アルゴリズムとデータ構造III
12回目：12月17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

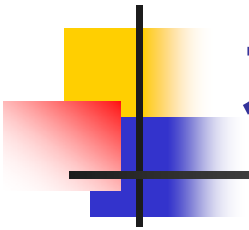
授業資料　http://ir.cs.yamanashi.ac.jp/~ysuzuki/algorithm3/index.html

授業の予定（中間試験まで）

1	10/01	スタック（後置記法で書かれた式の計算）
2	10/15	文脈自由文法，構文解析，CYK法
3	10/22	構文解析　CYK法
4	10/29	構文解析　CYK法
5	11/12	構文解析　CYK法，動的計画法
6	11/19	構文解析（チャート法），グラフ（ダイクストラ法）
7	11/26	グラフ（ダイクストラ法，DPマッチング，A*アルゴリズム）
8	12/03	グラフ（A＊アルゴリズム），前半のまとめ
9	12/04 4時限	教室：A1-41 全文検索アルゴリズム（simple search, KMP）

授業の予定(中間試験以降)

10	12/10	中間試験(8回目までの範囲)
11	12/11 4時限	教室:A1-41 全文検索アルゴリズム(BM, Aho-Corasick)
12	12/17	全文検索アルゴリズム(Aho-Corasick), データ圧縮
13	01/07	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	01/14	テキスト圧縮 (zip), 音声圧縮 (ADPCM, MP3, CELP), 画像圧縮(JPEG)
15	02/04	**期末試験**



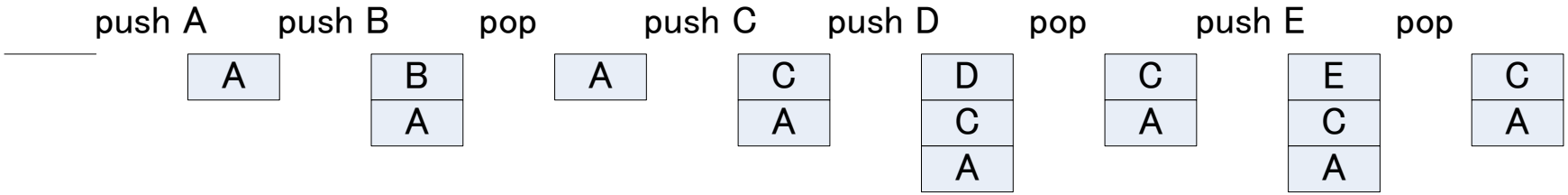
本日のメニュー

- 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

問題1.（スタック　平成21年秋期　基本情報技術者　午前　問5より）

- 空のスタックに対して次の操作を行った場合，スタックに残っているデータはどれか。ここで，“push　x”はスタックへデータ x を格納し，“pop”はスタックからデータを取り出す操作を表す。
- push A → push B → pop → push C → push D → pop → push E → pop

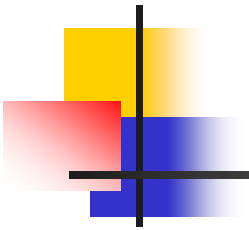
- 解答例：AとC



問題2.（文脈自由文法）

- 文脈自由文法と文脈依存文法の違いを200文字以内で説明せよ.

- 文脈依存文法の生成規則は　uαv→uβv　(αは非終端記号，β,u,vは終端または非終端記号列)の形で表される. これは非終端記号αの前後の記号u,vによりαからβの導出が制限される事を意味する.
uαv→uβv でuとvがεであるときα→βとなり，前後の記号(文脈)はαからβの導出に影響を与えない. そのためα→βのような生成規則だけを持つ文法を前後の文脈(記号)に対して影響を受けないという意味で文脈自由文法と呼ぶ.



問題3.（構文解析）

- 構文解析の代表的手法を3つ挙げよ.
- 解答例:
  - CYK法
  - Earley法（アーリー法）
  - トップダウンチャート法
  - LR法など

問題4.（CYK法）

- 下の図は「Nana met with Ramos. 」を構文解析中のCYK表である.
- 1）図中の①, ②, ③, ④, ⑤, ⑥には何が入るか答えよ.
- 2）CYK表から得られる「Nana met with Ramos.」の構文木を描け.

	Nana	met	with	Ramos
Nana	N→Nana	①	④	⑥
met		V→met	②	⑤
with			P → with	③
Ramos				N→Ramos

書き換え規則
S → N VP
S → N V
VP→ VP PP
VP→V PP
PP → P N
N → Nana
N → Ramos
V → met
P → with

①: S → N V
②: 該当なし
③: PP → P N
④: 該当なし
⑤: VP → V PP
⑥: S → N VP

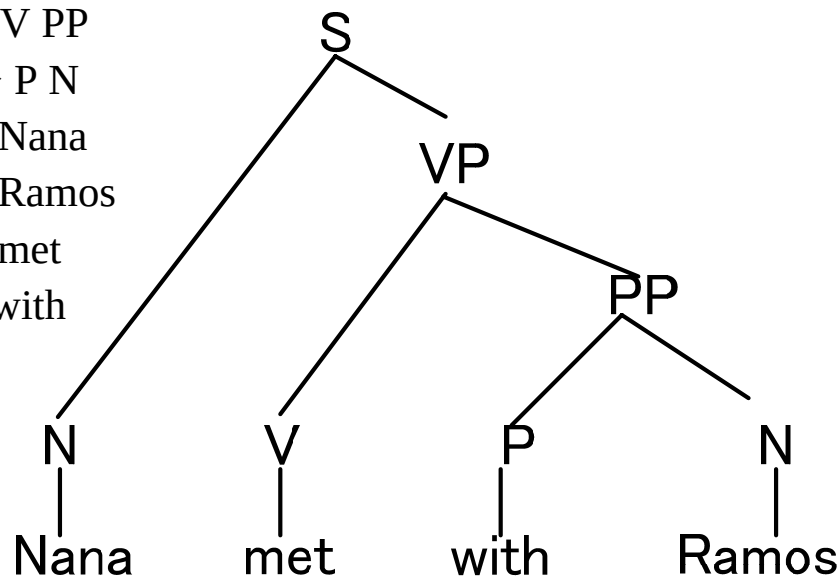
問題4.（CYK法）

- 下の図は「Nana met with Ramos. 」を構文解析中のCYK表である.
- 1） 図中の①, ②, ③, ④, ⑤, ⑥には何が入るか答えよ.
- 2）CYK表から得られる「Nana met with Ramos.」の構文木を描け.

	Nana	met	with	Ramos
Nana	N→Nana	①	④	⑥
met		V→met	②	⑤
with			P → with	③
Ramos				N→Ramos

書き換え規則

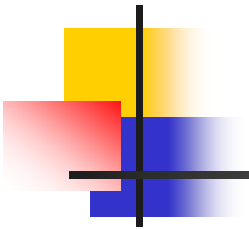
S → N VP
S → N V
VP→ VP PP
VP→V PP
PP → P N
N → Nana
N → Ramos
V → met
P → with



問題5.（トップダウンチャート法）

- CYK法と比較したときのトップダウンチャート法の特徴を簡潔に説明せよ.

- 文脈自由文法で書かれた文を構文解析するための代表的な手法
- アークとノードを使ったグラフで表される
- CKY法ではチョムスキーの標準形以外は扱えないが，チャート法では　X→ABCのような変換規則も扱うことができる.
- 簡単な予測を使うことが出来るため，CKY法より効率がよい



問題6.（動的計画法）

- 動的計画法を200文字以内で説明せよ.

- 解くのに時間のかかる問題を、複数の部分問題に分割することで効率的に解くアルゴリズム
- 動的計画法の適用例として，最短経路検索のためのダイクストラ法，パターンマッチングのためのDPマッチングがある.

問題7.（ダイクストラ法　平成15年　秋期　基本情報技術者　午後　問04より）

- 問題は長いので省略
- 解答例:
  - a: キ　Z=D[Y]
  - b: エ　S[Y]=T
  - c: カ　Y←S[Y]
  - d: ア　X>0

```
[プログラム]　プログラム名:SP(N,　C[ , ])
実数型:C[ , ],　D[N],　Z
整数型:P[N],　S[N],　W[N],　N,　T,　X,　Y
（初期設定）
X ← 1
while (X ≦ N){
    D[X] ← C[1, X]
    P[X] ← 0
    S[X] ← 1
    X ← X+1
}
P[1] ← 1
```

```
        （最短経路を求める処理）
        X ← 2
        while (X ≦ N){
            Y ← 2
            Z ← ∞
            while (Y ≦ N){
                if ((P[Y] = 0) and (D[Y] < Z)){
                    T ← Y
                    a: キ　Z=D[Y]
                }

                Y ← Y+1
            }
            P[T] ← 1
            Y ← 2
            while (Y ≦ N){
                if ((P[Y]=0) and (D[Y] >
(D[T]+C[T,Y]))){
                    D[Y] ← D[T] + C[T,Y]
                    b: エ　S[Y]=T
                }
                Y ← Y+1
            }
                X ← X+1
        }
```

```
（最短経路の出力処理）
X ← 1
Y ← N
while (Y ≠ 1){
    W[X] ← Y
    c: カ　Y←S[Y]
    X ← X+1
}
W[X] ← Y
while ( d: X>0 ){
    Output(W[X])
    X ←X − 1
}
```

問題8.（DPマッチング）

- 下の表は「abcd」と「accd」の単語間距離をDPマッチングにより計算しているところである．表中の①，②，③，④，⑤には何が入るか答えよ．但し，不一致ペナルティは3点，挿入ペナルティ=1，脱落ペナルティ=1とする．

通行ペナルティ積算表

	a	b	c	d
a	0	4	8	12
c	4	3	4	①
c	8	7	②	③
d	12	11	④	⑤

①：8
②：3
③：7
④：7
⑤：3

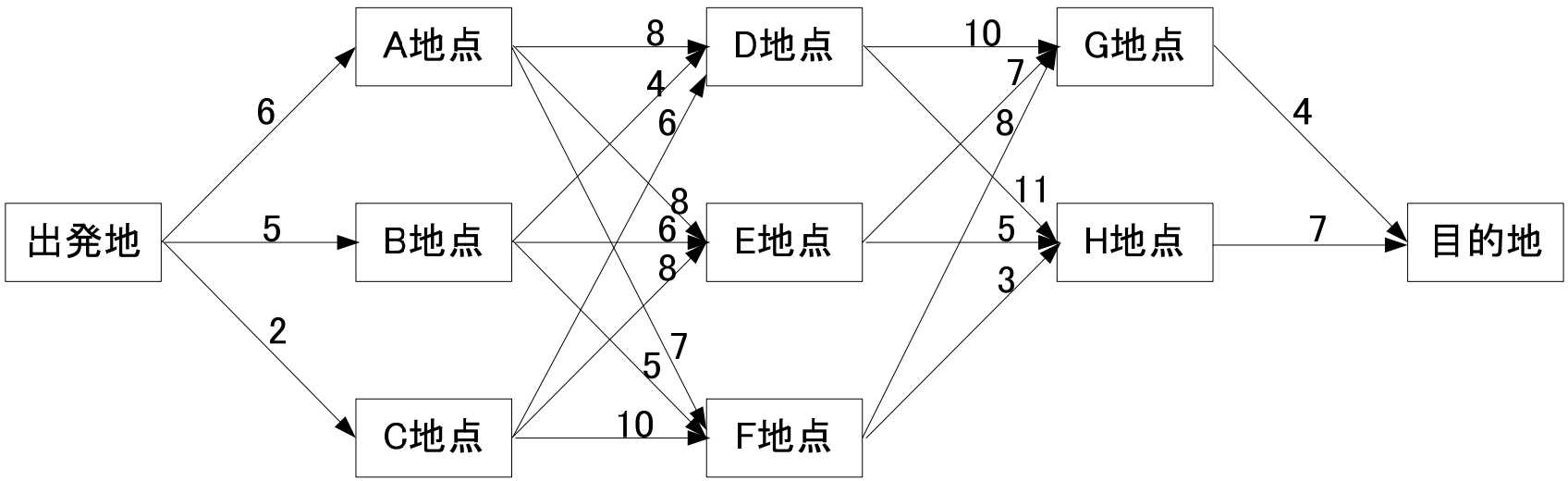
問題9.（A*アルゴリズム）

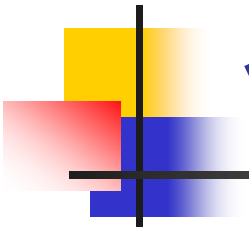
- A*アルゴリズムとダイクストラ法の類似点と相違点を300文字以内で説明せよ

- 類似点
  - どちらも最短経路問題を解くのに使われる.
  - マイナスのコストをもつ辺を含む経路の最短経路は解くことが出来ない.
- 相違点
  - ダイクストラ法はスタートから各節点までの移動コストを利用して最短経路を求めるのに対してA*アルゴリズムはスタートから各節点までの移動コストと節点からゴールまでの予測コスト（0≦予測コスト≦実際のコスト）を利用する.
  - A*アルゴリズムは予測コストを利用するためダイクストラ法よりも効率よく問題を解くことが出来る. 各節点からゴールまでの予測コストがすべて0である場合，ダイクストラ法のアルゴリズムと同じになる.

問題10.（平成19年　春期　基本情報技術者　午前　問78より）

- 図中の矢印に記した数値は、各区間の運賃を表す。出発地から目的地までの経路のうち、最も安い総運賃はいくらか。また、その時の経路を示せ。
- 解答例:
  - 最も安い総運賃:20
  - そのときの経路:出発地→B地点→F地点→H地点→目的地





全文検索

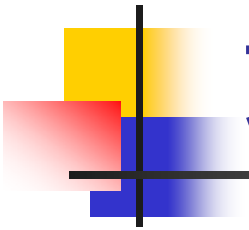
- 文書中から，与えられた文字列と完全に一致する部分を探し出す．

- 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

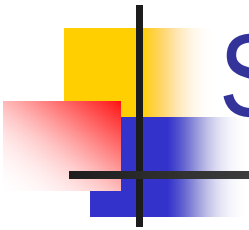
a	b	c	a	b	c	a	b	a	b	c	a	b	a	b	x	a	b	c	a
			a	b	c	a	b	a											
								a	b	c	a	b	a						



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

- テキストストリングの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:テキストストリング　text, キーワード key
- 出力:テキストストリング中のキーワードの位置
- m: テキストストリングの長さ
- 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

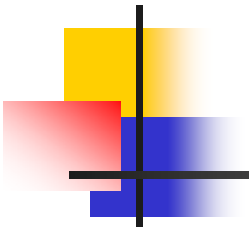
文字照合回数 (7-3+1)*3=15

(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

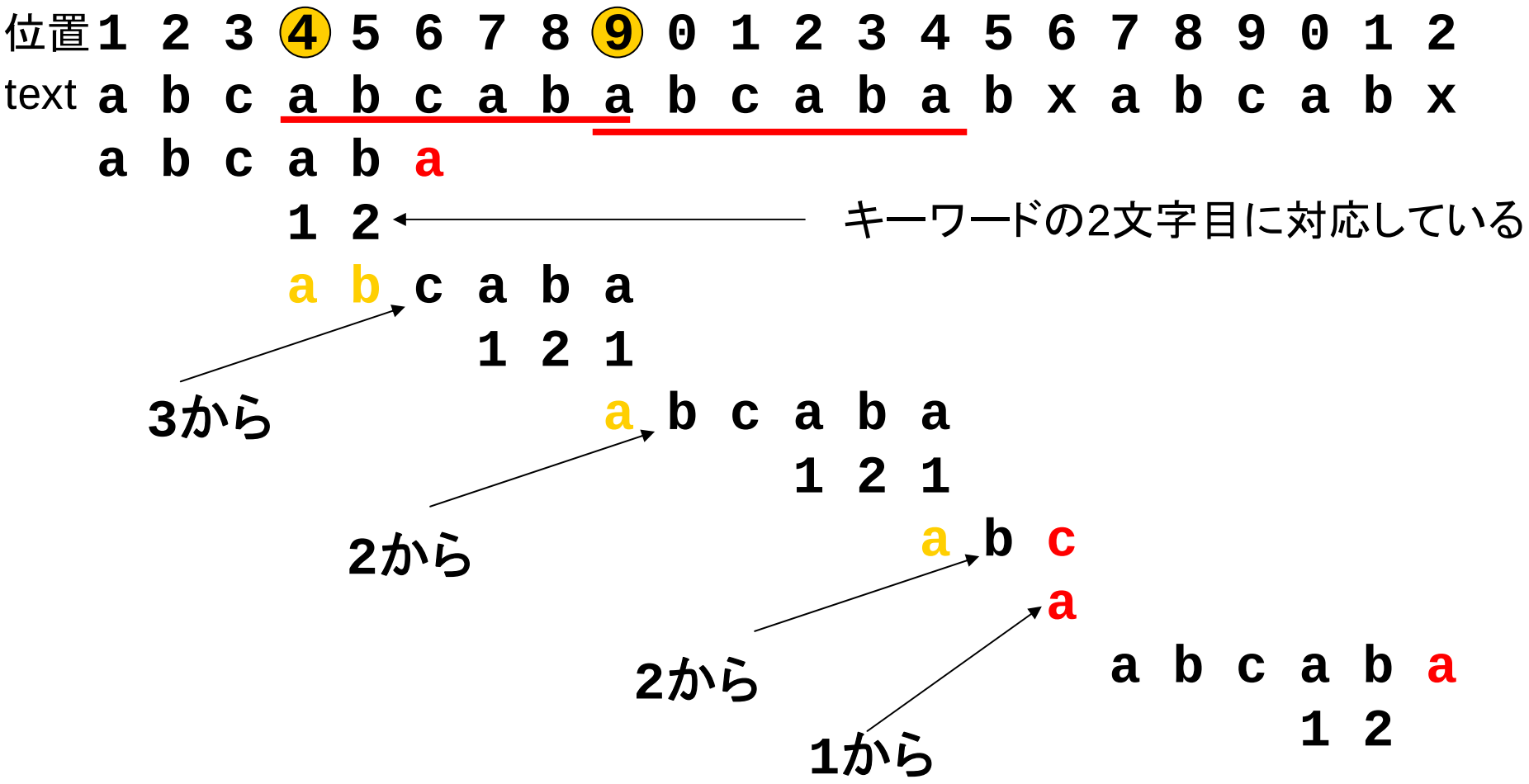


Knuth-Morris-Pratt法 （KMP法）

- Simple Search
  - テキストストリング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

```
Key:  a b c a b a
      1 2 3 4 5 6
next  0 1 1 0 1 3 2
```



KMP法　アルゴリズム

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

```
Method kmp
begin
      j:=1;
      for i:=1 to m do
      begin
            while j>0 and key[j] ≠text[i] do    照合
                  j:=next(j);  つぎの照合位置
            if j=n then                          ┐
                  print i-n+1:                   ┘ 照合成功
            j:=j+1;
      end
end
```

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

テキストストリング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
begin
        t:=0;
        next(1):=0;
        for j:=1 to n do
        begin
                while t ≠ 0 and key[j] ≠ key[t] do
                        t:=next(t);
                t:=t+1;
                if key[j+1]=key[t] then
                        next(j+1):=next(t);
                else
                        next(j+1):=t;
        end
end
```

n：keyの長さ
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

keyの各文字に対してnext関数値を計算

keyのj文字目までの文字列がkeyの接頭辞と一致しているか調べる

keyのj+1文字目のnext関数値を決定

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: テキスト文字列数
n: キーワード文字列数

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

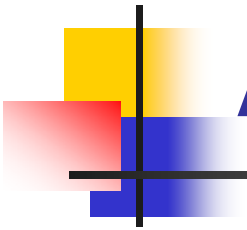
```
Method skip
begin
      for i:=p to q do
            skip(i):=n;
      for i:=1 to n-1 do
            skip(key[i]):=n-i;
end
```

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
- テキストストリング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

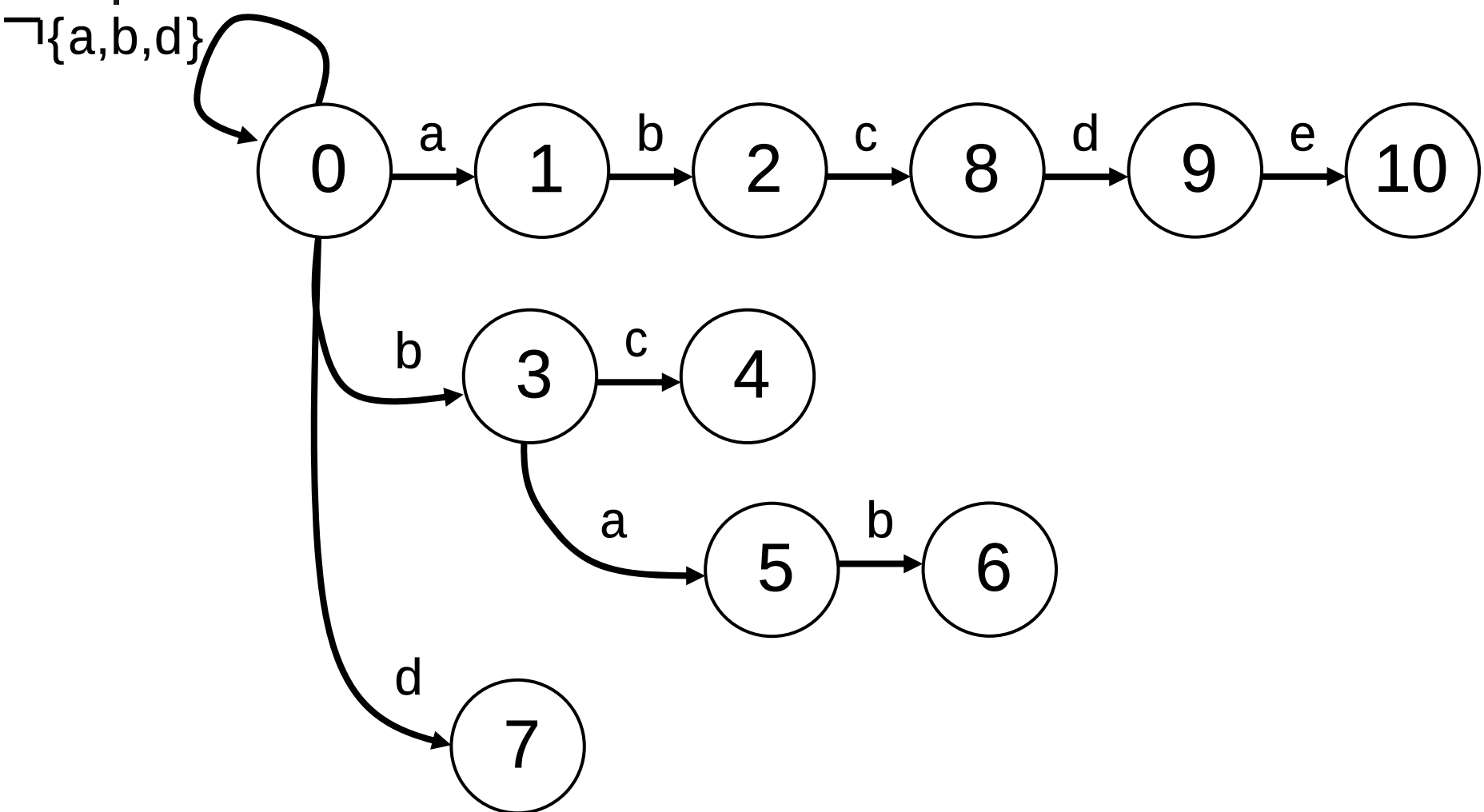
- goto関数，failure関数，output関数により構成される

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

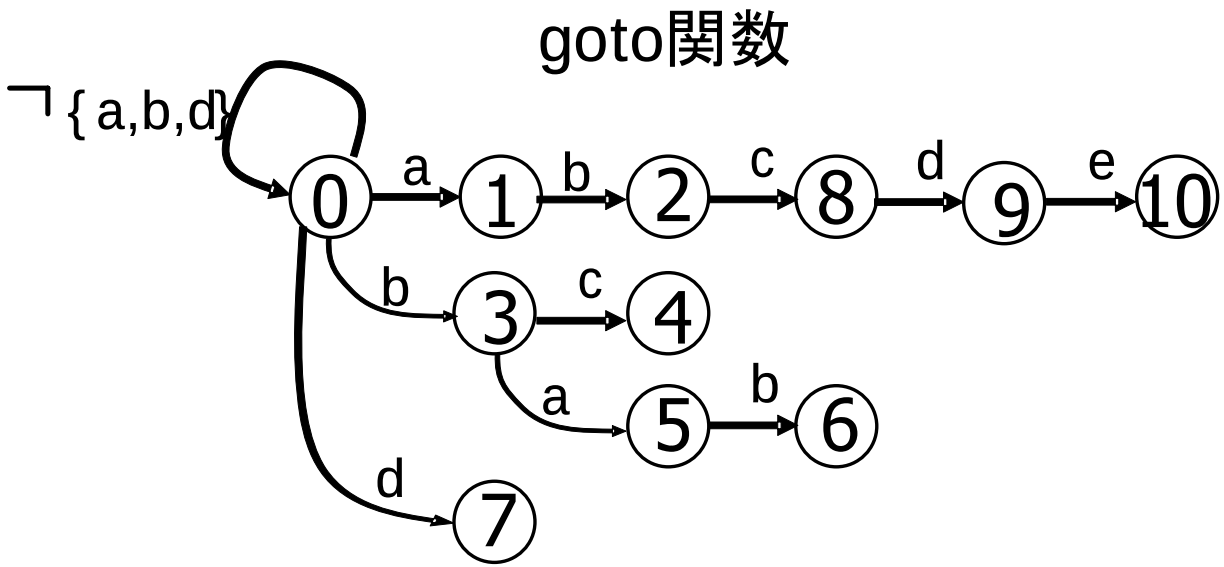


マシンAC　failure関数

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

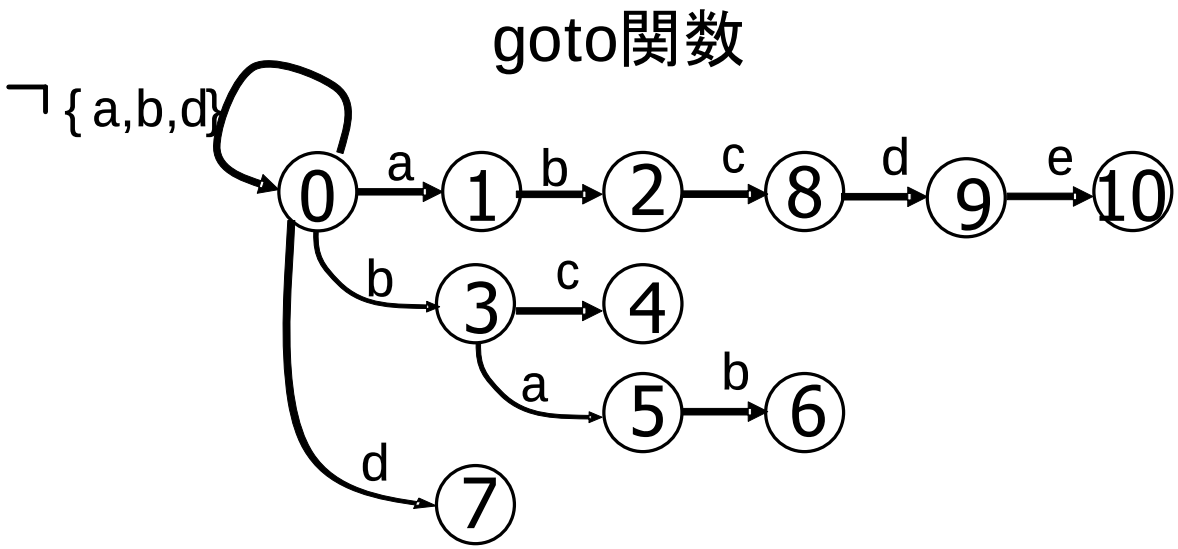


マシンAC　output関数

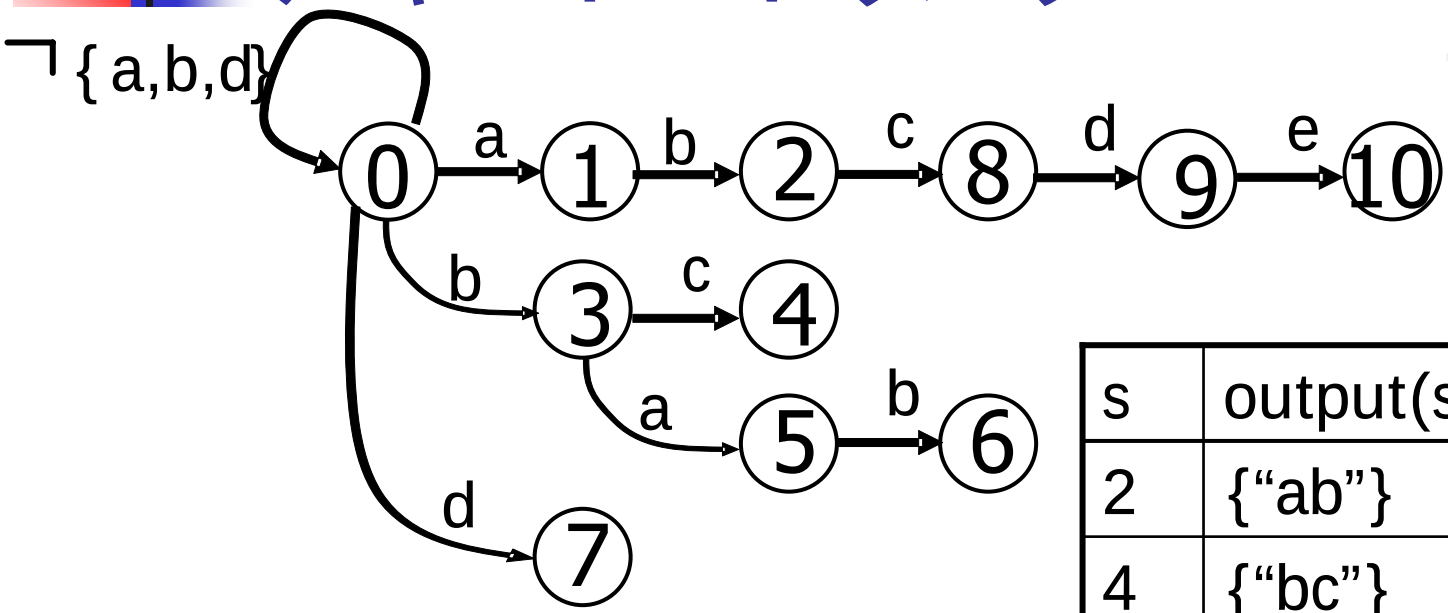
ある状態に遷移したときに検出できるキーワード

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”

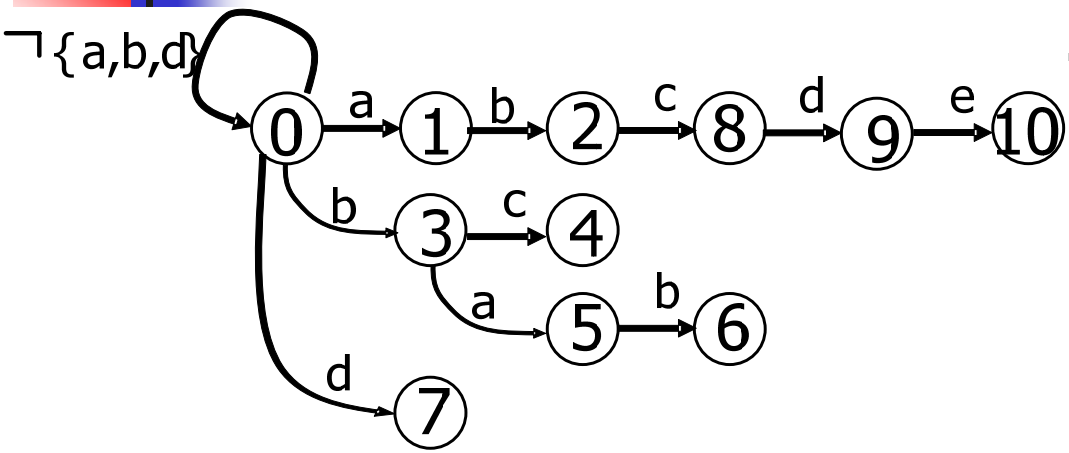


keyword
“ab”, ”bc”, ”bab”, ”d”, ”abcde”

s	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

照合ポインタの遷移
テキストストリング ”xbabcdex”



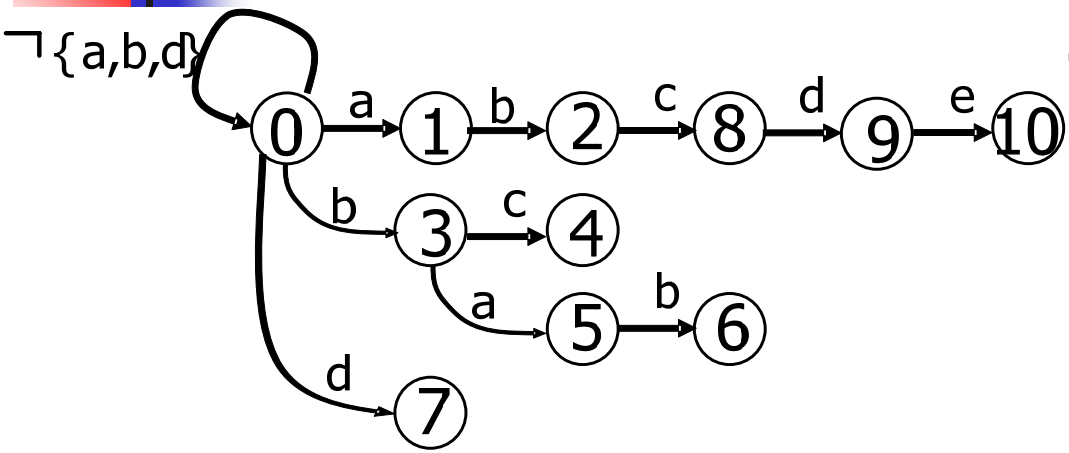
s	f(s)
1	0
2	3
3	0
4	0
5	1
6	2
7	0
8	4
9	7
10	0

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

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

入力文字	x	b	a	b	c	d	e	x
goto関数による遷移	0 → 0	→ 3	→ 5	→ 6		8 → 9	→ 10	0
failure関数による遷移					2		0	

練習問題　照合ポインタの遷移
テキストストリング ”abcdbcba”



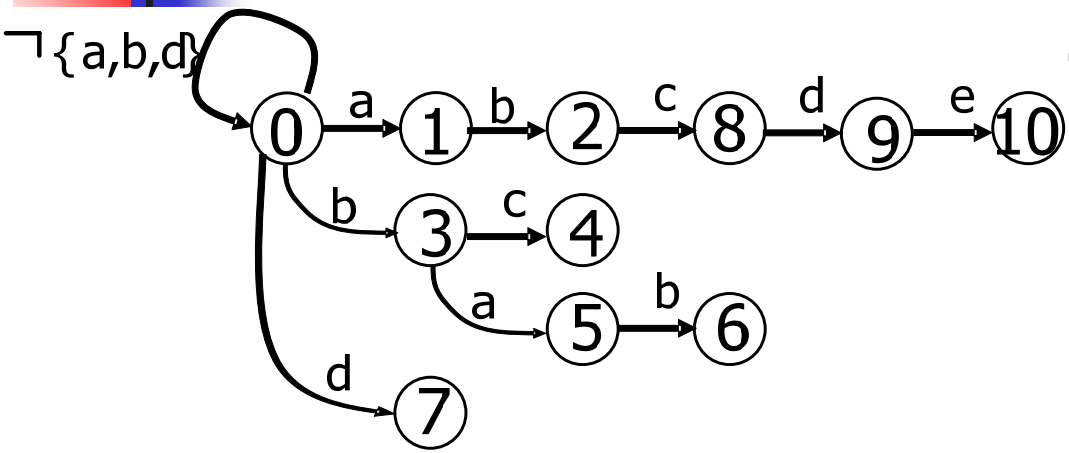
keyword
“ab”, ”bc”, ”bab”, ”d”, ”abcde”

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”



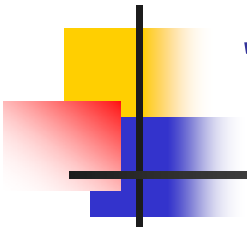
s	output(s)
2	{“ab”}
4	{“bc”}
6	{“bab”,”ab”}
7	{“d”}
8	{“bc”}
9	{“d”}
10	{“abcde”}

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

keyword

“ab”, ”bc”, ”bab”, ”d”, ”abcde”

入力文字	a	b	c	d	b	c	b	a
goto関数による遷移	0 → 1	→ 2	→ 8	→ 9	3	→ 4	3	→ 5
failure関数による遷移				7 → 0		0		



マシンACの構成方法

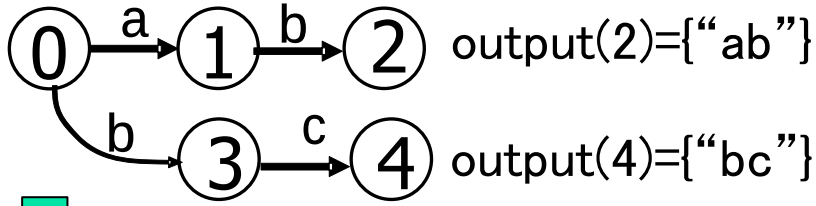
- goto関数とoutput関数の構成方法
- failure関数の構成方法

goto関数とoutput関数の構成方法 1/2

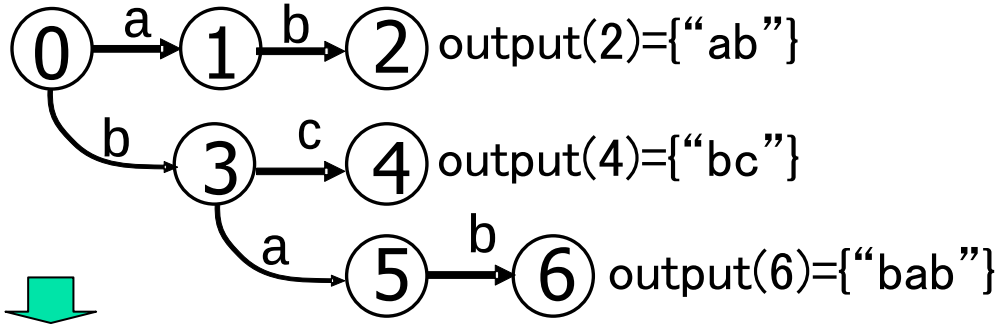
keyword
“ab” (2)
”bc” (4)
”bab” (6)
”d” (7)
”abcde” (10)

abを追加

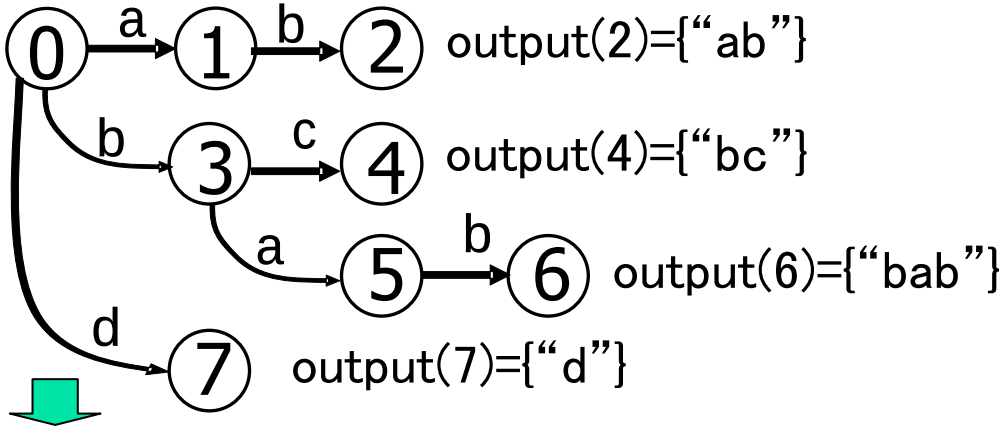
bcを追加



babを追加

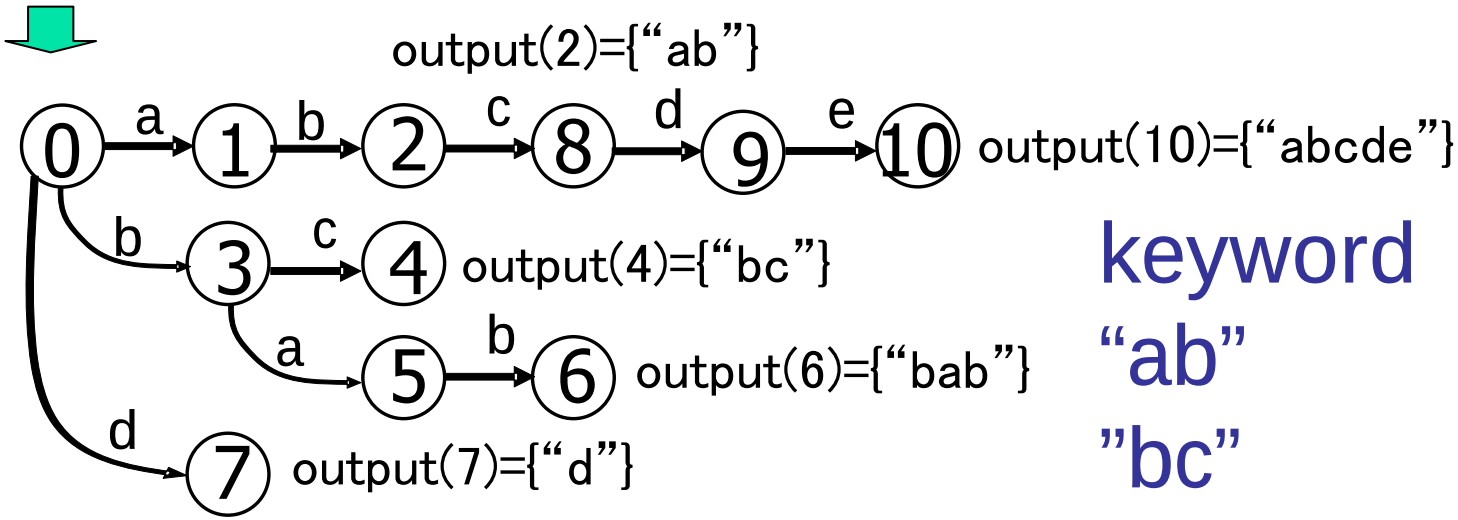


dを追加



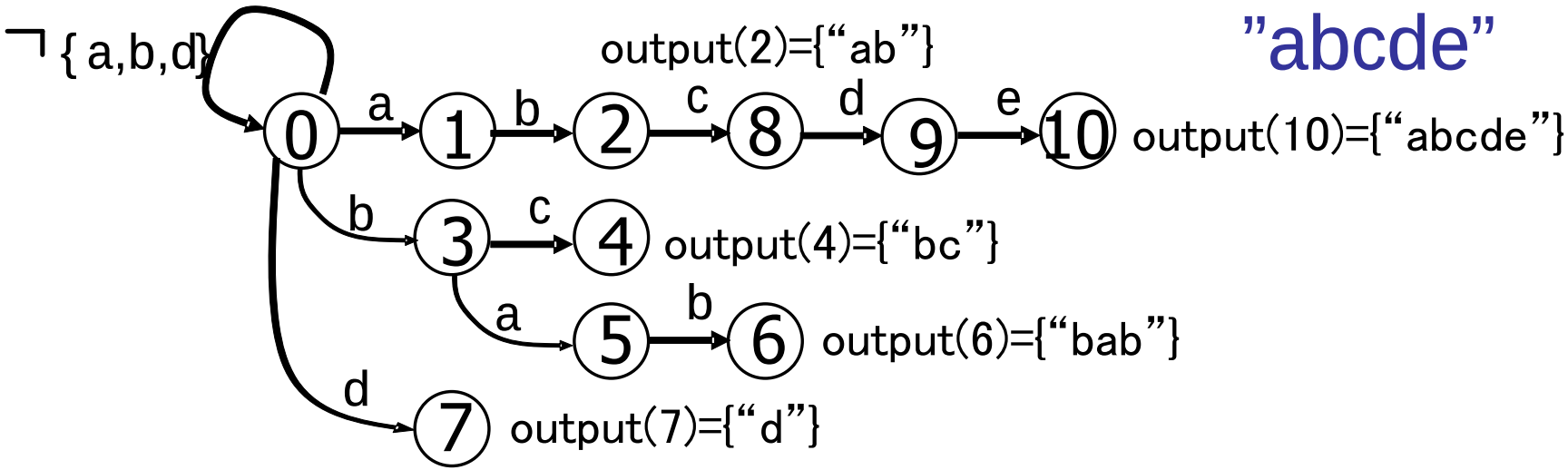
goto関数とoutput関数の構成方法 2/2

abcdeを追加



keyword
“ab”
”bc”
”bab”
”d”
”abcde”

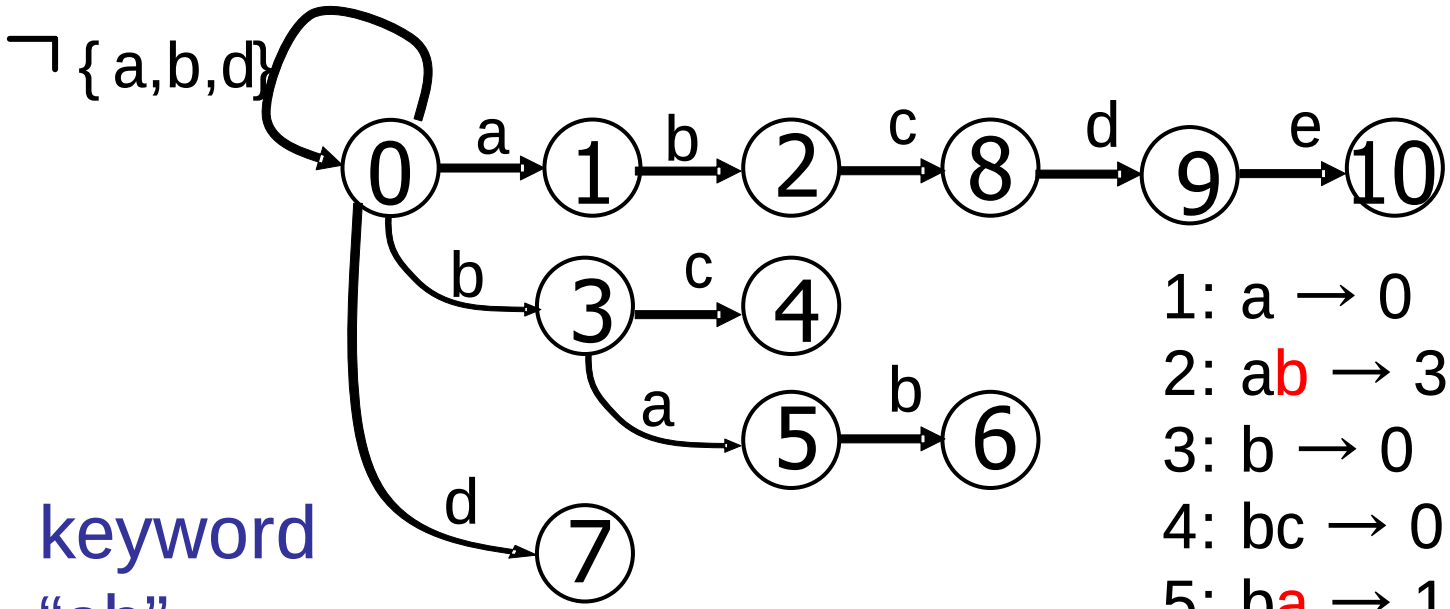
キーワード以外の
時の処理を追加



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



keyword

“ab”

”bc”

”bab”

”d”

”abcde”

1: a → 0

2: ab → 3

3: b → 0

4: bc → 0

5: ba → 1

6: bab → 2

7: d → 0

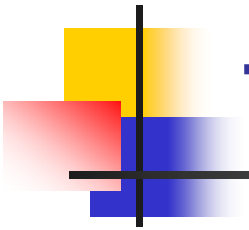
8: abc → 4

9: abcd → 7

10: abcde → 0

babの最長の接尾辞

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）

モールス信号の符号

- ・（短点）と－（長点）を用いてアルファベットを表現する
- 情報を早く送るための工夫
  - よく使われる文字（例えばe,t）は短い
    - e: ・ （短点1文字）
    - t: － （長点1文字）
  - あまり使われない文字（例えばqは4文字）は長い
    - q: －－・－

モールス信号の符号

- ・(短点)と－(長点：短点3つ分の長さ)を用いてアルファベットを表現する
- 区切り記号
  - 文字の切れ目：短点3つ分の間隔
  - 単語の切れ目：短点7つ分の間隔
- L: ・－・・ （LifeカードのCMに使われていた）
- SOS：・・・ －－－ ・・・