

INF2220 - Algoritmer og datastrukturer

HØSTEN 2008

Institutt for informatikk, Universitetet i Oslo

INF2220, forelesning 11:
Tekstalgoritmer 1

Bjarne Holen (Ifi, UiO)

INF2220

H2008, forelesning 11

1 / 1

Lokalisering av Substrenger

c	o	z	w	e	r	a	...	u
---	---	---	---	---	---	---	-----	---

Høystakk

c	y	g
---	---	---

Nål

- ▶ Fins nåla i høystakken?
- ▶ Hvis JA: hvor?

Pattern matching algorithms

Algoritmer for lokalisering av substrrenger

- ▶ Brute force
 - ▶ Enkleste tenkelige algoritme for å løse problemet
- ▶ `java.lang.String indexOf`
 - ▶ Vi ser på hvordan det gjøres i standard biblioteket
- ▶ Boyer Moore (Horspool)
 - ▶ Relativt komplisert algoritme, med rask **wort case**

Bjarne Holen (Ifi, UiO)

INF2220

H2008, forelesning 11

3 / 1

Bjarne Holen (Ifi, UiO)

INF2220

H2008, forelesning 11

4 / 1

Goodrich & Tamassia - Chapter 12.2.1

The **brute force** algorithmic design pattern is a powerful technique for algorithm design when we have something we wish to search for or when we wish to optimize some function.

tja

Brute force

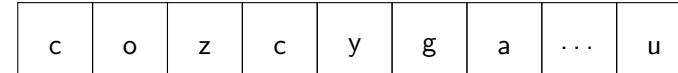
Brute force (rå kraft) brukes ofte synonymt med

- ▶ unødvendig tung
- ▶ dårlig
- ▶ treg
- ▶ lite gjennomtenkt
- ▶ nødløsning

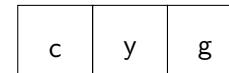
men er noen ganger nødvendig

- ▶ Brute force løsninger er typisk den første ideen vi får
- ▶ Stort sett hele dette kurset går ut på å unngå de

Brute force



Høystakk



Nål

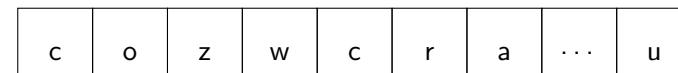
- ▶ Var ikke det den første algoritmen du tenkte på?
- ▶ Hva blir kompleksiteten av et sånt søk?

java.lang.String indexOf

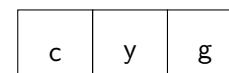
Hvordan løser standard biblioteket til Java denne oppgaven?

- ▶ String klassen har en funksjon som lokaliserer substring
- ▶ Rimelig å tro at standard biblioteket er bra implementert
- ▶ Vi skal se på hvordan `indexOf` er implementert, samt prøve å forklare hvorfor de har brukt den algoritmen
- ▶ Vi skal se på algoritmer som er raskere, gitt at visse forutsetninger er oppfylt

java.lang.String indexOf



Høystakk



Nål

- ▶ Vi lagrer første element i **nål**
- ▶ Så leiter vi etter match på første element før vi flytter **nål**

java.lang.String indexOf

- ▶ Andre **pattern matching** algoritmer har lavere kompleksitet
- ▶ Hvorfor bruker standard biblioteket en såpass enkel algoritme?
- ▶ Er det hensyn Java må ta som vi slipper å ta?

Boyer Moore

- ▶ Skal vi øke hastigheten må vi minske antall sammenligninger
 - ▶ Vi kan preprosessere informasjonen i **nål** og **høystakk**
 - ▶ Vi kan gjøre **rimelige** antagelser om input
- ▶ Boyer Moore antar at vi bare har 1-byte characters
- ▶ 1-byte characters gir oss 256 muligheter ($2^8 = 256$)
- ▶ Vi kan bruke den informasjonen til å preprosessere **nålen**

Boyer Moore

c	o	z	w	c	r	a	...	u
---	---	---	---	---	---	---	-----	---

Høystakk

c	y	g
---	---	---

Nål

- ▶ Vi matcher baklengs med Boyer Moore
- ▶ Merk at elementet **z ikke** finnes i nålen
- ▶ Dvs. etter første match kan nålen flyttes **3** hakk frem
- ▶ Informasjonen vi trenger for å beregne dette ligger i arrayen vi kaller **bad character shift**

Bad Character Shift

- ▶ Hvordan beregne **bad character shift**?
- ▶ Vi må raskt kunne svare på om en bokstav er med i **nålen**
- ▶ Bokstaver er 1-byte lange dvs. (**int**) bokstav $\in [0, 255]$
- ▶ **badCharShift** er en array **int[256]**
- ▶ Vi fyller denne med **shift** verdier ut i fra hva som er i **nålen**

bad character shift (forenklet)

```
int[] badCharShift = new int[256]; // assume 1-byte characters

for(int i = 0; i < badCharShift.length; i++){
    badCharShift[i] = needle.length;
}

/* shift size = 1 for characters inside needle */
for(int i = 0; i < needle.length; i++){
    badCharShift[ (int) needle[i] ] = 1;
}
```

bad character shift (forenklet)

Hvordan ser preprosesseringen ut hvis vi bruker **nålen** fra i sta?

c	y	g
---	---	---

Nål

- ▶ badCharShift[(int) 'c'] == 1 99 == (int) 'c'
- ▶ badCharShift[(int) 'y'] == 1 121 == (int) 'y'
- ▶ badCharShift[(int) 'g'] == 1 103 == (int) 'g'
- ▶ **For alle andre verdier $x \in [0, 255]$**
- ▶ badCharShift[x] == 3 (needle.length == 3)

bad character shift (forenklet)

c	o	z	w	r	c	a	v	u
---	---	---	---	---	---	---	---	---

Høystakk

c	y	g
---	---	---

Nål

- ▶ badCharShift[(int) 'z'] == 3 122 == (int) 'z'
- ▶ badCharShift[(int) 'c'] == 1 99 == (int) 'c'
- ▶ badCharShift[(int) 'a'] == 3 97 == (int) 'a'

bad character shift

```
int[] badCharShift = new int[256]; // assume 1-byte characters

for(int i = 0; i < badCharShift.length; i++){
    badCharShift[i] = needle.length;
}

/* calculate bad shift up to needle.length - 1 */

int last = needle.length - 1;

for(int i = 0; i < last; i++){
    badCharShift[ (int) needle[i] ] = last - i;
}
```

bad character shift

Hvordan ser preprosesseringen ut hvis vi bruker **nålen** fra i sta?

c	y	g
---	---	---

Nål

- ▶ badCharShift[(int) 'c'] == 2 99 == (int) 'c'
- ▶ badCharShift[(int) 'y'] == 1 121 == (int) 'y'
- ▶ **For alle andre verdier** $x \in [0, 255]$
- ▶ badCharShift[x] == 3 (needle.length == 3)

c	o	z	w	r	c	a	y	u
---	---	---	---	---	---	---	---	---

Høystakk

c	y	g
---	---	---

Nål

- ▶ badCharShift[(int) 'z'] == 3 122 == (int) 'z'
- ▶ badCharShift[(int) 'c'] == 2 99 == (int) 'c'
- ▶ badCharShift[(int) 'y'] == 1 121 == (int) 'y'

Boyer Moore Horspool

```
public int boyer_moore_horspool(char[] needle, char[] haystack){  
    if ( needle.length > haystack.length ) { return -1; }  
  
    int[] bad_shift = new int[CHAR_MAX]; // 256  
    for(int i = 0; i < CHAR_MAX; i++){  
        bad_shift[i] = needle.length;  
    }  
  
    int offset = 0, scan = 0;  
    int last = needle.length - 1;  
    int maxoffset = haystack.length - needle.length;  
  
    for(int i = 0; i < last; i++){  
        bad_shift[needle[i]] = last - i;  
    }  
  
    while(offset <= maxoffset){  
  
        for(scan = last; needle[scan] == haystack[scan+offset]; scan--){  
            if(scan == 0){ // match found!  
                return offset;  
            }  
        }  
        offset += bad_shift[haystack[offset + last]];  
    }  
    return -1;  
}
```

Boyer Moore Horspool

- ▶ **Horspool** tilnavnet er der fordi vi mangler **good suffix shift**
- ▶ **good suffix shift** beregner hvor langt vi kan flytte nålen, basert på antall matchende bokstaver før mismatch
- ▶ Alle algoritmer gjennomgått, blir lagt ut:
 - ▶ **brute force**
 - ▶ **java.lang.String indexOf**
 - ▶ forenklet **bad character shift**
 - ▶ **Boyer Moore Horspool**
- ▶ Mulig ukeoppgave: legg til **good suffix shift** til Boyer Moore Horspool