

# Talningarfræði

Bergur Snorrason

March 18, 2024

- ▶ Talningarfræði er sá angi strjállar stærðfræði sem fjallar um talningar á einhverjum fyrirbærum.
- ▶ Þegar við fjölluðum um tæmandi leit kom fram að fjöldi hlutmengja í  $n$  staka mengi er  $2^n$ .
- ▶ Einnig kom fram að fjöldi umraðana á menginu  $\{1, 2, \dots, n\}$  er  $n!$ .
- ▶ Bæði eru þetta mikilvægar niðurstöður úr talningarfræði.
- ▶ Skerpum aðeins á grunnatriðum.

- ▶ Ef við erum með  $n$  hluti af einni gerð og  $m$  hluti af annari gerð þá getum við valið einn hlut af hvorri gerð á  $n \cdot m$  vegu.
- ▶ Við þurfum í raun ekki meira en þetta.
- ▶ Við notuðum þessa reglu til að sanna niðurstöðurnar á glærunni á undan.
- ▶ Í talningarfræði er oft þægilegt að hugsa um endanleg mengi og fjöldatölur þeirra.
- ▶ Ef  $A$  er mengi þá táknar  $|A|$  fjölda staka í  $A$ .
- ▶ Við getum þá umorðað efsta punktinn sem: Ef  $A$  og  $B$  eru mengi þá er  $|A \times B| = |A| \cdot |B|$ .

- ▶ Við vitum að það eru  $2^n$  hlutmengi í  $n$  staka mengi, en hvað eru mörg hlutmengi af stærð  $k$ ?
- ▶ Þegar við veljum fyrsta stakið höfum við um  $n$  stök að velja, síðan  $n - 1$  stak og svo framvegis.
- ▶ Við fáum því  $n \cdot (n - 1) \cdot \dots \cdot (n - k + 1) = \frac{n!}{k!}$  mengi.
- ▶ Hvert mengi er þó talið  $(n - k)!$  sinnum, svo loka talan er

$$\frac{n!}{(n - k)!k!}.$$

- ▶ Þessi tala er táknuð með  $\binom{n}{k}$ .

- ▶ Tökum eftir að  $|A \cup B| \neq |A| + |B|$  því það gætu verið stök í bæði  $A$  og  $B$ .
- ▶ Ef svo er getum við einfaldlega fjarlægt þau stök sem eru tvítalin, og fáum

$$|A \cup B| = |A| + |B| - |A \cap B|.$$

- ▶ Gerum nú ráð fyrir að

$$|A_1 \cup \dots \cup A_n| = \sum_{\substack{J \subset \{1, \dots, n\} \\ J \neq \emptyset}} (-1)^{|J|-1} \left| \bigcap_{j \in J} A_j \right|.$$

- ▶ Þá fæst...

$$\begin{aligned}
|A_1 \cup \dots \cup A_{n+1}| &= |A_1 \cup \dots \cup A_n| + |A_{n+1}| - |(A_1 \cup \dots \cup A_n) \cap A_{n+1}| \\
&= \sum_{\substack{J \subset \{1, \dots, n\} \\ J \neq \emptyset}} (-1)^{|J|-1} \left| \bigcap_{j \in J} A_j \right| + |A_{n+1}| - |(A_1 \cap A_{n+1}) \cup \dots \cup (A_n \cap A_{n+1})| \\
&= \sum_{\substack{J \subset \{1, \dots, n\} \\ J \neq \emptyset}} (-1)^{|J|-1} \left| \bigcap_{j \in J} A_j \right| + |A_{n+1}| + \sum_{\substack{J \subset \{1, \dots, n\} \\ J \neq \emptyset}} (-1)^{|J|} \left| \bigcap_{j \in J} (A_j \cap A_{n+1}) \right| \\
&= \sum_{\substack{J \subset \{1, \dots, n\} \\ J \neq \emptyset}} (-1)^{|J|-1} \left| \bigcap_{j \in J} A_j \right| + |A_{n+1}| + \sum_{\substack{J \subset \{1, \dots, n\} \\ J \neq \emptyset}} (-1)^{|J|} \left| \bigcap_{j \in J \cup \{n+1\}} A_j \right| \\
&= \sum_{\substack{J \subset \{1, \dots, n\} \\ J \neq \emptyset}} (-1)^{|J|-1} \left| \bigcap_{j \in J} A_j \right| + \sum_{J \subset \{1, \dots, n\}} (-1)^{|J|} \left| \bigcap_{j \in J \cup \{n+1\}} A_j \right| \\
&= \sum_{\substack{J \subset \{1, \dots, n+1\} \\ J \neq \emptyset \\ n+1 \notin J}} (-1)^{|J|-1} \left| \bigcap_{j \in J} A_j \right| + \sum_{\substack{J \subset \{1, \dots, n+1\} \\ n+1 \in J}} (-1)^{|J|-1} \left| \bigcap_{j \in J} A_j \right| \\
&= \sum_{\substack{J \subset \{1, \dots, n, n+1\} \\ J \neq \emptyset}} (-1)^{|J|-1} \left| \bigcap_{j \in J} A_j \right|
\end{aligned}$$

- ▶ Við höfum því sýnt með þrepun að

$$|A_1 \cup \dots \cup A_n| = \sum_{\substack{J \subset \{1, \dots, n\} \\ J \neq \emptyset}} (-1)^{|J|-1} \left| \bigcap_{j \in J} A_j \right|.$$

- ▶ Vera má að þessi jafna komi spánkst fyrir sjónir en í raun lýsir hún hvernig við fjarlægjum stök sem eru tvítekin, bætum aftur við stökum sem eru þrítekin, fjarlægjum aftur stök sem eru fjórtekin og svo framvegis.
- ▶ Þessi jafna er kölluð *lögmálið um fjöldatölu sammengja* (e. *Inclusion-Exclusion principle*).

- ▶ Sjáum nú hagnýtingu á þessari jöfnu.
- ▶ Munum að gagntæk vörpun  $\sigma: \{1, 2, \dots, n\} \rightarrow \{1, 2, \dots, n\}$  kallast *umröðun* (e. *permutation*).
- ▶ Ef við festum  $n$  þá höfum við sýnt að til séu  $n!$  umraðanir.
- ▶ Næst segjum við að  $k \in \{1, 2, \dots, n\}$  sé *fastapunktur*  $\sigma$  (e. *fixed point of  $\sigma$* ) ef  $\sigma(k) = k$ .
- ▶ Með öðrum orðum hefur umröðunin ekki áhrif á þennan punkt.
- ▶ Hversu margar umraðanir hafa engan fastapunkt?



► Skoðum fyrst þegar  $n = 4$ :

1 2 3 4	2 1 4 3	3 1 2 4	4 1 2 3
1 2 4 3	2 1 3 4	3 1 4 2	4 1 3 2
1 3 2 4	2 3 1 4	3 2 1 4	4 2 1 3
1 3 4 2	2 3 4 1	3 2 4 1	4 2 3 1
1 4 2 3	2 4 1 3	3 4 1 2	4 3 1 2
1 4 3 2	2 4 3 1	3 4 2 1	4 3 2 1

► Skoðum fyrst þegar  $n = 4$ :

1 2 3 4 ^ ^ ^ ^	2 1 4 3	3 1 2 4 ^	4 1 2 3
1 2 4 3 ^ ^	2 1 3 4 ^ ^	3 1 4 2	4 1 3 2 ^
1 3 2 4 ^       ^	2 3 1 4 ^	3 2 1 4 ^   ^	4 2 1 3 ^
1 3 4 2 ^	2 3 4 1	3 2 4 1 ^	4 2 3 1 ^ ^
1 4 2 3 ^	2 4 1 3	3 4 1 2	4 3 1 2
1 4 3 2 ^       ^	2 4 3 1 ^	3 4 2 1	4 3 2 1

► Skoðum fyrst þegar  $n = 4$ :

2 1 4 3

4 1 2 3

3 1 4 2

2 3 4 1

2 4 1 3

3 4 1 2

4 3 1 2

3 4 2 1

4 3 2 1

- ▶ Við munum frekar telja hversu margar umraðanir hafa fastapunkt.
- ▶ Þetta er algennt að gera í talningarfræði.
- ▶ Látum nú  $A_j$  tákna mengi þeirra umraðana þar sem  $j$  er fastapunktur.
- ▶ Þá er  $\bigcap_{j \in J} A_j$  mengi þeirra umraðana þar sem allir punktar  $J$  eru fastapunktur.
- ▶ Ef við festum  $k$  punkta í umröðuninni getum við raðað restinni á  $(n - k)!$  marga vegu.
- ▶ Svo  $\left| \bigcap_{j \in J} A_j \right| = (n - |J|)!$ .
- ▶ Takið eftir að seinni stærðin er bara háð fjölda staka í menginu  $J$ .
- ▶ Við vitum einnig að fjöldi hlutmengja  $\{1, \dots, n\}$  með  $k$  stök er  $\binom{n}{k}$ .

- Við fáum loks að fjöldi umraðana með einhvern fastapunkt er

$$\begin{aligned} |A_1 \cup \dots \cup A_n| &= \sum_{\substack{J \subset \{1, \dots, n\} \\ J \neq \emptyset}} (-1)^{|J|-1} \left| \bigcap_{j \in J} A_j \right| \\ &= \sum_{j=1}^n (-1)^{j+1} \binom{n}{j} (n-j)! \\ &= \sum_{j=1}^n (-1)^{j+1} \frac{n!}{(n-j)!j!} (n-j)! \\ &= n! \sum_{j=1}^n \frac{(-1)^{j+1}}{j!}. \end{aligned}$$

- Fjöldi umraðana með engan fastapunkt er því

$$\begin{aligned} n! - n! \sum_{j=1}^n \frac{(-1)^{j+1}}{j!} &= n! \left( 1 + \sum_{j=1}^n \frac{(-1)^j}{j!} \right) \\ &= n! \left( \frac{(-1)^0}{0!} + \sum_{j=1}^n \frac{(-1)^j}{j!} \right) \\ &= n! \sum_{j=0}^n \frac{(-1)^j}{j!}. \end{aligned}$$

- Algengt er að kalla þessa tölu  $!n$ .
- Takið eftir að  $!n/n!$  er  $n$ -ta hlutsumma veldaraðar  $e^x$ , fyrir  $x = -1$ .
- Svo  $!n/n!$  er að stefna á  $e^{-1}$ , þegar  $n$  stefnir á  $\infty$ .

- Oft er hentugt að geta reiknað

$$\binom{n}{k} \bmod m.$$

fyrir jákvæðar heiltölur  $k \leq n < m$ .

- Munið að

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}.$$

- Ein leið til að gera þetta er að finna fyrst margföldunarandhverfur  $(n-k)!$  og  $k!$  með tilliti til  $m$ .
- Við reiknum svo  $n! \cdot ((n-k)!)^{-1} \cdot (k!)^{-1} \bmod m$ .
- Hér þarf að passa að margföldunarandhverfan sé til.
- Helst þarf  $m$  að vera frumtala.

```

26 ll f[MAXN], fm[MAXN];
27 ll prepare_nck(ll m)
28 {
29     ll i;
30     for (i = 0; i < MAXN; i++) f[i] = (i == 0 ? 1 : (f[i - 1]*i)%m);
31     for (i = 0; i < MAXN; i++) fm[i] = mulinv(f[i], m);
32 }
33
34 ll nck(ll n, ll k, ll m)
35 {
36     return (((f[n]*fm[n - k])%m)*fm[k])%m;
37 }

```



- ▶ Ef við erum með fast  $n$  og  $m$  og viljum reikna fyrir  $q$  mismunandi gildi á  $k$  þá er tímaflækjan á þessari að ferð  $\mathcal{O}(n \log m + q)$ .

- Önnur leið til að reikna  $\binom{n}{k}$  mod  $m$  byggir á kvikri bestun.
- Sjáum fyrst að ef  $n > 1$  og  $1 < k < n$  þá

$$\begin{aligned}
 \binom{n-1}{k-1} + \binom{n-1}{k} &= \frac{(n-1)!}{(n-k)!(k-1)!} + \frac{(n-1)!}{(n-k-1)!k!} \\
 &= \frac{k(n-1)!}{(n-k)!k!} + \frac{(n-k)(n-1)!}{(n-k)!k!} \\
 &= \frac{k(n-1)! + (n-k)(n-1)!}{(n-k)!k!} \\
 &= \frac{n(n-1)!}{(n-k)!k!} \\
 &= \frac{n!}{(n-k)!k!} \\
 &= \binom{n}{k}.
 \end{aligned}$$

- ▶ Látum því

$$f(n, k) = \begin{cases} 0, & \text{ef } n < 1 \\ 0, & \text{ef } k < 0 \\ 0, & \text{ef } k > n \\ 1, & \text{ef } k = 0 \text{ eða } k = n \\ f(n-1, k-1) + f(n-1, k) & \text{annars.} \end{cases}$$

- ▶ Við höfum svo að  $f(n, k) = \binom{n}{k}$ .
- ▶ Útfærslan notar ofansækna kvika bestun.

```

5  || d[MAXN][MAXN], m;
6  || dp_lookup(|| x, || y)
7  {
8      if (x < 0 || y < 0 || y > x) return 0;
9      if (d[x][y] != -1) return d[x][y];
10     if (y == 0 || y == x) return 1;
11     return d[x][y] = (dp_lookup(x - 1, y - 1) + dp_lookup(x - 1, y))%m;
12 }

```

- ▶ Sjáum að það eru  $n^2$  stöður.
- ▶ Við reiknum hverja stöðu í  $\mathcal{O}(1)$  tíma, svo við getum svarað  $q$  fyrirspurnum í  $\mathcal{O}(n^2 + q)$  tíma.
- ▶ Það sem forritið okkar er í rauninni að gera er að reikna gildin í þríhyrningin Pascals.
- ▶ Þekkt er að  $k$ -ta talan í  $n$ -tu línu þríhyrnings Pascals er  $\binom{n}{k}$ .

## Þríhyrningur Pascals

				1						
			1		1					
		1		2		1				
	1		3		3		1			
	1	4		6		4		1		
	1	5	10		10		5		1	
	1	6	15	20		15	6		1	
	1	7	21	35	35		21	7		1
1	8	28	56	70	56	28		8		1

## Fjöldi umhverfinga í umröðun

- ▶ Látum  $\sigma$  vera umröðun á  $\{1, \dots, n\}$ .
- ▶ Þá kallast heiltölupar  $(i, j)$  þannig að  $1 \leq i < j \leq n$  og  $\sigma(i) > \sigma(j)$ , *umhverfing* (e. *inversion*) í  $\sigma$ .
- ▶ Látum **a** tákna  $n$  staka lista þannig að  $j$ -ta stak listans sé  $\sigma(j)$ .
- ▶ Svona táknnum við iðulega umraðanir þegar við útfærum þær í tölvu.
- ▶ Gerum nú ráð fyrir að eina leiðin okkar til að breyta **a** er að skipta á aðlægum stökum.
- ▶ Hvað tekur það minnst margar aðgerðir að raða listanum?
- ▶ Það vill svo til að fjöldi umhverfinga í umröðununni er einmitt fjöldi aðgerða sem þarf til að raða listanum.
- ▶ Við getum notað okkur þetta til að finna fjölda umhverfinga.

- ▶ Ein leið til að gera þetta er að færa fyrst minnsta stakið fremst, síðan næst minnsta stakið næst fremst og svo framvegis.
- ▶ Þetta tekur  $\mathcal{O}(n^2)$  tíma.
- ▶ Þetta er því of hægt ef  $n > 10^4$ .
- ▶ Við getum þó notað biltré til að gera þetta hraðara.
- ▶ Tökum eftir að þegar við höfum sett stak á sinn stað getum við hætt að hugsa um það.
- ▶ Við þurfum í raun að geta sagt til um hversu mörg stök í listanum eru fyrir framan það stak sem við viljum færa (við teljum ekki þau stök sem eru komin á sinn stað).
- ▶ Við byrjum því með 1 í hverju staki í biltrénu okkar.
- ▶ Þegar við höfum fært stak á sinn stað setjum við tilheyrandi gildi sem 0.
- ▶ Summa fyrstu  $j - 1$  stakanna í trénu er því fjöldi staka sem  $j$ -ta stakið í listanum þarf að sipta á til að komast á sinn stað.



## Hæga aðferðin

4 1 6 5 7 3 2

Svar: 0

## Hæga aðferðin

4 1 6 5 7 3 2  
~

Svar: 0

## Hæga aðferðin

1 4 6 5 7 3 2

^

Svar: 1

## Hæga aðferðin

1 4 6 5 7 3 2  
                  ^

Svar: 1

## Hæga aðferðin

1 4 6 5 7 2 3  
          ^

Svar: 2

## Hæga aðferðin

1 4 6 5 2 7 3  
          ^

Svar: 3

## Hæga aðferðin

1 4 6 2 5 7 3  
^

Svar: 4

## Hæga aðferðin

1 4 2 6 5 7 3  
    ^

Svar: 5



## Hæga aðferðin

1 2 4 6 5 7 3

^

Svar: 6

## Hæga aðferðin

1 2 4 6 5 7 3  
                  ^

Svar: 6

## Hæga aðferðin

1 2 4 6 5 3 7  
          ^

Svar: 7

## Hæga aðferðin

1 2 4 6 3 5 7  
          ^

Svar: 8

## Hæga aðferðin

1 2 4 3 6 5 7  
^

Svar: 9

## Hæga aðferðin

1 2 3 4 6 5 7  
~

Svar: 10

## Hæga aðferðin

1 2 3 4 6 5 7  
^

Svar: 10

## Hæga aðferðin

1 2 3 4 6 5 7  
          ^

Svar: 10



## Hæga aðferðin

1 2 3 4 5 6 7  
          ^

Svar: 11

## Hæga aðferðin

1 2 3 4 5 6 7  
~

Svar: 11

## Hæga aðferðin

1 2 3 4 5 6 7  
          ^

Svar: 11

## Hæga aðferðin

1 2 3 4 5 6 7

Svar: 11

## Hraða aðferðin

Listinn: 4 1 6 5 7 3 2

Biltréð: 1 1 1 1 1 1 1

Svar: 0

## Hraða aðferðin

Listinn: 4 1 6 5 7 3 2

^

Biltréð: 1 1 1 1 1 1 1

Svar: 0

## Hraða aðferðin

Listinn: 4 1 6 5 7 3 2

^

Biltréð: 1 1 1 1 1 1 1

| |

Svar: 0

## Hraða aðferðin

Listinn: 4 x 6 5 7 3 2

Biltréð: 1 0 1 1 1 1

Svar: 1



## Hraða aðferðin

Listinn: 4 x 6 5 7 3 2  
                                  ^

Biltréð: 1 0 1 1 1 1 1

Svar: 1

# Hraða aðferðin

Listinn: 4 x 6 5 7 3 2

Biltréð: 1 0 1 1 1 1 1

|                      |

Svar: 1

## Hraða aðferðin

Listinn: 4 x 6 5 7 3 x

Biltréð: 1 0 1 1 1 1 0

Svar: 6

## Hraða aðferðin

Listinn: 4 x 6 5 7 3 x

^

Biltréð: 1 0 1 1 1 1 0

Svar: 6

## Hraða aðferðin

Listinn: 4 x 6 5 7 3 x

^

Biltréð: 1 0 1 1 1 1 0

|                    |

Svar: 6

## Hraða aðferðin

Listinn: 4 x 6 5 7 x x

Biltréð: 1 0 1 1 1 0 0

Svar: 10

## Hraða aðferðin

Listinn: 4 x 6 5 7 x x

^

Biltréð: 1 0 1 1 1 0 0

Svar: 10

## Hraða aðferðin

Listinn: 4 x 6 5 7 x x

^

Biltréð: 1 0 1 1 1 0 0

|

Svar: 10



## Hraða aðferðin

Listinn: x x 6 5 7 x x

Biltréð: 0 0 1 1 1 0 0

Svar: 10

## Hraða aðferðin

Listinn: x x 6 5 7 x x

^

Biltréð: 0 0 1 1 1 0 0

Svar: 10

## Hraða aðferðin

Listinn: x x 6 5 7 x x

^

Biltréð: 0 0 1 1 1 0 0

|        |

Svar: 10

## Hraða aðferðin

Listinn: x x 6 x 7 x x

Biltréð: 0 0 1 0 1 0 0

Svar: 11

## Hraða aðferðin

Listinn: x x 6 x 7 x x

^

Biltréð: 0 0 1 0 1 0 0

Svar: 11

## Hraða aðferðin

Listinn: x x 6 x 7 x x

^

Biltréð: 0 0 1 0 1 0 0

|   |

Svar: 11

## Hraða aðferðin

Listinn: x x x x 7 x x

Biltréð: 0 0 0 0 1 0 0

Svar: 11

## Hraða aðferðin

Listinn: x x x x 7 x x

^

Biltréð: 0 0 0 0 1 0 0

Svar: 11



## Hraða aðferðin

Listinn: x x x x 7 x x

^

Biltréð: 0 0 0 0 1 0 0

|            |

Svar: 11

## Hraða aðferðin

Listinn: x x x x x x x

Biltréð: 0 0 0 0 0 0 0

Svar: 11

- ▶ Hvað gerum við þó ef tölurnar sem eru gefnar eru ekki í  $\{1, 2, \dots, n\}$  (líkt og í dæminu *Ultra-QuickSort*)?
- ▶ Þá virkar oft að skipta minnstu tölunni út fyrir 1, næst minnstu fyrir 2 og svo framvegis.
- ▶ Ef það eru endurtekningar þarf að passa að öllum eins tölum sé breytt í eins tölur.
- ▶ Þetta má gera með því að raða.
- ▶ Við röðum fyrst tvenndunum  $(a_j, j)$ , þar sem  $a_j$  táknar  $j$ -ta stakið í listanum okkar, eftir fyrsta stakinu.
- ▶ Við getum þá labbað í gegn og breytt öllum tölunum.
- ▶ Að lokum röðum tvenndunum aftur eftir seinna stakinu.

```

33 typedef struct {ll x, y;} ii;
34 int cmpx(const void* p1, const void* p2)
35 {
36     if (((ii*)p1)->x == ((ii*)p2)->x) return ((ii*)p1)->y - ((ii*)p2)->y;
37     return ((ii*)p1)->x - ((ii*)p2)->x;
38 }
39
40 void compress(ll* a, ll n)
41 {
42     ii b[n];
43     for (ll i = 0; i < n; i++) b[i].x = a[i], b[i].y = i;
44     qsort(b, n, sizeof(b[0]), cmpx);
45     for (ll i = 0; i < n; i++) a[b[i].y] = i;
46 }
47
48 long long teljum_umhverfingar(ll* a, ll n)
49 {
50     ll p[5*n], b[n], i; long long r = 0;
51     compress(a, n);
52     st_init(p, n);
53     for (i = 0; i < n; i++) st_update(p, i, 1);
54     for (i = 0; i < n; i++) b[a[i]] = i;
55     for (i = 0; i < n; i++)
56         r += b[i] != 0 ? st_query(p, 0, b[i] - 1) : 0, st_update(p, b[i], 0);
57     return r;
58 }

```

- ▶ Það tekur  $\mathcal{O}(n \log n)$  tíma að smækka tölurnar.
- ▶ Hver aðgerð á biltrénu er framkvæmd í  $\mathcal{O}(\log n)$  tíma.
- ▶ Við finnum því fjölda umhverfinga í  $\mathcal{O}(n \log n)$  tíma.
- ▶ Svo við getum auðveldlega fundið fjölda umhverfinga fyrir  $n < 10^6$ .
- ▶ Takið þó eftir að fjöldinn gæti orðið  $n \cdot (n - 1)/2$ , svo það þarf alltaf að nota `long long`.

- ▶ Önnur leið til að telja umhverfingar er að deila og drottna.
- ▶ Við getum raðað tölunum með `mergesort` og talið í leiðinni hvað við þurfum að færa tölurnar um mörg sæti.
- ▶ Þegar við erum að sameina erum við með tvo raðaða lista.
- ▶ Við fjarlægjum síðan minnsta stakið sem er í listunum.
- ▶ Ef við fjarlægjum stak úr vinstri listanum færist það um ekkert sæti.
- ▶ Ef við fjarlægjum stak úr hægri listanum færist það um eitt sæti fyrir hvert stak sem er eftir í vinstri listanum.
- ▶ Við getum útfærst þetta neðansækið og ofansækið.

```

5  ll merge(ll* a, ll *b, ll l, ll m, ll r)
6  {
7      ll i = l, j = m, c = i, z = 0;
8      while (i < m || j < r)
9          if (i == m || (j < r && a[j] < a[i])) z += m - i, b[c++] = a[j++];
10         else b[c++] = a[i++];
11     for (i = l; i < r; i++) a[i] = b[i];
12     return z;
13 }
14
15 ll mergesort(ll* a, ll *b, ll l, ll r)
16 {
17     if (r - l < 2) return 0;
18     ll m = (l + r)/2, z = mergesort(a, b, l, m) + mergesort(a, b, m, r);
19     return z + merge(a, b, l, m, r);
20 }
21
22 ll teljum_umhverfingar(ll* a, ll n)
23 {
24     ll b[n];
25     return mergesort(a, b, 0, n);
26 }

```

```

6  ll teljum_umhverfingar(ll* a, ll n)
7  {
8      ll t = 0, z = 0, i, j, c, e, l, m, r, b[2][n];
9      for (i = 0; i < n; i++) b[t][i] = a[i];
10     for (e = 1; e < n; e *= 2, t = 1 - t) for (l = 0; l < n; l += 2*e)
11     {
12         i = c = l, j = m = min(l + e, n), r = min(l + 2*e, n);
13         while (i < m || j < r)
14             if (i == m || (j < r && b[t][j] < b[t][i]))
15                 z += m - i, b[1 - t][c++] = b[t][j++];
16             else b[1 - t][c++] = b[t][i++];
17     }
18     return z;
19 }

```



- ▶ Bæði reikniritin hafa sömu tímaflækju og mergesort, það er að segja  $\mathcal{O}(n \log n)$ .

- ▶ Mörg talningarfræði dæmi má smækka á þægilegan máta.
- ▶ Ef við látum, til dæmis,  $f(n)$  tákna fjölda hlutmengja í mengin  $\{1, 2, \dots, n\}$  þá höfum við að

$$f(n) = \begin{cases} 1, & \text{ef } n = 0 \\ 2 \cdot f(n-1), & \text{annars.} \end{cases}$$

- ▶ Þetta gildir því það eru  $f(n-1)$  hlutmengi sem innihalda  $n$  og  $f(n-1)$  hlutmengi sem innihalda ekki  $n$ .
- ▶ Ef við getum smækkað dæmin á þennan máta má svo nota kvika bestun til að leysa þau.
- ▶ Oft þarf að bæta við vídd til að halda utan um önnur gögn.
- ▶ Tökum dæmi.

- ▶ Gerum ráð fyrir að þú sért með  $n$  spilastokka.
- ▶ Hver stokkur inniheldur  $k$  spil, númeruð frá 1 og upp í  $k$ .
- ▶ Á hversu marga vegu getur þú valið eitt spil úr hverjum stokk þannig að summa spilanna sé nákvæmlega  $m$ ?
- ▶ Þar sem þessi tala getur verið mjög stór svo reikna skal hana mod  $10^9 + 7$ .

- ▶ Festum fyrsta spilið sem  $x$ .
- ▶ Við eigum þá eftir  $n - 1$  stökk og viljum fá summuna  $m - x$  úr þeim.
- ▶ Með öðrum orðum höfum við smækkað dæmið.
- ▶ Við skilgreinum því

$$f(x, y) = \begin{cases} 1, & \text{ef } n = 0 \text{ og } y = 0 \\ 0, & \text{ef } n = 0 \text{ og } y \neq 0 \\ \sum_{j=1}^k f(x-1, y-j), & \text{annars.} \end{cases}$$

- ▶ Nú gildir að  $f(x, y)$  er fjöldi leiða til að fá summuna  $y$  með  $x$  stökkum.
- ▶ Við getum svo útfært þetta eins of við höfum verið að útfæra kvikva bestun.

```

6  || dp_lookup(|| x, || y)
7  {
8      if (y < 0) return 0;
9      if (d[x][y] != -1) return d[x][y];
10     if (x == 0) return y == 0 ? 1 : 0;
11     d[x][y] = 0;
12     for (|| i = 0; i < k; i++)
13         d[x][y] = (d[x][y] + dp_lookup(x - 1, y - i - 1))%MOD;
14     return d[x][y];
15 }

```

- ▶ Við höfum  $n \cdot m$  stöður og hverja stöðu má reikna í  $\mathcal{O}(k)$  tíma.
- ▶ Svo tímaflækjan er  $\mathcal{O}(n \cdot m \cdot k)$ .
- ▶ Nú getur  $m$  ekki verið stærra en  $n \cdot k$  svo við fáum  $\mathcal{O}(n^2 \cdot k^2)$ .

# Flykjaaðgerðir

- ▶ Í stærðfræði þýðir „fylki” annað en í tölvunarfræði.
- ▶ Í stærðfræði er *fylki* (e. *matrix*) tvívíð uppröðun á tölum.
- ▶ Fylkið er sagt vera  $n \times m$  ef það hefur  $n$  línur og  $m$  dálka.
- ▶ Dæmi um  $2 \times 3$  fylki er

$$\begin{pmatrix} 2 & -1 & 0 \\ 0 & 14 & 0 \end{pmatrix}.$$

- ▶ Við táknum yfirleitt stakið í línu  $j$  og dálki  $k$  í fylki  $A$  með  $A_{jk}$ .
- ▶ Flyki í stærfræði er því eins og tvívítt fylki í tölvunarfærði.

- ▶ Þegar við viljum geyma stærðfræði fylki í tölvu notum við oftast tvívítt tölvunarfræði fylki.
- ▶ Við höfum þá að  $A_{jk}$  svarar til `a[j][k]`.
- ▶ Ef  $A$  er  $n \times m$  fylki getum við líka geymt það með einvíðu tölvunarfræði fylki með samsvöruninni  $A_{jk}$  við `a[j*m + k]`.



- ▶ Við getum lagt saman fylki af sömu stærð stakvíst, það er að segja  $(A + B)_{jk} = A_{jk} + B_{jk}$ .
- ▶ Frádráttur virkar eins.
- ▶ Við margföldum saman fylki af stærð  $n \times m$  og  $m \times r$  með

$$(A \cdot B)_{jk} = \sum_{l=1}^m A_{jl} \cdot B_{lk}.$$

- ▶ Oftast erum við að vinna með fylki sem eru af stærð  $n \times n$ .
- ▶ Slík fylki kallast *ferningsfylki*.
- ▶ Takið eftir að ferningsfylkið  $I$ , gefið með  $I_{jk} = 0$  ef  $j \neq k$  og  $I_{jk} = 1$  annars, er margföldunarhlutleysa.

```

4 void addto(int* a, int* b, int n)
5 { // a += b
6     int i, j;
7     for (i = 0; i < n; i++) for (j = 0; j < n; j++) a[i*n + j] += b[i*n + j];
8 }
9
10 void subfrom(int* a, int* b, int n)
11 { // a -= b
12     int i, j;
13     for (i = 0; i < n; i++) for (j = 0; j < n; j++) a[i*n + j] -= b[i*n + j];
14 }
15
16 void multo(int* a, int* b, int n)
17 { // a *= b
18     int i, j, k, c[n][n];
19     for (i = 0; i < n; i++) for (j = 0; j < n; j++) c[i][j] = 0;
20     for (i = 0; i < n; i++) for (j = 0; j < n; j++) for (k = 0; k < n; k++)
21         c[i][j] += a[i*n + k]*b[k*n + j];
22     for (i = 0; i < n; i++) for (j = 0; j < n; j++) a[i*n + j] = c[i][j];
23 }

```

- ▶ Takið eftir að `multo(...)` hefur tímaflækju  $\mathcal{O}(n^3)$ .
- ▶ Ef  $A$  er  $n \times n$  ferningsfylki getum við reinkað  $A^p$ .
- ▶ Með því að nota deila og drottna aðferð, líkt og við gerðum í síðustu viku, getum við reiknað  $A^p$  í  $\mathcal{O}(n^3 \log p)$  tíma.
- ▶ Þetta má nýta mikið í talningarfræði.
- ▶ Tökum dæmi.

- ▶ Látum  $G = (V, E)$  vera net og  $A$  vera nágrannflyki  $G$ .
- ▶ Við munum nú leyfa netinu að hafa fleiri en einn veg á milli tveggja hnúta.
- ▶ Þá segir  $A_{uv}$  hversu margir vegir liggja frá hnúta  $u$  til hnúts  $v$ .
- ▶ Sýna má með þrepuna að  $(A^p)_{uv}$  segir okkur þá hversu margir vegir liggja á milli hnúts  $u$  og hnúts  $v$ , sem eru af lengd nákvæmlega  $p$ .
- ▶ Takið eftir að við getum leyst þetta dæmi fyrir mjög stór  $p$ .
- ▶ Til dæmis væri lausnin okkar leifturhröð fyrir  $n = 50$  og  $p = 10^{18}$ .

# Úrvinnsla línulegra rakningarvensl

- ▶ Runa  $(a_n)_{n \in \mathbb{N}}$  kallast *k-ta stigs línulega rakningarvensl* ef til eru  $c_1, \dots, c_k$  þannig að

$$a_n = \sum_{j=1}^k c_j \cdot a_{n-j},$$

fyrir öll  $n$  þannig að  $n - k \in \mathbb{N}$ .

- ▶ Munið, til dæmis, að Fibonccí tölurnar eru gefnar með línulegu rakningarvenslunum þar sem  $a_1 = a_2 = c_1 = c_2 = 1$ .
- ▶ Látum *k-ta stigs línulega rakningarvensl* vera gefnin, líkt og að ofan.

- Skilgreinum nú flykið

$$M = \begin{pmatrix} c_1 & c_2 & \dots & c_{k-1} & c_k \\ 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 0 \end{pmatrix}.$$

- Takið eftir að

$$M \cdot \begin{pmatrix} a_{n-1} \\ a_{n-2} \\ \vdots \\ a_{n-k} \end{pmatrix} = \begin{pmatrix} \sum_{j=1}^k c_j \cdot a_{n-j} \\ a_{n-1} \\ a_{n-2} \\ \vdots \\ a_{n-k+1} \end{pmatrix} = \begin{pmatrix} a_n \\ a_{n-1} \\ a_{n-2} \\ \vdots \\ a_{n-k+1} \end{pmatrix}.$$

- Svo við getum notað fylkið  $M$  til að fá næstu tölu í rakningarvenslunum.

- ▶ Við þurfum þó ekki að hætta þar.
- ▶ Við getum notað  $M^p$  til að fá  $a_{k+p}$ .
- ▶ Þetta getum við reinkað í  $\mathcal{O}(k^3 \log p)$  tíma.
- ▶ Við getum, til dæmis, notað þetta til að reikna  $n$ -tu Fibonacci töluna í  $\mathcal{O}(\log n)$  tíma.
- ▶ Það er meira en nógu hratt fyrir  $n < 10^{18}$ .

```

14 void matpow(ll* a, ll p, ll n)
15 { // a = a^p
16     ll r[n][n], i, j;
17     for (i = 0; i < n; i++) for (j = 0; j < n; j++) r[i][j] = 0;
18     for (i = 0; i < n; i++) r[i][i] = 1;
19     while (p > 0)
20     {
21         if (p%2 == 1) multo(*r, a, n);
22         p /= 2;
23         multo(a, a, n);
24     }
25     for (i = 0; i < n; i++) for (j = 0; j < n; j++) a[i*n + j] = r[i][j];
26 }
27
28 int fib(ll n)
29 {
30     ll a[2][2] = {{1, 1}, {1, 0}};
31     if (n == 1 || n == 2) return 1;
32     matpow(*a, n - 2, 2);
33     return (a[0][0] + a[0][1])%MOD;
34 }

```



- ▶ Í dæmum sem beita má þessari aðferð er oft erfitt að finna rakningarvenslin (þegar þau eru ekki gefin beint).
- ▶ Þá má nota netafræðina í staðinn.
- ▶ Tökum dæmi.

- ▶ Þú vilt leggja flísar á ganginn þinn þannig að hver flís er annað hvort svört eða hvít, og engar tvær aðliggjandi flísar mega vera hvítar.
- ▶ Gerum ráð fyrir að gangurinn sé þrjár flísar á breidd og  $n$  flísar á lengd.
- ▶ Á hversu marga vegu getur þú lagt flísarnar?
- ▶ Flísar sem snertast horn í horn teljast ekki aðliggjandi.

- ▶ Hægt er að setja þessa talningu fram með fjórða stigs línulegum rakningarvenslum.
- ▶ Það er þó ekki auðséð hverjir stuðlarnir í þessum venslum erum.
- ▶ En hvað ef við breytum þessu í net.
- ▶ Látum hnútana í netinu vera mögulegir dálkar á ganginum:

Hnútur:	0	1	2	3	4	5	6	7
	0	0	0	0	1	1	1	1
Gangur:	0	0	1	1	0	0	1	1
	0	1	0	1	0	1	0	1

- ▶ Við bætum svo við legg á milli hnútanna  $u$  og  $v$  ef dálkarnir mega liggja hliðina á hvorum öðrum á ganginum.
- ▶ Látum 1 tákna hvítar flísar.
- ▶ Takið eftir að stöður 3, 6 og 7 eru alfarið ólöglegar.

- Við fáum þá nágrannafylkið:

	0	1	2	3	4	5	6	7
	+-----							
0	1	1	1	0	1	1	0	0
1	1	0	1	0	1	0	0	0
2	1	1	0	0	1	1	0	0
3	0	0	0	0	0	0	0	0
4	1	1	1	0	0	0	0	0
5	1	0	1	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0

- Köllum þetta fylki  $A$ .
- Við þurfum síðan að leggja saman  $(A^{p-1})_{uv}$  fyrir öll  $u$  og  $v$  þar sem  $u$  er löglegur dálkur.
- Þá eru við að leggja saman alla vegi í netinu af lengd  $p$  sem byrja í löglegum dálki.
- Við þurfum ekki að passa að síðasti hnúturinn sé löglegur því það liggur enginn leggur í ólöglegan hnút.

```

28 int main()
29 {
30     ll i, j, n, r = 0, a[8][8] =
31     {
32         {1, 1, 1, 0, 1, 1, 0, 0},
33         {1, 0, 1, 0, 1, 0, 0, 0},
34         {1, 1, 0, 0, 1, 1, 0, 0},
35         {0, 0, 0, 0, 0, 0, 0, 0},
36         {1, 1, 1, 0, 0, 0, 0, 0},
37         {1, 0, 1, 0, 0, 0, 0, 0},
38         {0, 0, 0, 0, 0, 0, 0, 0},
39         {0, 0, 0, 0, 0, 0, 0, 0}
40     };
41     scanf("%lld", &n);
42     matpow(*a, n - 1, 8);
43     for (i = 0; i < 8; i++) for (j = 0; j < 8; j++)
44         if (i != 3 && i != 6 && i != 7) r = (r + a[i][j])%MOD;
45     printf("%lld\n", r);
46     return 0;
47 }

```

## Gauss-eyðing

- ▶ Umræða um fylkjaaðgerðir er ekki fullkláruð fyrr en minnst er á Gauss-eyðingu.
- ▶ Þið munið eflaust eftir henni úr línulegri algebru.
- ▶ Hún er nytsamlega til að, til dæmis, leysa jöfnuhneppi eða finna andhverfur fylkja.

```

5 int gauss(double* a, int s, int n, int m)
6 {
7     int i, j, k, t, r = 0;
8     for (i = 0; i < n; i++)
9     {
10         for (t = 0; t < s && fabs(a[i*m + t]) < 1e-9; t++);
11         if (t == s) continue;
12         for (r++, j = m - 1; j >= t; j--) a[i*m + j] = a[i*m + j]/a[i*m + t];
13         for (j = 0; j < n; j++) if (i != j) for (k = m - 1; k >= t; k--)
14             a[j*m + k] = a[j*m + k] - a[i*m + k]*a[j*m + t];
15     }
16     return r;
17 }

```

