

Tímaflækjur

Bergur Snorrason

19. janúar 2023

Tímaflækjur

- ▶ Látum $f, g: [0, \infty) \mapsto \mathbb{R}$.
- ▶ Við segjum að fall g sé í menginu $\mathcal{O}(f)$ ef til eru rauntölur $c > 0$ og $x_0 > 0$ þannig að

$$|g(x)| \leq c \cdot |f(x)|$$

fyrir öll $x > x_0$.

- ▶ Þetta þýðir í raun að fallið $|g|$ verður á endanum minna en $c \cdot |f|$.
- ▶ Þessi lýsing undirstrikar að f er efra mat á g , það er að segja g hagar sér ekki verr en f .
- ▶ Ef $g \in \mathcal{O}(f)$ og $f \in \mathcal{O}(g)$ þá segjum við að $f \in \Theta(g)$ (og $g \in \Theta(f)$).

Dæmi

- ▶ Tökum nokkur dæmi.
- ▶ Ef $f \in \mathcal{O}(g)$ og $r > 0$ þá er $r \cdot f \in \mathcal{O}(g)$.
- ▶ Ef $f_1 \in \mathcal{O}(g_1)$ og $f_2 \in \mathcal{O}(g_2)$ þá er $f_1 \cdot f_2 \in \mathcal{O}(g_1 \cdot g_2)$.
- ▶ Dæmin fyrir ofan gilda ef „ \mathcal{O} ” er skipt út fyrir „ Θ ”.
- ▶ Ef p er n -ta stigs margliða þá er $p \in \Theta(x^n)$.
- ▶ Ef p er n -ta stigs margliða og q er m -ta stigs margliða með $n < m$ þá er $p \in \mathcal{O}(q)$, en $q \notin \mathcal{O}(p)$.
- ▶ Við höfum að $\log x \in \mathcal{O}(x)$ en $x \notin \mathcal{O}(\log x)$.
- ▶ Nú er $\log x^n = n \cdot \log x$, svo ef p er n -ta stigs margliða þá er $\log p \in \mathcal{O}(\log x)$.

Í forritun

- ▶ Takið eftir að í stað þessa að segja „ $f \in \mathcal{O}(g)$ ” er oft sagt „ $f = \mathcal{O}(g)$ ” eða „ f er $\mathcal{O}(g)$ ”.
- ▶ Ef forrit framkvæmir $T(n)$ aðgerðir, með $T \in \mathcal{O}(f)$, segjum við að *tímaflækja* (e. *time complexity*) forritsins sé $\mathcal{O}(f)$.
- ▶ Skoðum nú nokkur forrit og ákvörðum tímaflækjur þeirra.

```
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("Hello world!\n");
6     return 0;
7 }
```

- Hér er ekkert inntak svo forritið hefur tímaflækjuna $\mathcal{O}()$.

```
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("Hello world!\n");
6     return 0;
7 }
```

- ▶ Hér er ekkert inntak svo forritið hefur tímaflækjuna $\mathcal{O}(1)$.

```

1 #include <stdio.h>
2
3 int main()
4 {
5     int i, n, r = 0;
6     scanf("%d", &n);
7     for (i = 1; i <= n; i++)
8         r += i;
9     printf("%d\n", r);
10    return 0;
11 }

```

- ▶ Forritið les inn töluna n og reiknar svo summuna $1 + 2 + \dots + n$.
- ▶ Þetta er gert með forlykkju sem keyrir n sinnum.
- ▶ Hvert skipti eru tvær tölur lagðar saman.
- ▶ Svo þetta forrit er $\mathcal{O}(\quad)$.

```

1 #include <stdio.h>
2
3 int main()
4 {
5     int i, n, r = 0;
6     scanf("%d", &n);
7     for (i = 1; i <= n; i++)
8         r += i;
9     printf("%d\n", r);
10    return 0;
11 }

```

- ▶ Forritið les inn töluna n og reiknar svo summuna $1 + 2 + \dots + n$.
- ▶ Þetta er gert með forlykkju sem keyrir n sinnum.
- ▶ Hvert skipti eru tvær tölur lagðar saman.
- ▶ Svo þetta forrit er $\mathcal{O}(n)$.


```

1 #include <stdio.h>
2
3 int main()
4 {
5     int i, j, n, r = 0;
6     scanf("%d", &n);
7     for (i = 0; i < n; i++)
8         for (j = 0; j < i/2; j++)
9             r += i^j;
10    printf("%d\n", r);
11    return 0;
12 }

```

- ▶ Forritið les inn töluna n og reiknar svo summu.
- ▶ Summan er reiknuð með tvöfaldri forlykkju.
- ▶ Ytri forlykkjan keyrir n sinnum og seinni keyrir aldrei oftar en $n/2$ sinnum.
- ▶ Svo tímaflækja forritsins er $\mathcal{O}(\quad)$.

```

1 #include <stdio.h>
2
3 int main()
4 {
5     int i, j, n, r = 0;
6     scanf("%d", &n);
7     for (i = 0; i < n; i++)
8         for (j = 0; j < i/2; j++)
9             r += i^j;
10    printf("%d\n", r);
11    return 0;
12 }

```

- ▶ Forritið les inn töluna n og reiknar svo summu.
- ▶ Summan er reiknuð með tvöfaldri forlykkju.
- ▶ Ytri forlykkjan keyrir n sinnum og seinni keyrir aldrei oftar en $n/2$ sinnum.
- ▶ Svo tímaflækja forritsins er $\mathcal{O}(n^2)$.

```

1 #include <stdio.h>
2
3 int len(int n, int k)
4 {
5     int r = 0;
6     while (n > 0) r++, n /= k;
7     return r;
8 }
9
10 int main()
11 {
12     int i, j, n, r = 0;
13     scanf("%d", &n);
14     for (i = 2; i <= n; i++) r += len(n, i);
15     printf("%d\n", r);
16     return 0;
17 }

```

- ▶ Sjáum fyrst að lykkjan í `main(...)` keyrir $n - 1$ sinnum.
- ▶ Hún kallar síðan á fallið `len(...)`.
- ▶ Það fall hefur eina `while`-lykkju sem deilir tölunni n með k , án afgangs, þar til n er orðin núll.
- ▶ Þetta fallið `len(...)` hefur því tímaflækjuna $\mathcal{O}(\quad)$.
- ▶ Í heildina hefur forritið því tímaflækjuna $\mathcal{O}(\quad)$.

```

1 #include <stdio.h>
2
3 int len(int n, int k)
4 {
5     int r = 0;
6     while (n > 0) r++, n /= k;
7     return r;
8 }
9
10 int main()
11 {
12     int i, j, n, r = 0;
13     scanf("%d", &n);
14     for (i = 2; i <= n; i++) r += len(n, i);
15     printf("%d\n", r);
16     return 0;
17 }

```

- ▶ Sjáum fyrst að lykkjan í `main(...)` keyrir $n - 1$ sinnum.
- ▶ Hún kallar síðan á fallið `len(...)`.
- ▶ Það fall hefur eina `while`-lykkju sem deilir tölunni n með k , án afgangs, þar til n er orðin núll.
- ▶ Þetta fallið `len(...)` hefur því tímaflækjuna $\mathcal{O}(\log n)$.
- ▶ Í heildina hefur forritið því tímaflækjuna $\mathcal{O}(\quad)$.

```

1  #include <stdio.h>
2
3  int len(int n, int k)
4  {
5      int r = 0;
6      while (n > 0) r++, n /= k;
7      return r;
8  }
9
10 int main()
11 {
12     int i, j, n, r = 0;
13     scanf("%d", &n);
14     for (i = 2; i <= n; i++) r += len(n, i);
15     printf("%d\n", r);
16     return 0;
17 }

```

- ▶ Sjáum fyrst að lykkjan í `main(...)` keyrir $n - 1$ sinnum.
- ▶ Hún kallar síðan á fallið `len(...)`.
- ▶ Það fall hefur eina `while`-lykkju sem deilir tölunni n með k , án afgangs, þar til n er orðin núll.
- ▶ Þetta fallið `len(...)` hefur því tímaflækjuna $\mathcal{O}(\log n)$.
- ▶ Í heildina hefur forritið því tímaflækjuna $\mathcal{O}(n \log n)$.

```

1 #include <stdio.h>
2
3 int fib(int n)
4 {
5     if (n < 2) return n;
6     return fib(n - 1) + fib(n - 2);
7 }
8
9 int main()
10 {
11     int n;
12     scanf("%d", &n);
13     printf("%d\n", fib(n));
14     return 0;
15 }

```

- ▶ Hér erum við með endurkvæmt fall.
- ▶ Við sjáum að ef $n \geq 2$ þá kallar `fib(n)` tvisvar á sjálft sig.
- ▶ Svo þetta forrit er $\mathcal{O}(\quad)$.
- ▶ Það er hægt að fá betra mat (við getum minnkað veldisstofninn).

```

1 #include <stdio.h>
2
3 int fib(int n)
4 {
5     if (n < 2) return n;
6     return fib(n - 1) + fib(n - 2);
7 }
8
9 int main()
10 {
11     int n;
12     scanf("%d", &n);
13     printf("%d\n", fib(n));
14     return 0;
15 }

```

- ▶ Hér erum við með endurkvæmt fall.
- ▶ Við sjáum að ef $n \geq 2$ þá kallar `fib(n)` tvisvar á sjálft sig.
- ▶ Svo þetta forrit er $\mathcal{O}(2^n)$.
- ▶ Það er hægt að fá betra mat (við getum minnkað veldisstofninn).

- ▶ Skoðum skiladæmið *Reiknirit*.
- ▶ Fyrsta lína inntaksins inniheldur heiltölu n .
- ▶ Síðan koma n heiltölur a_1, a_2, \dots, a_n .
- ▶ Gerum ráð fyrir að við séum með forrit sem gerir eftirfarandi:
 - ▶ Prentar tölurnar.
 - ▶ Fjarlægir öll eintök af algengustu tölunni í listan.
 - ▶ Endurtekur skrefin að ofan þar til listinn er tómur.
- ▶ Dæmið snýst um að finna hversu margar tölur eru prentaðar í heildina.
- ▶ Tökum dæmi.

- ▶ Gerum ráð fyrir að tölurnar í inntakinu séu 1, 2, 1, 4, 4.
- ▶ Þá myndi forritið í dæminu prenta:

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

- ▶ Svo það prentar 9 tölur.

- ▶ Ein leið til að leysa þetta dæmi er að útfæra forritið.
- ▶ Við þurfum þá að geta fjarlægt algengasta stakið.

```

3 typedef long long ll;
4
5 int cmp(const void *p1, const void *p2)
6 {
7     ll x = *(ll*)p1, y = *(ll*)p2;
8     return (y <= x) - (x <= y);
9 }
10
11 ll fjarlaegja_algengasta_stakid(ll *a, ll n)
12 {
13     ll i, j, mx, r;
14     for (i = j = mx = 0; i < n; i = j)
15     {
16         while (j < n && a[i] == a[j]) j++;
17         if (mx < j - i) mx = j - i, r = a[i];
18     }
19     for (i = 0, j = 0; i < n; i++) if (a[i] != r) a[j++] = a[i];
20     return j;
21 }
22
23 int main()
24 {
25     ll i, n, r = 0;
26     scanf("%lld", &n);
27     ll a[n];
28     for (i = 0; i < n; i++) scanf("%lld", &a[i]);
29     qsort(a, n, sizeof *a, cmp);
30     while (n > 0)
31     {
32         r += n;
33         n = fjarlaegja_algengasta_stakid(a, n);
34     }
35     printf("%lld\n", r);
36     return 0;
37 }

```

- ▶ Þetta forrit leysir dæmið (að því sem ég best veit).
- ▶ Nú er spurning hvort það sé nógu hratt.
- ▶ Í dæminu á Kattis er gefið að $n \leq 10^6$.
- ▶ Reynum nú að ákvarða tímaflækjuna.
- ▶ Sjáum aftur útfærsluna.

```

11 ll fjarlaegja_algengasta_stakid(ll *a, ll n)
12 {
13     ll i, j, mx, r;
14     for (i = j = mx = 0; i < n; i = j)
15     {
16         while (j < n && a[i] == a[j]) j++;
17         if (mx < j - i) mx = j - i, r = a[i];
18     }
19     for (i = 0, j = 0; i < n; i++) if (a[i] != r) a[j++] = a[i];
20     return j;
21 }
22
23 int main()
24 {
25     ll i, n, r = 0;
26     scanf("%lld", &n);
27     ll a[n];
28     for (i = 0; i < n; i++) scanf("%lld", &a[i]);
29     qsort(a, n, sizeof *a, cmp);
30     while (n > 0)
31     {
32         r += n;
33         n = fjarlaegja_algengasta_stakid(a, n);
34     }
35     printf("%lld\n", r);
36     return 0;
37 }

```

- ▶ Í fyrsta lagi röðum við listanum, sem er $\mathcal{O}(n \log n)$.
- ▶ Síðan erum við með `while`-lykkju, sem gerir ekkert annað en að kalla á fallið `fjarlaegja_algengasta_stakid(...)`.
- ▶ Í versta falli fjarlægjum við eitt stak í einu, svo þessi `while`-lykkja keyrir allt að n sinnum.
- ▶ Í fallinu `fjarlaegja_algengasta_stakid(...)` eru tvær lykkjur, hvor um sig keyrir n sinnum.
- ▶ Takið eftir að n er þá lengd listans á þeim tímapunkti.
- ▶ Fallið `fjarlaegja_algengasta_stakid(...)` hefur því tímaflækjuna $\mathcal{O}(n)$.
- ▶ Forritið í heild sinni hefur því tímaflækjuna $\mathcal{O}(n^2)$.

