# Amortizovaná analýza

# Amortizovaná analýza

Provádění algoritmů v sobě často zahrnuje provádění posloupnosti nějakých operací (většinou operací pracujících s nějakou datovou strukturou):

$$op_1, op_2, \cdots, op_n$$

- Může se jednat o operace jednoho typu nebo více různých typů operací (např. přidávání a odebírání prvku do a z dané datové struktury, vyhledávání prvků, apod.)
- Označme $t_i$ čas, který se stráví prováděním operace $op_i$.

  Celkový čas, který se pro konkrétní posloupnost operací $op_1, op_2, \cdots, op_n$ stráví prováděním této posloupnosti operací, je pak

  $$t_1 + t_2 + \cdots + t_n$$

- Chtěli bychom určit tento celkový čas v **nejhorším případě**.

# Amortizovaná analýza

Naivní přístup k určení celkového času, který se stráví prováděním posloupnosti operací

$$op_1, op_2, \cdots, op_n,$$

může vypadat takto:

- Určit, která operace je časově nejnáročnější, a jaká je její časová složitost v nejhorším případě.

  Tato složitost může záviset např. na počtu prvků, které se v okamžiku provádění dané operace již nacházejí v dané datové struktuře.

- Vynásobit tuto složitost celkovým počtem prováděných operací.

To, co dostaneme, bude určitě korektní **horní odhad** celkové doby provádění všech operací.

# Amortizovaná analýza

V některých případech však bude odhad celkové časové složitosti založený na výše uvedeném jednoduchém přístupu příliš pesimistický a odvozená složitost bude mnohem větší než reálná doba, která se stráví prováděním dané posloupnosti operací.

Existuje řada datových struktur, kde:

- některé operace mohou mít v nejhorším případě poměrně velkou časovou složitost (např. $\Theta(n)$, kde $n$ je počet prvků, které se v dané datové struktuře právě nachází)
- velká většina operací proběhne v čase, který je výrazně kratší než ony nejhorší případy (např. $\Theta(1)$)
- i v **nejhorších** případech (tj. v nejhorších možných posloupnostech operací) nastávají „drahé“ (tj. dlouho trvající časově náročné operace) málo často, přičemž předtím než nastanou, musí jim typicky předcházet poměrně velký počet „levnějších“ operací (tj. operací s krátkou dobou výpočtu)

# Amortizovaná analýza

V takových případech je vhodnější, než určovat jen výpočetní složitost jednotlivých operací v nejhorším případě, určovat celkovou časovou složitost celých posloupností operací, kde sčítáme celkový čas jednotlivých operací.

Tento druh analýzy výpočetní složitosti se označuje jako **amortizovaná analýza**:

- Jedná se o analýzu **nejhoršího případu**.
- Dává smysl ji provádět v těch případech, kdy většina operací proběhne v mnohem kratším čase než je nejhorší případ.
- Přestože se zde může počítat, jaká je průměrná doba trvání jedné operace, nejedná se o analýzu složitosti v průměrném případě.
- Protože „drahé“ případy operací budou nastávat jen vzácně, celková doba strávená prováděním těchto operací typicky nebude nijak výrazně větší než doba, která se celkem stráví prováděním „levnějších“ operací.

# Amortizovaná analýza

**Příklad:**

Uvažujme **zásobník**, na kterém budeme provádět následující operace:

- $\textsc{Push}(x)$ — přidá na vrchol zásobníku hodnotu $x$
- $\textsc{Pop}()$ — odebere jeden prvek z vrcholu zásobníku;
  pokud je zásobník prázdný, skončí chybou
- $\textsc{Multipop}(k)$ — odebere z vrcholu zásobníku $k$ prvků;
  pokud zásobník obsahuje méně než $k$ prvků,
  odebere všechny, ale neskočí chybou

Předpokládáme, že máme také k dispozici funkci, která otestuje, zda je zásobník prázdný, ale jeho obsah nemění:

- $\textsc{IsEmpty}()$

# Amortizovaná analýza

Konkrétní implementace operací PUSH, POP a ISEMPTY není podstatná — budeme však předpokládat, že časová složitost těchto elementárních operací je $\Theta(1)$.

Činnost operace MULTIPOP($k$) je možné popsat následujícím pseudokódem:

**Algoritmus:** Operace MULTIPOP

```
MULTIPOP (k):
    while ¬ IsEmpty() and k > 0 do
        POP()
        k := k − 1
```

Vidíme, že časová složitost této operace je $\Theta(\min(s, k))$, kde $s$ je aktuální počet prvků na zásobníku.

# Amortizovaná analýza

Pro jednoduchost budeme při analýze uvažovat následující doby trvání jednotlivých operací:

| Operace | Počet kroků | |
|---|---|---|
| Push($x$) | 1 | |
| Pop() | 1 | |
| Multipop($k$) | $\min(s, k)$ | (kde $s$ je akt. počet prvků) |

Budeme uvažovat libovolné posloupnosti tvořené operacemi těchto tří typů, kde:

- se začíná s prázdným zásobníkem
- operace Pop není nikdy zavolána na prázdný zásobník

Celková doba vykonání dané posloupnosti operací je součet počtu kroků jednotlivých operací.

# Amortizovaná analýza

Je zjevné, že:

- Pokud začneme s prázdným zásobníkem a provedeme libovolnou posloupnost $n$ operací, tak v každém okamžiku během výpočtu bude zásobník obsahovat nejvýše $n$ prvků.

  (Např. pokud provedeme $n$ operací Push, bude zásobník obsahovat přesně $n$ prvků.)

- Pokud aktuálně obsahuje zásobník $n$ prvků a zavoláme operaci

$$\textsc{Multipop}(n)$$

  bude doba trvání této operace $n$ kroků.

Vidíme tedy, že pokud provedeme $n$ operací, provede se $n$ krát nejvýše $n$ kroků, a že tedy celková složitost je v $\mathcal{O}(n^2)$.

*Toto je však příliš pesimistický odhad celkové složitosti!*

# Amortizovaná analýza

Je jasné, že:

- Každý prvek, který je odebírán ze zásobníku (ať už přímým voláním operace Pop nebo v rámci cyklu operace Multipop), musel byl někdy dříve vložen na zásobník operací Push.
- Každý prvek může být odebrán ze zásobníku pouze jednou.
- Celkový počet operací Pop (včetně těch prováděných v rámci operací Multipop) tedy nemůže být větší než počet provedených operací Push.
- Celkový počet kroků provedených v rámci operací Pop a Multipop tedy může být nejvýše takový, jaký je celkový počet provedených operací Push.

# Amortizovaná analýza

V libovolné sekvenci $n$ operací máme nejvýše $n$ operací Push.

Celkem se tedy provede:

- Nejvýše $n$ kroků při provádění operací Push
- Nejvýše $n$ kroků při provádění operací Pop a Multipop.

Dostáváme tedy následující tvrzení:

Pro každou posloupnost délky $n$ tvořenou operacemi Push, Pop a Multipop (v libovolném pořadí) se provede nejvýše $2n$ kroků.

Celková časová složitost vykonání všech operací v této posloupnosti je tedy $\mathcal{O}(n)$.

**Poznámka:** Doba vykonání $n$ operací je také jistě v $\Omega(n)$, protože každá operace vyžaduje provedení alespoň jednoho kroku.

# Amortizovaná analýza

Jestliže se tedy při provádění posloupnosti $n$ operací provede celkem nejvýše $2n$ kroků, tak:

- v průměru trvá provedení jedné operace nejvýše $2$ kroky

Všimněte si, že tato průměrná doba trvání jedné operace:

- je vyšší než počet kroků nutných k provedení jedné operace $\textsc{Push}$ nebo $\textsc{Pop}$ ($1$ krok)
- Je menší než počet kroků nutných k provedení jedné operace $\textsc{Multipop}(k)$ ($\min(s, k)$ kroků)

# Amortizovaná analýza

Výše uvedený postup, kdy sčítáme celkový počet kroků nutných k provedení jednotlivých operací v rámci celé sekvence těchto operací, se označuje jako **agregovaná analýza**.

Jestliže celková doba provedení sekvence $n$ operací je **v nejhorším případě** $T(n)$ (tj. je shora omezená $T(n)$), tak průměrná doba strávená prováděním jedné operace, tj.

$$T(n)/n$$

se označuje jako **amortizovaná cena** jedné operace.

Pro různé operace bude jejich skutečná cena (tj. doba strávená jejich prováděním) větší nebo naopak menší než tato amortizovaná cena.

# Amortizovaná analýza

Důležité je, že součet **amortizovaných cen** jednotlivých operací představuje omezení shora (tj. horní odhad) hodnoty součtu **skutečných cen** jednotlivých operací.

Jedná se tedy o omezení časové složitosti shora, které platí **v nejhorším případě** (tj. pro jakoukoli posloupnost), nikoli o nějaký pravděpodobnostní odhad.

# Amortizovaná analýza

**Příklad:** Vezměme si pole $A$ velikosti $k$ indexované od $0$ (tj. prvky tohoto pole jsou $A[0]$, $A[1]$, ..., $A[k-1]$), kde prvky tohoto pole nabývají pouze hodnot $0$ a $1$:

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 |

Na obsah tohoto pole se můžeme dívat jako na zápis čísla ve dvojkové soustavě.
(Bit $0$, tj. hodnota $A[0]$, je nejméně významný bit.)

Obsah pole $A$ tak reprezentuje číslo

$$\sum_{i=0}^{k-1} A[i] \cdot 2^i$$

# Amortizovaná analýza

Operaci zvětšení čísla o 1 je možné popsat následujícím pseudokódem:

---

**Algoritmus:** Zvýšení hodnoty binárně reprezentovaného čísla o 1

---

```
INCREMENT (A, k):
    i := 0
    while i < k and A[i] = 1 do
        A[i] := 0
        i := i + 1
    if i < k then
        A[i] := 1
```

---

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Operaci zvětšení čísla o 1 je možné popsat následujícím pseudokódem:

**Algoritmus:** Zvýšení hodnoty binárně reprezentovaného čísla o 1

```
INCREMENT (A, k):
    i := 0
    while i < k and A[i] = 1 do
        A[i] := 0
        i := i + 1
    if i < k then
        A[i] := 1
```

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

Operaci zvětšení čísla o 1 je možné popsat následujícím pseudokódem:

**Algoritmus:** Zvýšení hodnoty binárně reprezentovaného čísla o 1

```
INCREMENT (A, k):
    i := 0
    while i < k and A[i] = 1 do
        A[i] := 0
        i := i + 1
    if i < k then
        A[i] := 1
```

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

# Amortizovaná analýza

Operaci zvětšení čísla o 1 je možné popsat následujícím pseudokódem:

**Algoritmus:** Zvýšení hodnoty binárně reprezentovaného čísla o 1

```
INCREMENT (A, k):
    i := 0
    while i < k and A[i] = 1 do
        A[i] := 0
        i := i + 1
    if i < k then
        A[i] := 1
```

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

Operaci zvětšení čísla o 1 je možné popsat následujícím pseudokódem:

**Algoritmus:** Zvýšení hodnoty binárně reprezentovaného čísla o 1

```
INCREMENT (A, k):
    i := 0
    while i < k and A[i] = 1 do
        A[i] := 0
        i := i + 1
    if i < k then
        A[i] := 1
```

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

Operaci zvětšení čísla o 1 je možné popsat následujícím pseudokódem:

**Algoritmus:** Zvýšení hodnoty binárně reprezentovaného čísla o 1

```
INCREMENT (A, k):
    i := 0
    while i < k and A[i] = 1 do
        A[i] := 0
        i := i + 1
    if i < k then
        A[i] := 1
```

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |

Operaci zvětšení čísla o $1$ je možné popsat následujícím pseudokódem:

**Algoritmus:** Zvýšení hodnoty binárně reprezentovaného čísla o $1$

INCREMENT $(A, k)$:

- $i := 0$
- **while** $i < k$ **and** $A[i] = 1$ **do**
  - $A[i] := 0$
  - $i := i + 1$
- **if** $i < k$ **then**
  - $A[i] := 1$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |

Operaci zvětšení čísla o 1 je možné popsat následujícím pseudokódem:

**Algoritmus:** Zvýšení hodnoty binárně reprezentovaného čísla o 1

```
INCREMENT (A, k):
    i := 0
    while i < k and A[i] = 1 do
        A[i] := 0
        i := i + 1
    if i < k then
        A[i] := 1
```

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |

# Amortizovaná analýza

Operaci zvětšení čísla o 1 je možné popsat následujícím pseudokódem:

**Algoritmus:** Zvýšení hodnoty binárně reprezentovaného čísla o 1

```
INCREMENT (A, k):
    i := 0
    while i < k and A[i] = 1 do
        A[i] := 0
        i := i + 1
    if i < k then
        A[i] := 1
```

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |

# Amortizovaná analýza

Operaci zvětšení čísla o 1 je možné popsat následujícím pseudokódem:

**Algoritmus:** Zvýšení hodnoty binárně reprezentovaného čísla o 1

```
INCREMENT (A, k):
    i := 0
    while i < k and A[i] = 1 do
        A[i] := 0
        i := i + 1
    if i < k then
        A[i] := 1
```

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |

# Amortizovaná analýza

Operaci zvětšení čísla o 1 je možné popsat následujícím pseudokódem:

**Algoritmus:** Zvýšení hodnoty binárně reprezentovaného čísla o 1

```
INCREMENT (A, k):
    i := 0
    while i < k and A[i] = 1 do
        A[i] := 0
        i := i + 1
    if i < k then
        A[i] := 1
```

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |

Operaci zvětšení čísla o 1 je možné popsat následujícím pseudokódem:

**Algoritmus:** Zvýšení hodnoty binárně reprezentovaného čísla o 1

```
INCREMENT (A, k):
    i := 0
    while i < k and A[i] = 1 do
        A[i] := 0
        i := i + 1
    if i < k then
        A[i] := 1
```

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |

# Amortizovaná analýza

Operaci zvětšení čísla o 1 je možné popsat následujícím pseudokódem:

**Algoritmus:** Zvýšení hodnoty binárně reprezentovaného čísla o 1

```
INCREMENT (A, k):
    i := 0
    while i < k and A[i] = 1 do
        A[i] := 0
        i := i + 1
    if i < k then
        A[i] := 1
```

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |

Operaci zvětšení čísla o $1$ je možné popsat následujícím pseudokódem:

**Algoritmus:** Zvýšení hodnoty binárně reprezentovaného čísla o $1$

```
INCREMENT (A, k):
    i := 0
    while i < k and A[i] = 1 do
        A[i] := 0
        i := i + 1
    if i < k then
        A[i] := 1
```

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |

# Amortizovaná analýza

Operaci zvětšení čísla o 1 je možné popsat následujícím pseudokódem:

**Algoritmus:** Zvýšení hodnoty binárně reprezentovaného čísla o 1

```
INCREMENT (A, k):
    i := 0
    while i < k and A[i] = 1 do
        A[i] := 0
        i := i + 1
    if i < k then
        A[i] := 1
```

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |

# Amortizovaná analýza

Operaci zvětšení čísla o 1 je možné popsat následujícím pseudokódem:

**Algoritmus:** Zvýšení hodnoty binárně reprezentovaného čísla o 1

```
INCREMENT (A, k):
    i := 0
    while i < k and A[i] = 1 do
        A[i] := 0
        i := i + 1
    if i < k then
        A[i] := 1
```

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

# Amortizovaná analýza

Operaci zvětšení čísla o 1 je možné popsat následujícím pseudokódem:

**Algoritmus:** Zvýšení hodnoty binárně reprezentovaného čísla o 1

Increment $(A, k)$:

$i := 0$
**while** $i < k$ **and** $A[i] = 1$ **do**
    $A[i] := 0$
    $i := i + 1$
**if** $i < k$ **then**
    $A[i] := 1$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

# Amortizovaná analýza

Zvětšení čísla o $1$ tedy spočívá v tom, že:

- všechny nejméně významné bity s hodnotou $1$ se změní na $0$
- nejméně významný bit s hodnotou $0$ se změní na $1$

Pokud budeme počítat změnu jednoho bitu jako jeden krok, tak

- jedno provedení operace INCREMENT vyžaduje $\ell + 1$ kroků,

kde $\ell$ označuje počet jedniček na konci zápisu daného čísla.

Pokud tedy čísla reprezentujeme $k$ bity (tj. $A$ má délku $k$), tak při provedení jedné operace INCREMENT může být provedeno až $k$ kroků.

# Amortizovaná analýza

Cílem je spočítat, kolik se celkem provede kroků, pokud:

- začneme s číslem $0$ (tj. s polem $A$ délky $k$ obsahujícím samé nuly)
- $n$ krát zavoláme operaci INCREMENT.

Označme tento celkový počet kroků $T(n)$.

Protože každé provedení operace INCREMENT vyžaduje nejvýše $k$ kroků, je zřejmé, že

$$T(n) \leq k \cdot n$$

a tedy celková doba výpočtu je $\mathcal{O}(k \cdot n)$.

*Toto je opět příliš pesimistický odhad!*

# Amortizovaná analýza

| | |
|---|---|
| **0** | 0 0 0 0 0 0 0 0 |
| **1** | 0 0 0 0 0 0 0 1 |
| **2** | 0 0 0 0 0 0 1 0 |
| **3** | 0 0 0 0 0 0 1 1 |
| **4** | 0 0 0 0 0 1 0 0 |
| **5** | 0 0 0 0 0 1 0 1 |
| **6** | 0 0 0 0 0 1 1 0 |
| **7** | 0 0 0 0 0 1 1 1 |
| **8** | 0 0 0 0 1 0 0 0 |
| **9** | 0 0 0 0 1 0 0 1 |
| **10** | 0 0 0 0 1 0 1 0 |
| **11** | 0 0 0 0 1 0 1 1 |
| **12** | 0 0 0 0 1 1 0 0 |
| **13** | 0 0 0 0 1 1 0 1 |
| **14** | 0 0 0 0 1 1 1 0 |
| **15** | 0 0 0 0 1 1 1 1 |
| **16** | 0 0 0 1 0 0 0 0 |

# Amortizovaná analýza

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| **2** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| **3** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| **4** | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| **5** | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| **6** | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| **7** | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| **8** | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| **9** | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| **10** | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| **11** | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| **12** | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| **13** | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| **14** | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| **15** | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| **16** | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

# Amortizovaná analýza

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| **2** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 3 |
| **3** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 4 |
| **4** | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 7 |
| **5** | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 8 |
| **6** | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 10 |
| **7** | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 11 |
| **8** | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 15 |
| **9** | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 16 |
| **10** | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 18 |
| **11** | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 19 |
| **12** | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 22 |
| **13** | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 23 |
| **14** | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 25 |
| **15** | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 26 |
| **16** | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 31 |

# Amortizovaná analýza

Můžeme provést **agregovanou analýzu**.

Je zřejmé, že:

- bit $A[0]$ se mění při každém zavolání operace INCREMENT (tj. celkem $n$ krát)
- bit $A[1]$ se mění při každém druhém zavolání operace INCREMENT (tj. celkem $\lfloor n/2 \rfloor$ krát)
- bit $A[2]$ se mění při každém čtvrtém zavolání operace INCREMENT (tj. celkem $\lfloor n/4 \rfloor$ krát)
- bit $A[3]$ se mění při každém osmém zavolání operace INCREMENT (tj. celkem $\lfloor n/8 \rfloor$ krát)
- $\ldots$

# Amortizovaná analýza

Protože počet kroků přesně odpovídá tomu, kolikrát se celkem změní hodnoty jednotlivých bitů, dostáváme celkový počet kroků

$$T(n) = \sum_{i=0}^{k-1} \left\lfloor \frac{n}{2^i} \right\rfloor$$

Zjevně platí

$$\sum_{i=0}^{k-1} \left\lfloor \frac{n}{2^i} \right\rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$$

Celkový počet kroků $T(n)$ je v $\mathcal{O}(n)$.

(Vidíme tedy, že s každým zavoláním operace INCREMENT se změní v průměru nejvýše dva bity.)

# Amortizovaná analýza

Pro provedení **amortizované analýzy** se kromě **agregované analýzy**, kdy se počítají celkové počty provedených instrukcí, používají, zejména ve složitějších případech, také následující dva přístupy:

- **účetní metoda** (**accounting method**) — někdy se používá pro označení tohoto přístupu též pojem **bankéřův pohled** na amortizaci (**banker's view** of amortization)
- **metoda založená na potenciálech** (**the potential method**) — díváme se na analýzu pohledem fyzika

# Amortizovaná analýza — účetní metoda

**Účetní metoda** pracuje s následující představou:

- Algoritmus je vykonáván strojem, do kterého musíme házet mince, aby běžel.

  Po vhození jedné mince vykoná vždy jen nějaké omezené množství práce — typicky nějaký výpočet, který se dá provést v čase $\mathcal{O}(1)$.

- Za každou operaci platíme nějakým počtem mincí, který může záviset na typu dané operace.

  Částka (tj. počet mincí), kterou platíme za danou operaci, se označuje jako **amortizovaná cena** dané operace.

- Amortizovaná cena operace nemusí odpovídat skutečné časové náročnosti operace — za některé operace platíme více a za některé naopak méně, než kolik mincí by bylo potřeba k jejich vykonání.

# Amortizovaná analýza — účetní metoda

- Pokud za nějakou operaci platíme více, než kolik je potřeba pro její vykonání, na její vykonání se použijí jen některé mince.

  Zbylé mince se uloží do dané datové struktury — většinou se příslušné jednotlivé mince přiřadí nějakým konkrétním částem dané datové struktury (např. konkrétní položce pole, konkrétnímu vrcholu stromu, apod.)

- Pokud částka zaplacená za nějakou operaci nestačí na její plné vykonání (tj. pokud platíme za danou operaci menší počet mincí než je potřeba na její vykonání), musíme na zaplacení chybějící částky použít mince uložené z předchozích operací v dané datové struktuře.

  Tyto mince tím ze struktury odebereme.

  Nikdy nesmí dojít k tomu, že bychom neměli dostatek mincí na dokončení dané operace!

# Amortizovaná analýza — účetní metoda

**Poznámka:** Ve skutečnosti se do dané datové struktury samozřejmě žádné mince neukládají (ani žádná data, která by je reprezentovala).

Jedná se čistě o myšlenkový konstrukt použitý při analýze daného algoritmu. Není to nic, co by bylo přímo součástí implementace.

# Amortizovaná analýza — účetní metoda

Pro posloupnost $n$ operací

$$op_1,\ op_2,\ \cdots,\ op_n$$

označme

$$t_1,\ t_2,\ \cdots,\ t_n$$

**skutečné doby trvání** jednotlivých operací (tj. $t_i$ je **skutečná cena** operace $op_i$, tedy počet kroků nutných skutečně nutný k jejímu vykonání).

Zápisem

$$a_1,\ a_2,\ \cdots,\ a_n$$

označme **amortizované ceny** jednotlivých operací, tj. $a_i$ je počet mincí, které musíme zaplatit za provedení operace $op_i$.

# Amortizovaná analýza — účetní metoda

Analýza s použitím účetní metody vypadá typicky tak, že:

- Musíme určit jaké jsou skutečné a amortizované ceny jednotlivých operací.

  Skutečné ceny (tj. $t_i$) jsou jednoznačně dány — závisí pouze na tom, jak konkrétně jsou tyto operace implementovány a jaká je časová složitost této implementace.

  Amortizované ceny (tj. $a_i$) si můžeme libovolně sami zvolit.
  Je ale samozřejmě třeba si je zvolit tak, aby analýza pro tuto volbu „fungovala".

- Musíme ukázat, že v každém okamžiku bude vždy dostatečný počet mincí na provedení dané operace.

# Amortizovaná analýza — účetní metoda

Musí tedy platit nejen to, že zaplacené amortizované ceny stačí na provedení všech $n$ operací

$$\sum_{i=1}^{n} t_i \leq \sum_{i=1}^{n} a_i$$

ale totéž musí platit i pro jakýkoli prefix těchto $n$ operací, tj. pro každé $k$, kde $0 \leq k \leq n$, musí platit

$$\sum_{i=1}^{k} t_i \leq \sum_{i=1}^{k} a_i$$

Součet amortizovaných cen tak představuje omezení shora na celkovou dobu výpočtu $T(n)$.

# Amortizovaná analýza — účetní metoda

**Příklad:** Vezměme opět stejný příklad se **zásobníkem**, který jsme uvažovali dříve.

Amortizované ceny jednotlivých operací můžeme zvolit následovně:

| Operace | Skutečná cena | Amortizovaná cena |
|---|---|---|
| PUSH | 1 | 2 |
| POP | 1 | 0 |
| MULTIPOP | $\min(s, k)$ | 0 |

- Při operaci PUSH vždy zaplatíme 2 mince.
  Jednu použijeme na zaplacení vykonání této operace.
  Druhou uložíme spolu s ukládaným prvkem na zásobník.
- Při operacích POP a MULTIPOP platíme tyto operace mincemi uloženými u jednotlivých prvků na zásobníku, které odebíráme.

# Amortizovaná analýza — účetní metoda

- V každém okamžiku platí, že každý prvek na zásobníku obsahuje právě jednu minci.

Je zjevné, že celková částka, kterou zaplatíme při provedení $n$ operací, je $2k$, kde $k$ je počet operací $\textsc{Push}$.

Protože $k \leq n$, očividně je celkový počet kroků nejvýše $2n$ a doba výpočtu je tedy v $\mathcal{O}(n)$.

# Amortizovaná analýza — účetní metoda

**Příklad:** Vezměme si nyní dřívější příklad s **binárním čítačem** s $k$ bity, který inkrementujeme pomocí operace INCREMENT.

- Skutečná cena jednoho volání operace INCREMENT je $\ell + 1$, kde $\ell$ je počet jedniček na nejméně významných pozicích v zápisu hodnoty čítače.
- Jako amortizovanou cenu jednoho provedení operace INCREMENT zvolme $2$ mince.
  - Jednu minci použijeme na změnu bitu s hodnotou $0$ na hodnotu $1$.
  - Druhou minci položíme na tento bit (na bit, jehož hodnota se právě změnila z $0$ na $1$.
- V každém okamžiku tak budou ležet mince na těch bitech, které mají momentálně hodnotu $1$.

# Amortizovaná analýza — účetní metoda

- Kroky, při kterých se mění hodnota daného bitu z $1$ na $0$, platíme mincemi, které leží na těchto bitech.

Je zjevné, že:

- Celkově za provedení $n$ operací INCREMENT zaplatíme částku $2n$.

Celková časová složitost provedení $n$ operací INCREMENT je tak $\mathcal{O}(n)$.

# Amortizovaná analýza — metoda založená na potenciálech

**Metoda založená na potenciálech** je v některých ohledech podobná jako účetní metoda, ale je založena na trochu odlišné představě.

- Při provádění sekvence operací

  $$op_1,\ op_2,\ \cdots,\ op_n$$

  prochází daná datová struktura postupně posloupností **stavů**

  $$D_0,\ D_1,\ D_2,\ \cdots,\ D_n$$

  kde $D_i$ představuje celkový stav dané datové struktury (např. obsah všech jejích prvků a všech dalších pomocných dat v této struktuře) po provedení prvních $i$ operací.

  Struktura tedy začíná ve stavu $D_0$ a končí ve stavu $D_n$.

  (Stav $D_i$ je tedy něco jako konfigurace, ale omezená jen na danou strukturu.)

# Amortizovaná analýza — metoda založená na potenciálech

Na rozdíl od účetní metody nezačneme tím, že bychom definovali amortizovanou cenu jednotlivých operací.

Místo toho definujeme funkci $\Phi$, která každému stavu $D$ přiřazuje reálné číslo $\Phi(D)$, které udává hodnotu **potenciálu** daného stavu $D$.

- Struktura začíná s hodnotou potenciálu $\Phi(D_0)$.
- Různé operace změnou stavu dané struktury mění její potenciál.

  Jeho hodnota se může provedením některých operací zvýšit a provedením jiných zase snížit.

- Je třeba ukázat, že hodnota potenciálu nikdy neklesne pod počáteční hodnotu potenciálu, tj. pro každé $i$, kde $0 \leq i \leq n$ platí

  $$\Phi(D_0) \leq \Phi(D_i)$$

  Většinou se volí funkce $\Phi$ tak, aby platilo $\Phi(D_0) = 0$.

  V tom případě stačí ukázat, že hodnota $\Phi(D)$ je vždy nezáporná.

# Amortizovaná analýza — metoda založená na potenciálech

Poté, co jsme zvolili funkci $\Phi$ udávající hodnotu potenciálu, stanovíme **amortizovanou cenu** operace $op_i$, označenou zápisem $a_i$, jako součet:

- **skutečné ceny** $t_i$ operace $op_i$ a
- **změny potenciálu** způsobené provedením operace $op_i$

Tedy pro každé $i$, kde $1 \leq i \leq n$, platí:

$$a_i = t_i + \Phi(D_i) - \Phi(D_{i-1})$$

- Amortizovaná cena dané operace je tedy vyšší než její skutečná cena, pokud tato operace vede ke zvýšení hodnoty potenciálu.
- Amortizovaná cena dané operace je nižší než její skutečná cena, pokud tato operace vede ke snížení hodnoty potenciálu.

# Amortizovaná analýza — metoda založená na potenciálech

Když sečteme amortizované ceny jednotlivých operací, hodnoty všech potenciálů, s výjimkou počátečního a koncového, se vzájemně vyruší:

$$\begin{aligned} \sum_{i=1}^{n} a_i &= \sum_{i=1}^{n} (t_i + \Phi(D_i) - \Phi(D_{i-1})) \\ &= \Phi(D_n) - \Phi(D_0) + \sum_{i=1}^{n} t_i \end{aligned}$$

Protože $\Phi(D_0) \leq \Phi(D_n)$, zjevně platí

$$\sum_{i=1}^{n} t_i \leq \sum_{i=1}^{n} a_i$$

a součet amortizovaných cen operací tak shora omezuje celkovou časovou složitost $T(n)$.

# Amortizovaná analýza — metoda založená na potenciálech

**Příklad:** Uvažujme opět dřívější příklad se **zásobníkem**.

Stav zásobníku $D$ je zde dán obsahem zásobníku v dané chvíli.

- Jako hodnotu potenciálu $\Phi(D)$ zde můžeme zvolit počet prvků v zásobníku $D$.
- Stav $D_0$ je prázdný zásobník, takže $\Phi(D_0) = 0$.
- Počet prvků v zásobníku není nikdy záporný, takže pro každé $D$ platí $\Phi(D) \geq 0$.

**Amortizované ceny** jednotlivých operací stanovíme následovně:

- Operace Push zvyšuje potenciál o $1$. Platíme tedy $1$ za provedení této operace a $1$ za zvýšení potenciálu.

  Amortizovaná cena operace Push je proto $2$.

# Amortizovaná analýza — metoda založená na potenciálech

- Operace Pop a Multipop snižují potenciál o počet odebraných prvků.

  Toto snížení potenciálu plně pokryje skutečnou cenu provedení těchto operací.
  Za tyto operace tedy neplatíme nic a amortizovaná cena těchto operací je $0$.

Stejně jako v předchozích případech tedy vidíme, že součet amortizovaných cen všech $n$ operací je nejvýše $2n$.

Celková časová složitost je tedy $\mathcal{O}(n)$.

# Amortizovaná analýza — metoda založená na potenciálech

**Příklad:** Vraťme se k příkladu s **binárním čítačem**.

Stav $D$ je zde dán obsahem pole $A$, tj. hodnotami jednotlivých bitů čítače. Tento stav můžeme ztotožnit s číselnou hodnotou čítače reprezentovanou stavem $D$.

- Hodnotu potenciálu $\Phi(D)$ zde můžeme definovat jako počet jedniček v zápise čísla reprezentovaného stavem $D$.
- Amortizovaná cena jednoho provedení operace INCREMENT je $2$:
  - $1$ jednotka se použije na změnu jednoho bitu z $0$ na $1$
  - $1$ jednotka se použije na zvýšení potenciálu o $1$ způsobeného touto změnou
  - Náklady na všechny změny hodnot bitů z $1$ na $0$ se pokryjí snížením potenciálu o příslušnou hodnotu danou počtem těchto bitů.

# Amortizovaná analýza — metoda založená na potenciálech

Opět vidíme, že součet amortizovaných cen za posloupnost $n$ operací INCREMENT je $2n$, a že celková složitost je tak $\mathcal{O}(n)$.

# Amortizovaná analýza — metoda založená na potenciálech

Zatím jsme předpokládali, že na začátku daný $k$-bitový čítač obsahuje hodnotu $0$, tj. že všechny prvky pole $A$ na začátku obsahují hodnotu $0$.

Za tohoto předpokladu platí $\Phi(D_0) = 0$.

Uvažujme nyní obecnější případ, kdy počáteční hodnota čítače může být libovolná.

Rovněž předpokládejme, že $k$ nemusí být nutně konstanta, ale nějaká funkce závislá na dalších hodnotách.

Zjevně pro jakýkoli stav $D$ platí $0 \leq \Phi(D) \leq k$.

Z dříve odvozených vztahů plyne

$$\sum_{i=1}^{n} t_i \; = \; \Phi(D_0) - \Phi(D_n) + \sum_{i=1}^{n} a_i$$

# Amortizovaná analýza — metoda založená na potenciálech

Protože rozdíl $\Phi(D_0) - \Phi(D_n)$ je nejvýše $k$ (a nejméně $-k$), platí tak

$$T(n) = \sum_{i=1}^{n} t_i \in \mathcal{O}(k) + \mathcal{O}(n)$$

Speciálně v případě, kdy bude navíc platit, že $k \in \mathcal{O}(n)$, tak bude $T(n) \in \mathcal{O}(n)$ bez ohledu na počáteční hodnotu čítače.

# Amortizovaná analýza

Dva zde uvedené příklady byly velmi jednoduché:

- zásobník s operací MULTIPOP
- binární čítač s operací INCREMENT

V praxi se však výše uvedené metody (agregovaná analýza, účetní metoda a metoda založená na potenciálech) používají k analýze podstatně složitějších algoritmů a datových struktur.

Stručně zde pro ilustraci zmíníme dva velmi známé typické příklady datových struktur, při jejichž analýze jsou tyto metody využity:

- **splay stromy** (**splay trees**)
- **union-find**

(Detailní popisy a analýzy těchto struktur jsou poměrně složité a technicky komplikované, proto si je zde nebudeme podrobně uvádět.)

# Amortizovaná analýza — splay stromy

**Splay stromy** (**splay trees**):

- Jedná se o speciální případ binárních vyhledávacích stromů.
- Nejedná se o vyvažované stromy — v nejhorším případě mohou mít všechny operace (vložení prvku, odstranění prvku, vyhledání prvku) složitost $\mathcal{O}(n)$.
- Podobně jako u vyvažovaných stromů se strom u jednotlivých operací dále upravuje — týká se to nejen operací vložení a odstranění prvku, ale i operace hledání.
- Amortizovaná složitost všech operací je podobná jako u vyvažovaných stromů — $\mathcal{O}(\log n)$.

  Jakákoli posloupnost operací tak v součtu bude trvat podobnou dobu jako by trvala v případě vyvažovaných stromů.

# Amortizovaná analýza — splay stromy

- Na rozdíl od vyvažovaných stromů jsou u splay stromů prvky, které jsou vyhledávány, postupně přesunovány blíže ke kořeni.

  Důsledkem je, že se prvky, které jsou často vyhledávány, dostanou na místa blízko ke kořeni a jejich vyhledání bude trvat kratší dobu.

  Datová struktura se tak dynamicky přizpůsobuje tomu, jaké prvky jsou v ní často vyhledávány.

# Amortizovaná analýza — Union-Find

Chceme udržovat informace o prvcích $x_1, x_2, \ldots, x_n$, které jsou rozděleny do disjunktních množin $S_1, S_2, \ldots, S_k$.

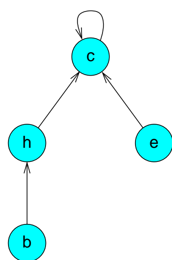
Každá množina má jednoho reprezentanta.

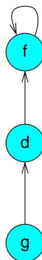
Chceme provádět následující operace:

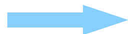
- $\textsc{Make-Set}(x)$ – vytvoření nové množiny obsahující pouze prvek $x$ ($x$ nesmí být prvkem žádné již vytvořené množiny).
- $\textsc{Union}(x, y)$ – sjednocení množin obsahujících prvky $x$ a $y$.
- $\textsc{Find-Set}(x)$ – nalezení reprezentanta množiny obsahující prvek $x$.

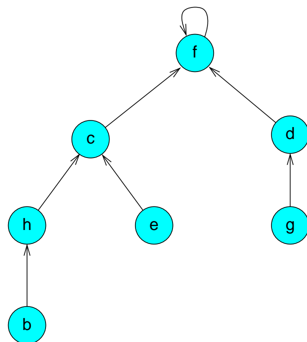
# Amortizovaná analýza — Union-Find

Jednotlivé množiny mohou být reprezentovány jako stromy, přičemž kořen stromu je reprezentantem dané množiny:

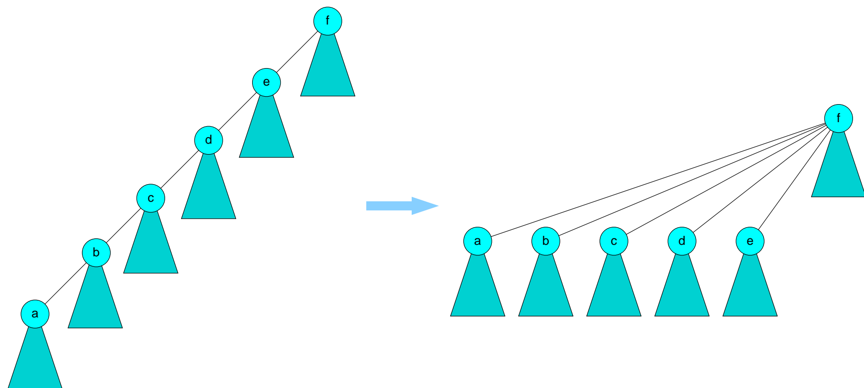




# Amortizovaná analýza — Union-Find



# Amortizovaná analýza — Union-Find

**Algoritmus:** Vytvoření jednoprvkové množiny

```
MAKE-SET (x):
    p[x] := x
    rank[x] := 0
```

**Algoritmus:** Vyhledání množiny, do které prvek patří

```
FIND-SET (x):
    if x ≠ p[x] then
        p[x] := FIND-SET(p[x])
    return p[x]
```

**Algoritmus:** Spojení dvou množin do jedné

```
UNION (x, y):
    v := FIND-SET(x)
    w := FIND-SET(y)
    if rank[v] > rank[w] then
        p[w] := v
    else
        p[v] := w
        if rank[v] = rank[w] then
            rank[w] := rank[w] + 1
```

# Amortizovaná analýza — Union-Find

**Tvrzení**

Sekvence $m$ operací Make-Set, Union a Find-Set, z nichž $n$ operací je Make-Set, vyžaduje v nejhorším případě čas $\mathcal{O}(m\,\alpha(n))$.

**Poznámka:** $\alpha(n)$ je extrémně pomalu rostoucí funkce, která se pro jakékoliv „rozumné“ hodnoty $n$ chová podobně jako konstantní funkce (například pro $n < 10^{80}$ je $\alpha(n) \leq 4$).

Přesná definice funkce $\alpha(n)$: $\alpha(n) = \min\{k : A_k(1) \geq n\}$

$$A_k(j) = \begin{cases} j+1 & \text{pokud } k = 0 \\ A_{k-1}^{(j+1)}(j) & \text{pokud } k \geq 1 \end{cases}$$

kde $A_k^{(0)}(j) = j$ a $A_k^{(i+1)}(j) = A_k(A_k^{(i)}(j))$.