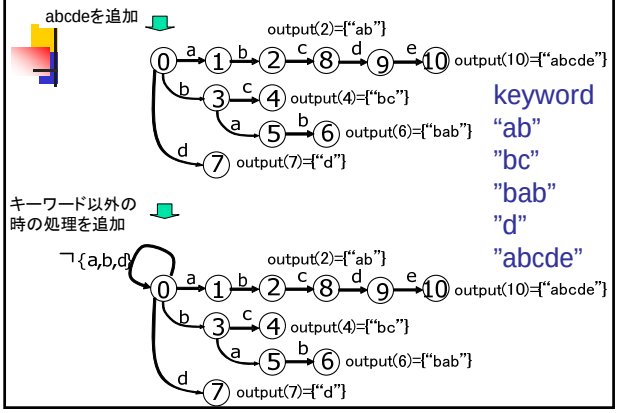
マシンACの構成方法

- goto関数とoutput関数の構成方法
- failure関数の構成方法

goto関数とoutput関数の構成方法 1/2



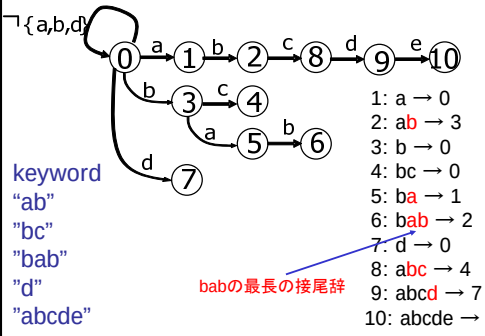
goto関数とoutput関数の構成方法 2/2



failure関数の構成方法

状態sのfailure関数

f(s)=q | ACstring[q]がACstring[s]の最長の接尾辞になる状態q



s	f(s)
1	0
2	3
3	0
4	0
5	1
6	2
7	0
8	4
9	7
10	0

データ圧縮

- 対象データ
  - テキスト
  - 音声
    - 音楽
    - 話し声
  - 画像
  - 動画
- 圧縮方式
  - 可逆圧縮（ロスレス圧縮）
  - 非可逆圧縮（ロッシー圧縮）

ジップの法則(Zipf's law)

「あるタイプの現象が生起する確率はその現象の生起する順位に反比例する」:経験則

$$\text{生起確率} = \frac{\text{定数}C}{\text{順位}}$$

- Zipfの法則が当てはまる事象
  - 文字毎の出現頻度
  - コンピュータにおけるコマンドの使用頻度
  - Webページのアクセス頻度
  - 都市の人口
  - 文献の参照回数
  - 会社でのランク（役職）と給料など
  - ケータイのシェア(docomo, au, softbank, e-mobile)

携帯電話：各グループ毎の加入者数累計 (2009年12月 ケータイWatchより)

順位	事業者	累計	割合(確率)	Zipf's law C=0.51
1	NTTドコモ	55,297,200	50.2%	51.0%
2	KDDI	31,329,400	28.4%	25.5%
3	ソフトバンク	21,501,900	19.5%	17.0%
4	イー・モバイル	2,048,200	1.8%	12.8%

$$\text{生起確率} = \frac{\text{定数}C}{\text{順位}}$$

自然言語の統計的性質

- 文字の使用頻度（英語） _はスペース

順位	文字	%	2	%	3	%	4	%
1	_	17.4	e_	3.0	_th	1.6	_the	1.2
2	e	9.7	_t	2.4	the	1.3	the_	1.0
3	t	7.0	th	2.0	he_	1.3	_of_	0.6
4	a	6.1	he	1.9	_of	0.6	and_	0.4
5	o	5.9	_a	1.7	of_	0.6	_and	0.4
6	i	5.5	s_	1.7	ed_	0.5	_to_	0.4
7	n	5.5	d_	1.5	_an	0.5	ing_	0.3