Stanu se rychlejším a ještě rychlejším, až budu nejrychlejším na celém světě

Radim Bača

Katedra informatiky
Fakulta elektrotechniky a informatiky
VŠB – Technická univerzita Ostrava

Stanu se rychlejším a ještě rychlejším, až budu nejrychlejším na celém světě

Radim Bača

Katedra informatiky
Fakulta elektrotechniky a informatiky
VŠB – Technická univerzita Ostrava

Obsah

- Reprezentace čísel a kódování
- Algoritmy Fibonacciho kódování a dekódování
- Motivace
- Algoritmus rychlého Fibonacciho dekódování
- Testy

Reprezentace čísel

- Binární kód
- $83_{10} \Rightarrow 1010011_2$
- Pokud ukládáme pole čísel, pak je obvykle velikost paměti vyhrazené pro každé číslo totožná.
- U celých čísel to obvykle bývají 4 byty $\Rightarrow 2^{32} = 4\ 294\ 967\ 296$

Reprezentace čísel

- Binární kód
- $83_{10} \Rightarrow 1010011_2$
- Pokud ukládáme pole čísel, pak je obvykle velikost paměti vyhrazené pro každé číslo totožná.
- U celých čísel to obvykle bývají 4 byty $\Rightarrow 2^{32} = 4\ 294\ 967\ 296$

Reprezentace čísel

- Binární kód
- $83_{10} \Rightarrow 1010011_2$
- Pokud ukládáme pole čísel, pak je obvykle velikost paměti vyhrazené pro každé číslo totožná.
- U celých čísel to obvykle bývají 4 byty $\Rightarrow 2^{32} = 4\ 294\ 967\ 296$

Reprezentace čísel

- Binární kód
- $83_{10} \Rightarrow 1010011_2$
- Pokud ukládáme pole čísel, pak je obvykle velikost paměti vyhrazené pro každé číslo totožná.
- U celých čísel to obvykle bývají 4 byty $\Rightarrow 2^{32} = 4\ 294\ 967\ 296$

Kódování

- V určitých případech však můžeme ukládat pole čísel, kde má většina malou hodnotu, a chceme takovéto pole „zmenšit“.
- Toho je možné dosáhnout mnoha různými technikami komprese, které v mnoha případech využívají různé typy kódování.
- Nejvýznamnějším typem kódování jsou tzv. *prefixové kódy* - žádné zakódované slovo není prefixem („předponou“) jiného kódového slova.

$$1001,\ 110,\ 10,\ \ldots \quad \times$$

- Všimněme si, že binární kód mezi prefixové rozhodně nepatří:

$$1_{10} \Rightarrow 1_2 \quad \text{a} \quad 2_{10} \Rightarrow 10_2 \quad \times$$

Kódování

- V určitých případech však můžeme ukládat pole čísel, kde má většina malou hodnotu, a chceme takovéto pole „zmenšit“.
- Toho je možné dosáhnout mnoha různými technikami komprese, které v mnoha případech využívají různé typy kódování.
- Nejvýznamnějším typem kódování jsou tzv. *prefixové kódy* - žádné zakódované slovo není prefixem („předponou“) jiného kódového slova.

$$1001,\ 110,\ 10,\ \ldots \quad \times$$

- Všimněme si, že binární kód mezi prefixové rozhodně nepatří:

$$1_{10} \Rightarrow 1_2 \quad \text{a} \quad 2_{10} \Rightarrow 10_2 \quad \times$$

Kódování

- V určitých případech však můžeme ukládat pole čísel, kde má většina malou hodnotu, a chceme takovéto pole „zmenšit“.
- Toho je možné dosáhnout mnoha různými technikami komprese, které v mnoha případech využívají různé typy kódování.
- Nejvýznamnějším typem kódování jsou tzv. *prefixové kódy* - žádné zakódované slovo není prefixem („předponou“) jiného kódového slova.

$$1001,\ 110,\ 10,\ \ldots \quad \times$$

- Všimněme si, že binární kód mezi prefixové rozhodně nepatří:

$$1_{10} \Rightarrow 1_2 \quad \text{a} \quad 2_{10} \Rightarrow 10_2 \quad \times$$

Kódování

- V určitých případech však můžeme ukládat pole čísel, kde má většina malou hodnotu, a chceme takovéto pole „zmenšit“.
- Toho je možné dosáhnout mnoha různými technikami komprese, které v mnoha případech využívají různé typy kódování.
- Nejvýznamnějším typem kódování jsou tzv. *prefixové kódy* - žádné zakódované slovo není prefixem („předponou“) jiného kódového slova.

$$1001,\ 110,\ 10,\ \ldots \quad \times$$

- Všimněme si, že binární kód mezi prefixové rozhodně nepatří:

$$1_{10} \Rightarrow 1_2 \quad \text{a} \quad 2_{10} \Rightarrow 10_2 \quad \times$$

Fibonacciho čísla

- Fibonacciho čísla: 1, 1, 2, 3, 5, 8, ...
- Fibonacciho posloupnost F_n je definována (rekurentně) takto:
  - $F_1 = 1$,
  - $F_2 = 1$,
  - $F_i = F_{i-1} + F_{i-2}$ pro $i > 2$.
- Idealizovaný růst králičí populace

Fibonacciho čísla

- Fibonacciho čísla: 1, 1, 2, 3, 5, 8, ...
- Fibonacciho posloupnost F_n je definována (rekurentně) takto:
  - $F_1 = 1$,
  - $F_2 = 1$,
  - $F_i = \mathrm{F}_{i-1} + \mathrm{F}_{i-2}$ pro $i > 2$.
- Idealizovaný růst králičí populace

Fibonacciho čísla

- Fibonacciho čísla: 1, 1, 2, 3, 5, 8, ...
- Fibonacciho posloupnost F_n je definována (rekurentně) takto:
  - $F_1 = 1$,
  - $F_2 = 1$,
  - $F_i = \mathrm{F}_{i-1} + \mathrm{F}_{i-2}$ pro $i > 2$.
- Idealizovaný růst králičí populace

Fibonacciho kódování

- Každé přirozené číslo lze vyjádřit jako součet několika různých Fibonacciho čísel.
- Nechť $n \in \mathbb{N}$. Vytvořme reprezentaci $FR(n) = b_1 b_2 \cdots b_s$ čísla n pomocí Fibonacciho čísel:

$$n = \sum_{i=1}^{s} b_i F_{i+1}, \quad b_i \in \{0, 1\}.$$

- Nechť reprezentace $FR(n)$ nikdy neobsahuje dvě jedničky za sebou.
- Fibonacciho kód $\mathcal{F}(n)$ čísla n pak vzniká přidáním jedničky za $FR(n)$.

n	$FR(n)$	$\mathcal{F}(n)$
1	1	11
2	01	011
3	001	0011
4	101	1011
8	00001	000011
100	0010100001	00101000011

Fibonacciho kódování

- Každé přirozené číslo lze vyjádřit jako součet několika různých Fibonacciho čísel.
- Nechť $n \in \mathbb{N}$. Vytvořme reprezentaci $FR(n) = b_1 b_2 \cdots b_s$ čísla n pomocí Fibonacciho čísel:

$$n = \sum_{i=1}^{s} b_i F_{i+1}, \quad b_i \in \{0, 1\}.$$

- Nechť reprezentace $FR(n)$ nikdy neobsahuje dvě jedničky za sebou.
- Fibonacciho kód $\mathcal{F}(n)$ čísla n pak vzniká přidáním jedničky za $FR(n)$.

n	$FR(n)$	$\mathcal{F}(n)$
1	1	11
2	01	011
3	001	0011
4	101	1011
8	00001	000011
100	0010100001	00101000011

Fibonacciho kódování

- Každé přirozené číslo lze vyjádřit jako součet několika různých Fibonacciho čísel.
- Nechť $n \in \mathbb{N}$. Vytvořme reprezentaci $FR(n) = b_1 b_2 \cdots b_s$ čísla n pomocí Fibonacciho čísel:

$$n = \sum_{i=1}^{s} b_i F_{i+1}, \quad b_i \in \{0, 1\}.$$

- Nechť reprezentace $FR(n)$ nikdy neobsahuje dvě jedničky za sebou.
- Fibonacciho kód $\mathcal{F}(n)$ čísla n pak vzniká přidáním jedničky za $FR(n)$.

n	$FR(n)$	$\mathcal{F}(n)$
1	1	11
2	01	011
3	001	0011
4	101	1011
8	00001	000011
100	0010100001	00101000011

Fibonacciho kódování

- Každé přirozené číslo lze vyjádřit jako součet několika různých Fibonacciho čísel.
- Nechť $n \in \mathbb{N}$. Vytvořme reprezentaci $FR(n) = b_1 b_2 \cdots b_s$ čísla n pomocí Fibonacciho čísel:

$$n = \sum_{i=1}^{s} b_i F_{i+1}, \quad b_i \in \{0, 1\}.$$

- Nechť reprezentace $FR(n)$ nikdy neobsahuje dvě jedničky za sebou.
- Fibonacciho kód $\mathcal{F}(n)$ čísla n pak vzniká přidáním jedničky za $FR(n)$.

n	$FR(n)$	$\mathcal{F}(n)$
1	1	11
2	01	011
3	001	0011
4	101	1011
8	00001	000011
100	0010100001	00101000011

Fibonacciho kódování

- Každé přirozené číslo lze vyjádřit jako součet několika různých Fibonacciho čísel.
- Nechť $n \in \mathbb{N}$. Vytvořme reprezentaci $FR(n) = b_1 b_2 \cdots b_s$ čísla n pomocí Fibonacciho čísel:

$$n = \sum_{i=1}^{s} b_i F_{i+1}, \quad b_i \in \{0, 1\}.$$

- Nechť reprezentace $FR(n)$ nikdy neobsahuje dvě jedničky za sebou.
- Fibonacciho kód $\mathcal{F}(n)$ čísla n pak vzniká přidáním jedničky za $FR(n)$.

n	$FR(n)$	$\mathcal{F}(n)$
1	1	11
2	01	011
3	001	0011
4	101	1011
8	00001	000011
100	0010100001	00101000011

Algoritmus Fibonacciho kódování

- Při kódování čísla $n \in \mathbb{N}$ používáme zásobník:
  1. Nalezneme index i největšího Fibonacciho čísla splňující podmínku $F_i \leq n$.
  2. Pokud $F_i \leq n$, nastavíme $n = n - F_i$ a umístíme 1 na zásobník. V opačném případě umístíme 0 na zásobník.
  3. Nastavíme $i = i - 1$ a pokud $i > 1$, pokračujeme krokem 2.
- Zásobník nyní obsahuje reprezentaci $FR(n)$ pomocí Fibonacciho čísel, takže pouze stačí tyto čísla přečíst a na konec umístit jedničku.
- Výsledkem je Fibonacciho kód $\mathcal{F}(n)$.

Algoritmus Fibonacciho kódování

- Při kódování čísla $n \in \mathbb{N}$ používáme zásobník:
  1. Nalezneme index i největšího Fibonacciho čísla splňující podmínku $F_i \leq n$.
  2. Pokud $F_i \leq n$, nastavíme $n = n - F_i$ a umístíme 1 na zásobník. V opačném případě umístíme 0 na zásobník.
  3. Nastavíme $i = i - 1$ a pokud $i > 1$, pokračujeme krokem 2.
- Zásobník nyní obsahuje reprezentaci $FR(n)$ pomocí Fibonacciho čísel, takže pouze stačí tyto čísla přečíst a na konec umístit jedničku.
- Výsledkem je Fibonacciho kód $\mathcal{F}(n)$.

Algoritmus Fibonacciho kódování

- Při kódování čísla $n \in \mathbb{N}$ používáme zásobník:
  1. Nalezneme index i největšího Fibonacciho čísla splňující podmínku $F_i \leq n$.
  2. Pokud $F_i \leq n$, nastavíme $n = n - F_i$ a umístíme 1 na zásobník. V opačném případě umístíme 0 na zásobník.
  3. Nastavíme $i = i - 1$ a pokud $i > 1$, pokračujeme krokem 2.
- Zásobník nyní obsahuje reprezentaci $FR(n)$ pomocí Fibonacciho čísel, takže pouze stačí tyto čísla přečíst a na konec umístit jedničku.
- Výsledkem je Fibonacciho kód $\mathcal{F}(n)$.

Algoritmus Fibonacciho kódování

- Při kódování čísla $n \in \mathbb{N}$ používáme zásobník:
  1. Nalezneme index i největšího Fibonacciho čísla splňující podmínku $F_i \leq n$.
  2. Pokud $F_i \leq n$, nastavíme $n = n - F_i$ a umístíme 1 na zásobník. V opačném případě umístíme 0 na zásobník.
  3. Nastavíme $i = i - 1$ a pokud $i > 1$, pokračujeme krokem 2.
- Zásobník nyní obsahuje reprezentaci $FR(n)$ pomocí Fibonacciho čísel, takže pouze stačí tyto čísla přečíst a na konec umístit jedničku.
- Výsledkem je Fibonacciho kód $\mathcal{F}(n)$.

Algoritmus Fibonacciho kódování

- Při kódování čísla $n \in \mathbb{N}$ používáme zásobník:
  1. Nalezneme index i největšího Fibonacciho čísla splňující podmínku $F_i \leq n$.
  2. Pokud $F_i \leq n$, nastavíme $n = n - F_i$ a umístíme 1 na zásobník. V opačném případě umístíme 0 na zásobník.
  3. Nastavíme $i = i - 1$ a pokud $i > 1$, pokračujeme krokem 2.
- Zásobník nyní obsahuje reprezentaci $FR(n)$ pomocí Fibonacciho čísel, takže pouze stačí tyto čísla přečíst a na konec umístit jedničku.
- Výsledkem je Fibonacciho kód $\mathcal{F}(n)$.

Algoritmus Fibonacciho kódování

- Při kódování čísla $n \in \mathbb{N}$ používáme zásobník:
  1. Nalezneme index i největšího Fibonacciho čísla splňující podmínku $F_i \leq n$.
  2. Pokud $F_i \leq n$, nastavíme $n = n - F_i$ a umístíme 1 na zásobník. V opačném případě umístíme 0 na zásobník.
  3. Nastavíme $i = i - 1$ a pokud $i > 1$, pokračujeme krokem 2.
- Zásobník nyní obsahuje reprezentaci $FR(n)$ pomocí Fibonacciho čísel, takže pouze stačí tyto čísla přečíst a na konec umístit jedničku.
- Výsledkem je Fibonacciho kód $\mathcal{F}(n)$.

Algoritmus Fibonacciho dekódování

- Postup při dekódování slova $\mathcal{F}(n)$:
  1. Nastavíme proměnné $n = 0$, $prev = 0$ a $i = 2$.
  2. Načteme aktuální bit ze vstupu obsahující zakódované číslo.
  3. Pokud aktuální bit a *prev* jsou rovny 1, pak algoritmus ukončíme.
  4. Pokud je aktuální bit roven 1, pak $n = n + F_i$.
  5. Nastavíme $i = i + 1$ a *prev* na hodnotu aktuálního bitu a pokračujeme krokem 2.
- V proměnné n je nyní hodnota $V(\mathcal{F}(n))$;

  $V(\mathcal{F}(n))$ ... dekódované slovo $\mathcal{F}(n)$, tj. n.
- *Všimneme si, jak algoritmy pracují s jednotlivými bity slova!*

Algoritmus Fibonacciho dekódování

- Postup při dekódování slova $\mathcal{F}(n)$:
  1. Nastavíme proměnné $n = 0$, $prev = 0$ a $i = 2$.
  2. Načteme aktuální bit ze vstupu obsahující zakódované číslo.
  3. Pokud aktuální bit a *prev* jsou rovny 1, pak algoritmus ukončíme.
  4. Pokud je aktuální bit roven 1, pak $n = n + F_i$.
  5. Nastavíme $i = i + 1$ a *prev* na hodnotu aktuálního bitu a pokračujeme krokem 2.
- V proměnné n je nyní hodnota $V(\mathcal{F}(n))$;

  $V(\mathcal{F}(n))$ ... dekódované slovo $\mathcal{F}(n)$, tj. n.
- *Všimneme si, jak algoritmy pracují s jednotlivými bity slova!*

Algoritmus Fibonacciho dekódování

- Postup při dekódování slova $\mathcal{F}(n)$:
  1. Nastavíme proměnné $n = 0$, $prev = 0$ a $i = 2$.
  2. Načteme aktuální bit ze vstupu obsahující zakódované číslo.
  3. Pokud aktuální bit a *prev* jsou rovny 1, pak algoritmus ukončíme.
  4. Pokud je aktuální bit roven 1, pak $n = n + F_i$.
  5. Nastavíme $i = i + 1$ a *prev* na hodnotu aktuálního bitu a pokračujeme krokem 2.
- V proměnné n je nyní hodnota $V(\mathcal{F}(n))$;

  $V(\mathcal{F}(n))$ ... dekódované slovo $\mathcal{F}(n)$, tj. n.
- *Všimneme si, jak algoritmy pracují s jednotlivými bity slova!*

Algoritmus Fibonacciho dekódování

- Postup při dekódování slova $\mathcal{F}(n)$:
  1. Nastavíme proměnné $n = 0$, $prev = 0$ a $i = 2$.
  2. Načteme aktuální bit ze vstupu obsahující zakódované číslo.
  3. Pokud aktuální bit a *prev* jsou rovny 1, pak algoritmus ukončíme.
  4. Pokud je aktuální bit roven 1, pak $n = n + F_i$.
  5. Nastavíme $i = i + 1$ a *prev* na hodnotu aktuálního bitu a pokračujeme krokem 2.
- V proměnné n je nyní hodnota $V(\mathcal{F}(n))$;

  $V(\mathcal{F}(n))$ ... dekódované slovo $\mathcal{F}(n)$, tj. n.
- *Všimneme si, jak algoritmy pracují s jednotlivými bity slova!*

Algoritmus Fibonacciho dekódování

- Postup při dekódování slova $\mathcal{F}(n)$:
  1. Nastavíme proměnné $n = 0$, $prev = 0$ a $i = 2$.
  2. Načteme aktuální bit ze vstupu obsahující zakódované číslo.
  3. Pokud aktuální bit a *prev* jsou rovny 1, pak algoritmus ukončíme.
  4. Pokud je aktuální bit roven 1, pak $n = n + F_i$.
  5. Nastavíme $i = i + 1$ a *prev* na hodnotu aktuálního bitu a pokračujeme krokem 2.
- V proměnné n je nyní hodnota $V(\mathcal{F}(n))$;

  $V(\mathcal{F}(n))$ ... dekódované slovo $\mathcal{F}(n)$, tj. n.
- *Všimneme si, jak algoritmy pracují s jednotlivými bity slova!*

Algoritmus Fibonacciho dekódování

- Postup při dekódování slova $\mathcal{F}(n)$:
  1. Nastavíme proměnné $n = 0$, $prev = 0$ a $i = 2$.
  2. Načteme aktuální bit ze vstupu obsahující zakódované číslo.
  3. Pokud aktuální bit a *prev* jsou rovny 1, pak algoritmus ukončíme.
  4. Pokud je aktuální bit roven 1, pak $n = n + F_i$.
  5. Nastavíme $i = i + 1$ a *prev* na hodnotu aktuálního bitu a pokračujeme krokem 2.
- V proměnné n je nyní hodnota $V(\mathcal{F}(n))$;

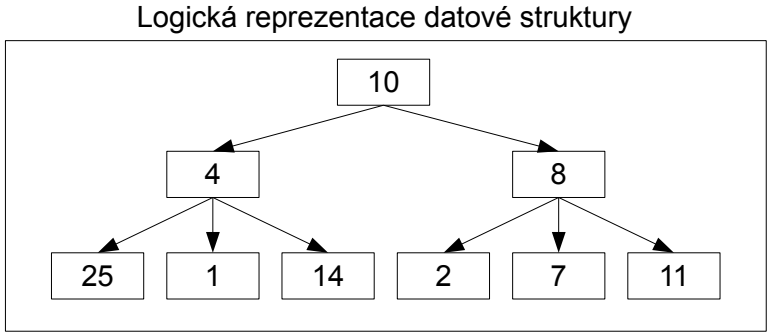
  $V(\mathcal{F}(n))$ ... dekódované slovo $\mathcal{F}(n)$, tj. n.
- *Všimneme si, jak algoritmy pracují s jednotlivými bity slova!*

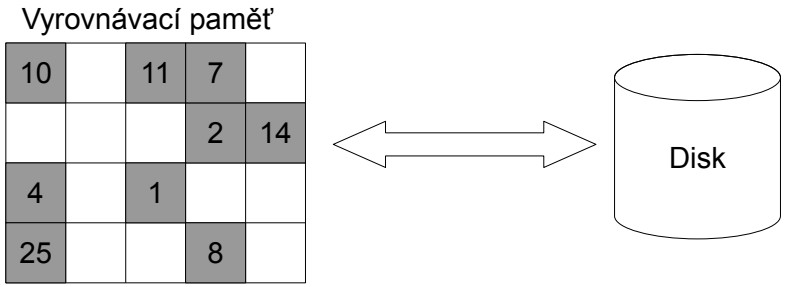
Algoritmus Fibonacciho dekódování

- Postup při dekódování slova $\mathcal{F}(n)$:
  1. Nastavíme proměnné $n = 0$, $prev = 0$ a $i = 2$.
  2. Načteme aktuální bit ze vstupu obsahující zakódované číslo.
  3. Pokud aktuální bit a *prev* jsou rovny 1, pak algoritmus ukončíme.
  4. Pokud je aktuální bit roven 1, pak $n = n + F_i$.
  5. Nastavíme $i = i + 1$ a *prev* na hodnotu aktuálního bitu a pokračujeme krokem 2.
- V proměnné n je nyní hodnota $V(\mathcal{F}(n))$;

  $V(\mathcal{F}(n))$ ... dekódované slovo $\mathcal{F}(n)$, tj. n.
- *Všimneme si, jak algoritmy pracují s jednotlivými bity slova!*

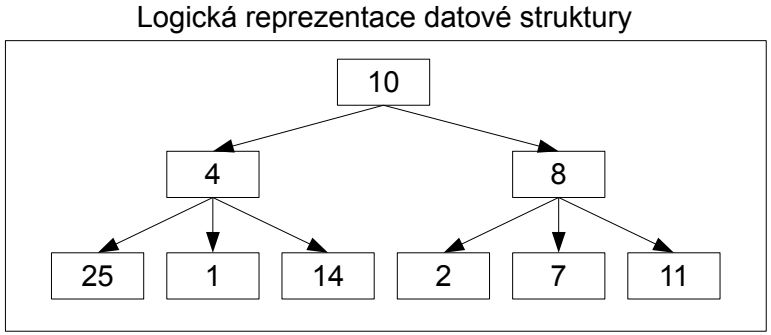
Datové struktury

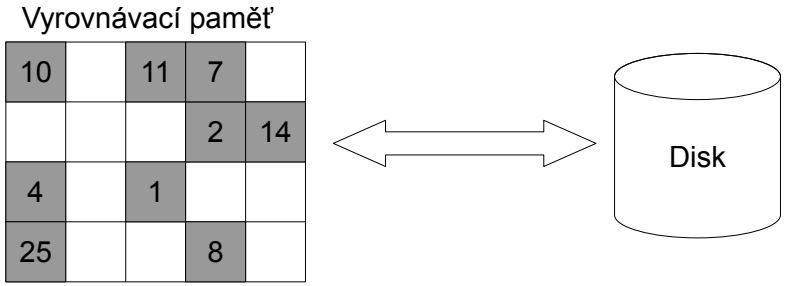




- V databázích se používá celá řada různých datových struktur, které slouží k rychlému vyhledávání dat (tzv. indexy).
- Funkčnost datové struktury si zjednodušeně můžeme představit jako funkci $f : K \mapsto H$, kde K je množina klíčů a H je množina vyhledávaných hodnot.
- Data jsou organizovány do stránek s pevnou velikostí, aby mohly být efektivně uloženy na disku.
- Používá se vyrovnávací paměť (cache), aby nedocházelo ke neustálému čtení z disku.

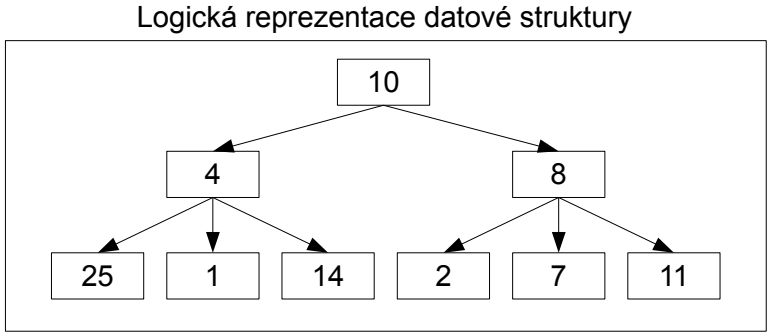
Datové struktury

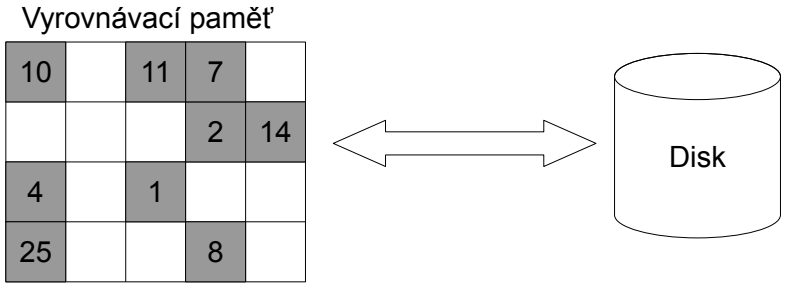




- V databázích se používá celá řada různých datových struktur, které slouží k rychlému vyhledávání dat (tzv. indexy).
- Funkčnost datové struktury si zjednodušeně můžeme představit jako funkci $f : K \mapsto H$, kde K je množina klíčů a H je množina vyhledávaných hodnot.
- Data jsou organizovány do stránek s pevnou velikostí, aby mohly být efektivně uloženy na disku.
- Používá se vyrovnávací paměť (cache), aby nedocházelo ke neustálému čtení z disku.

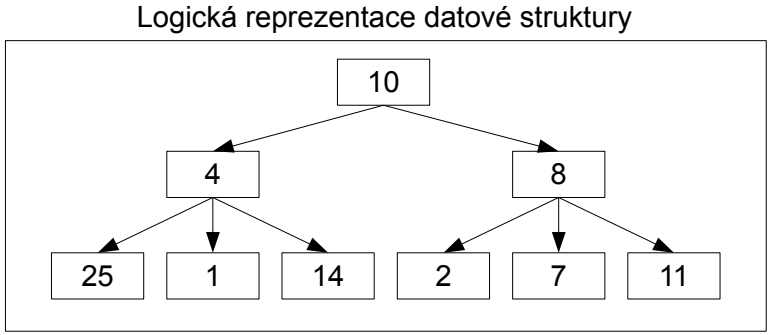
Datové struktury

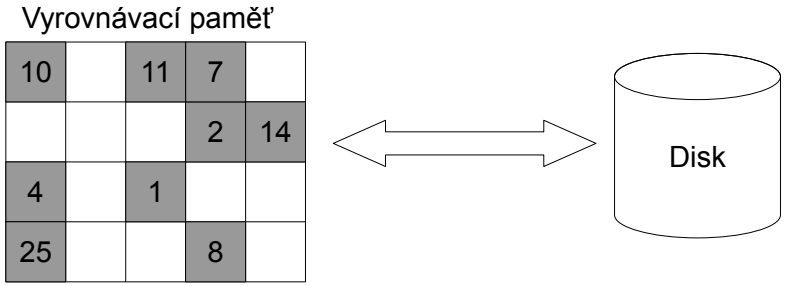




- V databázích se používá celá řada různých datových struktur, které slouží k rychlému vyhledávání dat (tzv. indexy).
- Funkčnost datové struktury si zjednodušeně můžeme představit jako funkci $f : K \mapsto H$, kde K je množina klíčů a H je množina vyhledávaných hodnot.
- Data jsou organizovány do stránek s pevnou velikostí, aby mohly být efektivně uloženy na disku.
- Používá se vyrovnávací paměť (cache), aby nedocházelo ke neustálému čtení z disku.

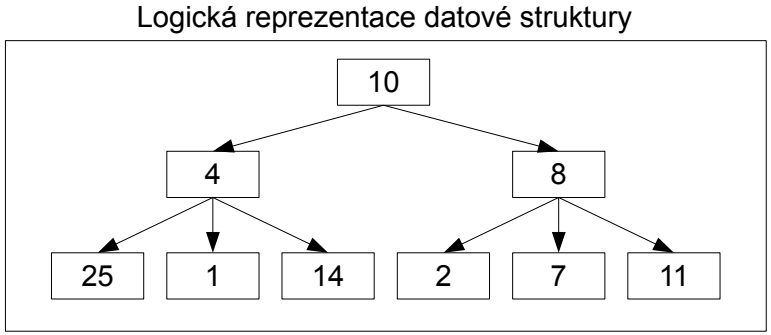
Datové struktury

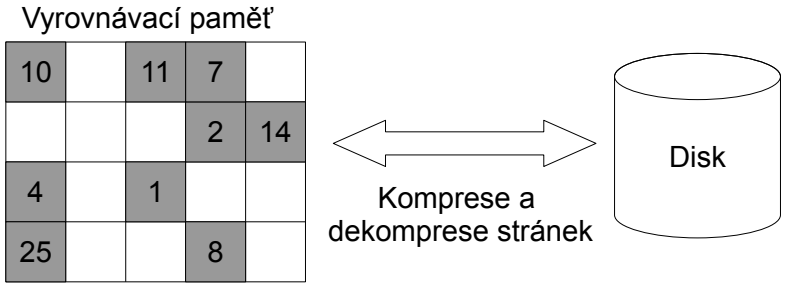




- V databázích se používá celá řada různých datových struktur, které slouží k rychlému vyhledávání dat (tzv. indexy).
- Funkčnost datové struktury si zjednodušeně můžeme představit jako funkci $f : K \mapsto H$, kde K je množina klíčů a H je množina vyhledávaných hodnot.
- Data jsou organizovány do stránek s pevnou velikostí, aby mohly být efektivně uloženy na disku.
- Používá se vyrovnávací paměť (cache), aby nedocházelo ke neustálému čtení z disku.

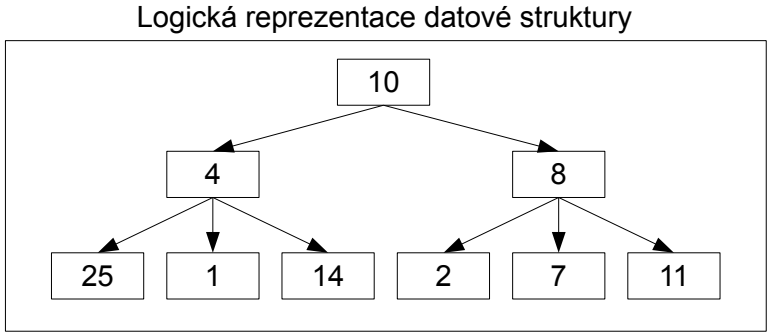
Komprese datových struktur

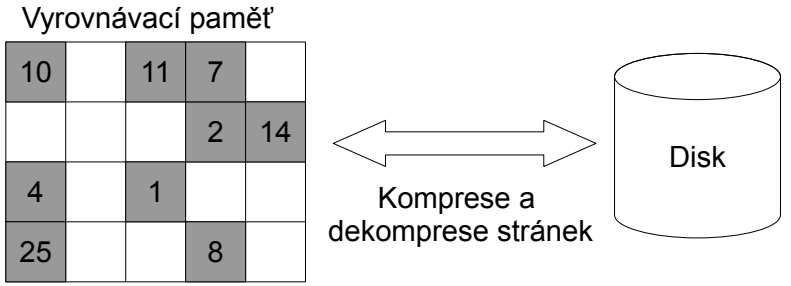




- Data ve stránce datové struktury jsou často velmi podobná (vychází to z uspořádání klíčů).
- Přenos mezi vyrovnávací pamětí a diskem je řádově pomalejší než jakákoli jiná operace.
- Provedli jsme tedy sérii testů, kde jsou data na disku komprimována, abychom minimalizovali dobu a velikost přenosu mezi cache a diskem.
- *V takovémto schématu je kritická rychlost komprese a dekomprese stránek.*

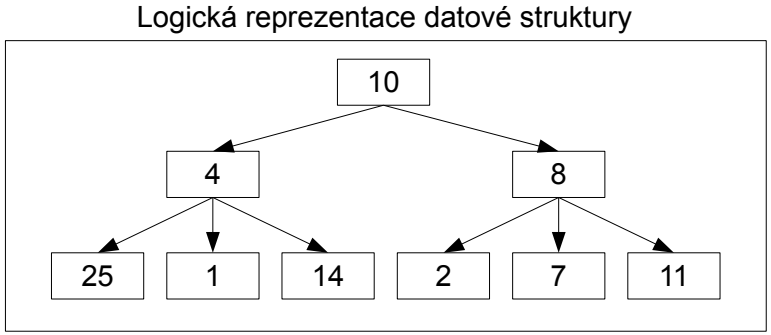
Komprese datových struktur

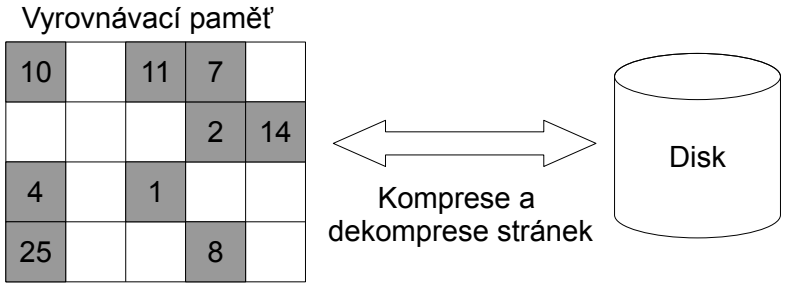




- Data ve stránce datové struktury jsou často velmi podobná (vychází to z uspořádání klíčů).
- Přenos mezi vyrovnávací pamětí a diskem je řádově pomalejší než jakákoli jiná operace.
- Provedli jsme tedy sérii testů, kde jsou data na disku komprimována, abychom minimalizovali dobu a velikost přenosu mezi cache a diskem.
- *V takovémto schématu je kritická rychlost komprese a dekomprese stránek.*

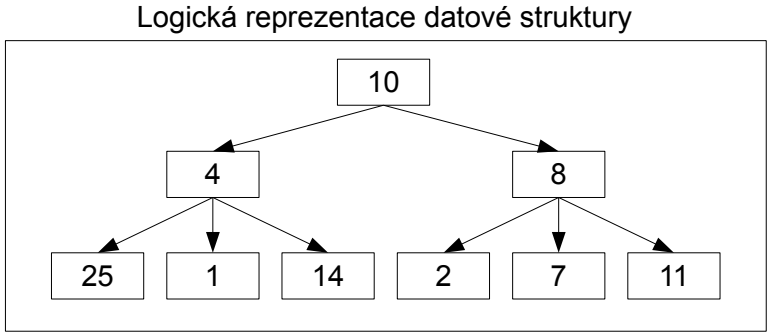
Komprese datových struktur

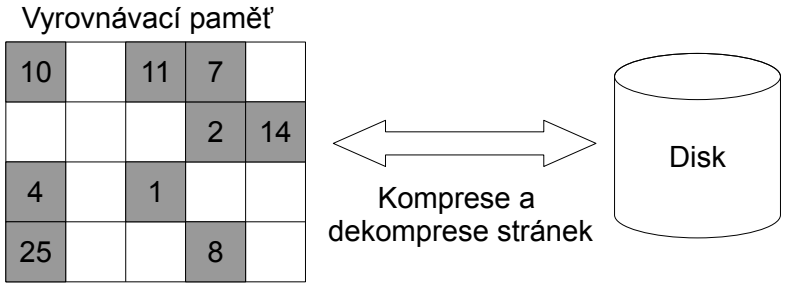




- Data ve stránce datové struktury jsou často velmi podobná (vychází to z uspořádání klíčů).
- Přenos mezi vyrovnávací pamětí a diskem je řádově pomalejší než jakákoli jiná operace.
- Provedli jsme tedy sérii testů, kde jsou data na disku komprimována, abychom minimalizovali dobu a velikost přenosu mezi cache a diskem.
- *V takovémto schématu je kritická rychlost komprese a dekomprese stránek.*

Komprese datových struktur





- Data ve stránce datové struktury jsou často velmi podobná (vychází to z uspořádání klíčů).
- Přenos mezi vyrovnávací pamětí a diskem je řádově pomalejší než jakákoli jiná operace.
- Provedli jsme tedy sérii testů, kde jsou data na disku komprimována, abychom minimalizovali dobu a velikost přenosu mezi cache a diskem.
- *V takovémto schématu je kritická rychlost komprese a dekomprese stránek.*

Fibonacciho dekódování

- Čtení kódového slova po bitech je pomalé.
- Nabízí se metoda čtení celých bytů, přičemž budeme mít tabulku MAP, kde budou předpočítány hodnoty dekódovaných čísel pro vstupní zakódovanou sekvenci o velikosti jednoho bytu (8 bitů).
- Mějme kódové slovo 00101011, pak dekadická hodnota 43 tohoto slova nám udává adresu v tabulce MAP, kde je předpočítána hodnota $V(00101011) = 32$.
- *Tato metoda má však háček ...*

Fibonacciho dekódování

- Čtení kódového slova po bitech je pomalé.
- Nabízí se metoda čtení celých bytů, přičemž budeme mít tabulku MAP, kde budou předpočítány hodnoty dekódovaných čísel pro vstupní zakódovanou sekvenci o velikosti jednoho bytu (8 bitů).
- Mějme kódové slovo 00101011, pak dekadická hodnota 43 tohoto slova nám udává adresu v tabulce MAP, kde je předpočítána hodnota $V(00101011) = 32$.
- *Tato metoda má však háček ...*

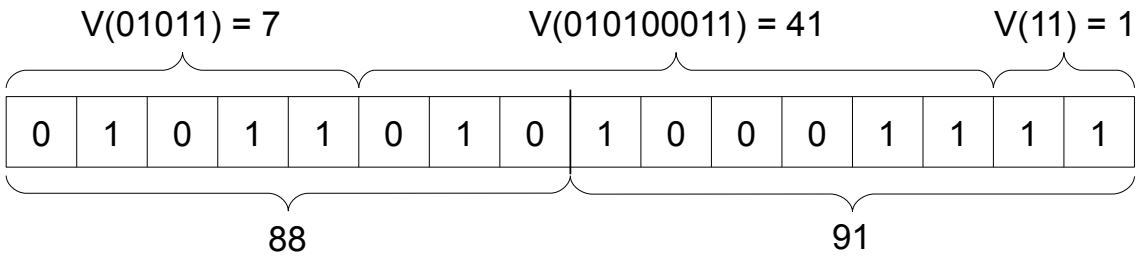
Fibonacciho dekódování

- Čtení kódového slova po bitech je pomalé.
- Nabízí se metoda čtení celých bytů, přičemž budeme mít tabulku MAP, kde budou předpočítány hodnoty dekódovaných čísel pro vstupní zakódovanou sekvenci o velikosti jednoho bytu (8 bitů).
- Mějme kódové slovo 00101011, pak dekadická hodnota 43 tohoto slova nám udává adresu v tabulce MAP, kde je předpočítána hodnota $V(00101011) = 32$.
- *Tato metoda má však háček ...*

Fibonacciho dekódování

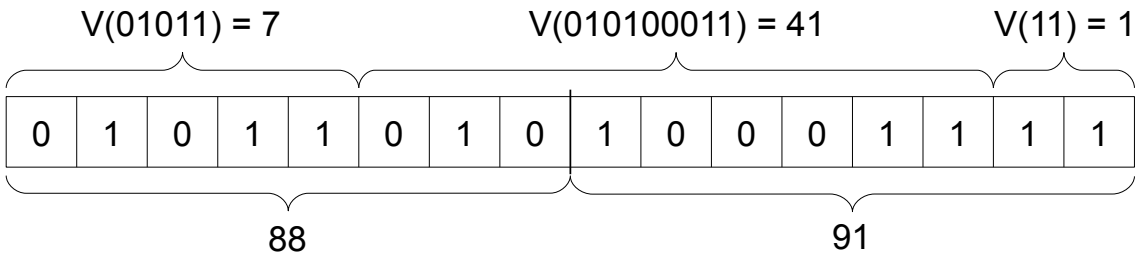
- Čtení kódového slova po bitech je pomalé.
- Nabízí se metoda čtení celých bytů, přičemž budeme mít tabulku MAP, kde budou předpočítány hodnoty dekódovaných čísel pro vstupní zakódovanou sekvenci o velikosti jednoho bytu (8 bitů).
- Mějme kódové slovo 00101011, pak dekadická hodnota 43 tohoto slova nám udává adresu v tabulce MAP, kde je předpočítána hodnota $V(00101011) = 32$.
- *Tato metoda má však háček ...*

Příklad (1/3)



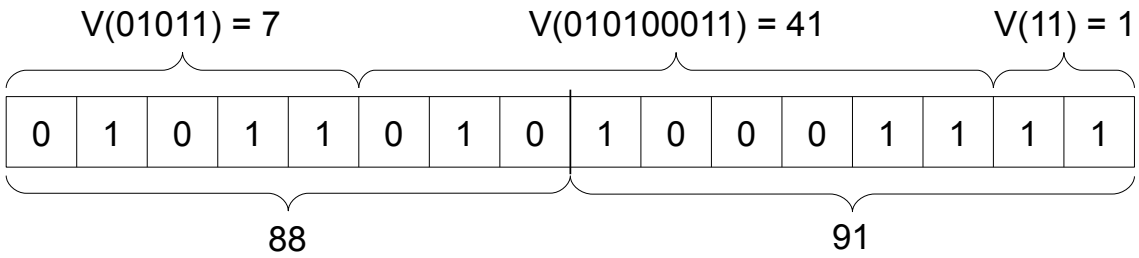
- Co když je slovo kódováno do více bytů?
- Slovo nahoře na obrázku je zakódovaná sekvence čísel 7, 41, 1.
- MAP[88] = 7, 2
- MAP[91] = 9, 1
- První a poslední čísla byla korektně dekódována, jelikož jsou v bytu obsažena celá. Číslo 41 je ale rozděleno do dvou bytů a otázkou je, zda-li je možné získat výsledné číslo bez bitového čtení slova.

Příklad (1/3)



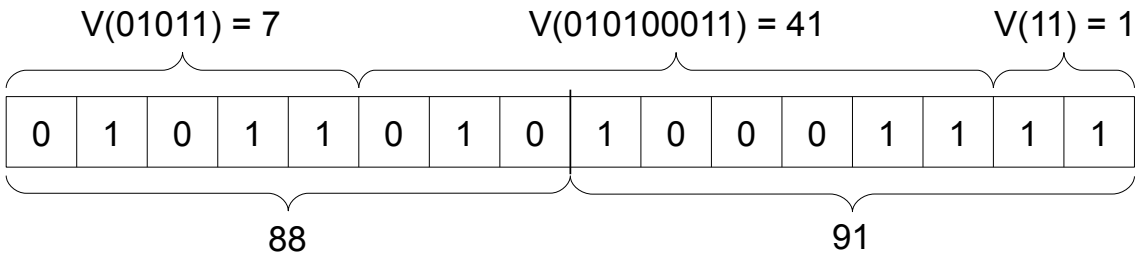
- Co když je slovo kódováno do více bytů?
- Slovo nahoře na obrázku je zakódovaná sekvence čísel 7, 41, 1.
- MAP[88] = 7, 2
- MAP[91] = 9, 1
- První a poslední čísla byla korektně dekódována, jelikož jsou v bytu obsažena celá. Číslo 41 je ale rozděleno do dvou bytů a otázkou je, zda-li je možné získat výsledné číslo bez bitového čtení slova.

Příklad (1/3)



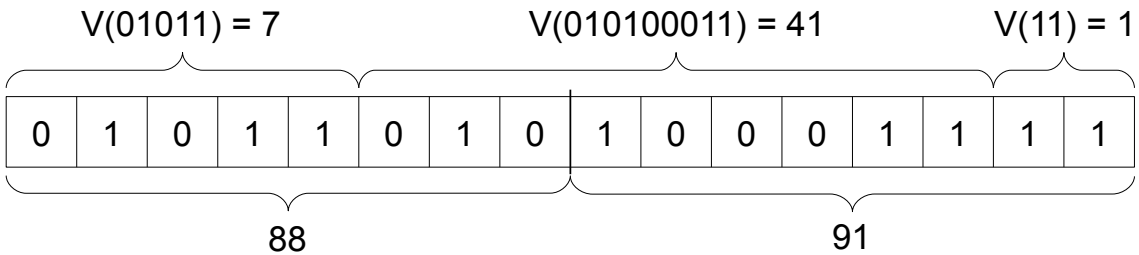
- Co když je slovo kódováno do více bytů?
- Slovo nahoře na obrázku je zakódovaná sekvence čísel 7, 41, 1.
- MAP[88] = 7, 2
- MAP[91] = 9, 1
- První a poslední čísla byla korektně dekódována, jelikož jsou v bytu obsažena celá. Číslo 41 je ale rozděleno do dvou bytů a otázkou je, zda-li je možné získat výsledné číslo bez bitového čtení slova.

Příklad (1/3)



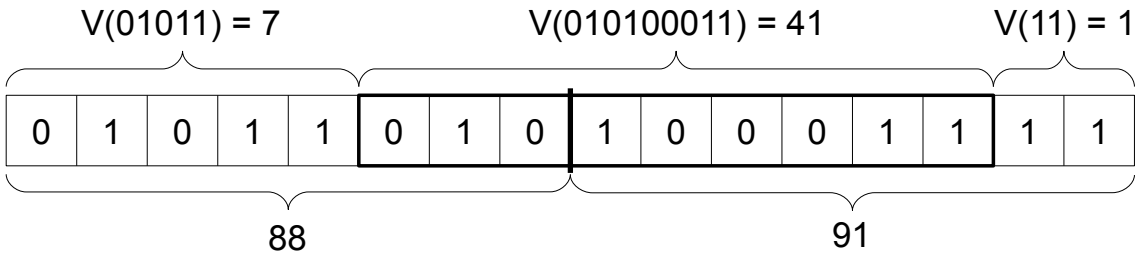
- Co když je slovo kódováno do více bytů?
- Slovo nahoře na obrázku je zakódovaná sekvence čísel 7, 41, 1.
- MAP[88] = 7, 2
- MAP[91] = 9, 1
- První a poslední čísla byla korektně dekódována, jelikož jsou v bytu obsažena celá. Číslo 41 je ale rozděleno do dvou bytů a otázkou je, zda-li je možné získat výsledné číslo bez bitového čtení slova.

Příklad (1/3)



- Co když je slovo kódováno do více bytů?
- Slovo nahoře na obrázku je zakódovaná sekvence čísel 7, 41, 1.
- MAP[88] = 7, 2
- MAP[91] = 9, 1
- První a poslední čísla byla korektně dekódována, jelikož jsou v bytu obsažena celá. Číslo 41 je ale rozděleno do dvou bytů a otázkou je, zda-li je možné získat výsledné číslo bez bitového čtení slova.

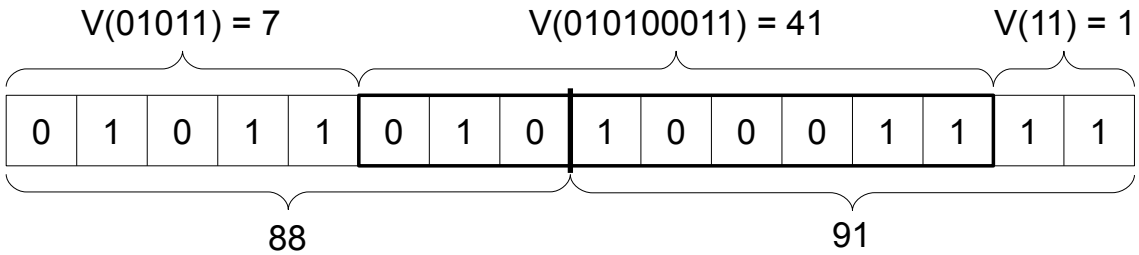
Příklad (2/3)



- Při dekódování víme, že v posledním bytu byly tři bity nedokončeného slova.
- Potřebujeme tedy původní kódové slovo 10001 „posunout“ o tři bity doprava a toto slovo dekódovat.
- Tuto operaci nazveme *Fibonacciho pravý shift* a pro kódové slovo $\mathcal{F}(n) = b_1 b_2 \cdots b_s 1$ je definována takto:

$$\mathcal{F}(n) <<_F k \;=\; \overbrace{00\cdots0}^{k} b_1 b_2 \cdots b_s 1$$

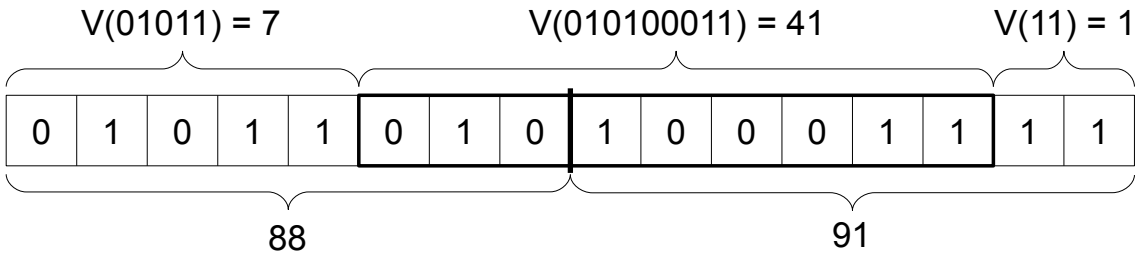
Příklad (2/3)



- Při dekódování víme, že v posledním bytu byly tři bity nedokončeného slova.
- Potřebujeme tedy původní kódové slovo 10001 „posunout“ o tři bity doprava a toto slovo dekódovat.
- Tuto operaci nazveme *Fibonacciho pravý shift* a pro kódové slovo $\mathcal{F}(n) = b_1 b_2 \cdots b_s 1$ je definována takto:

$$\mathcal{F}(n) <<_F k \;=\; \overbrace{00\cdots0}^{k} b_1 b_2 \cdots b_s 1$$

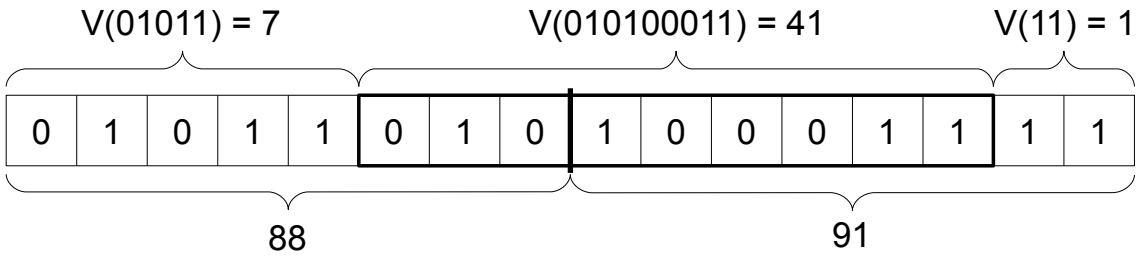
Příklad (2/3)



- Při dekódování víme, že v posledním bytu byly tři bity nedokončeného slova.
- Potřebujeme tedy původní kódové slovo 10001 „posunout“ o tři bity doprava a toto slovo dekódovat.
- Tuto operaci nazveme *Fibonacciho pravý shift* a pro kódové slovo $\mathcal{F}(n) = b_1 b_2 \cdots b_s 1$ je definována takto:

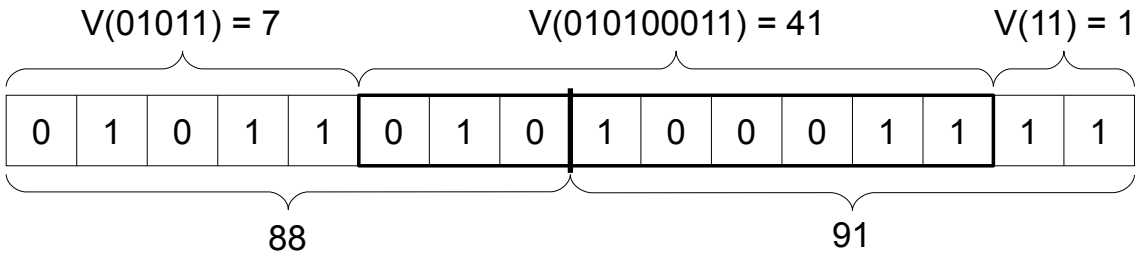
$$\mathcal{F}(n) <<_F k \;=\; \overbrace{00\cdots0}^{k} b_1 b_2 \cdots b_s 1$$

Příklad (3/3)



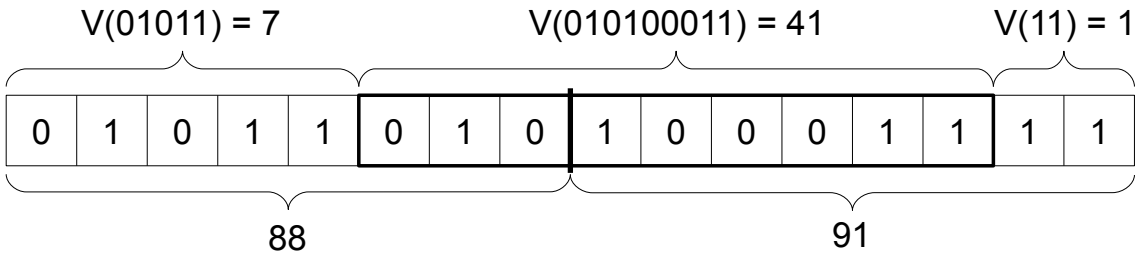
- Pokud bychom skutečně vzali původní kódové slovo 100011, „přidali“ tři nuly na začátek a zkoušeli hledat v MAP tabulce, tak neuspějeme, jelikož je délka slova větší než 8 bitů.
- Naštěstí existuje řešení, které nám umožní dekódovat slovo posunuté o libovolný počet bitů. A to dokonce bez nutnosti samotného vytvoření posunutého kódového slova.

Příklad (3/3)



- Pokud bychom skutečně vzali původní kódové slovo 100011, „přidali“ tři nuly na začátek a zkoušeli hledat v MAP tabulce, tak neuspějeme, jelikož je délka slova větší než 8 bitů.
- Naštěstí existuje řešení, které nám umožní dekódovat slovo posunuté o libovolný počet bitů. A to dokonce bez nutnosti samotného vytvoření posunutého kódového slova.

Příklad (3/3)



- Pokud bychom skutečně vzali původní kódové slovo 100011, „přidali“ tři nuly na začátek a zkoušeli hledat v MAP tabulce, tak neuspějeme, jelikož je délka slova větší než 8 bitů.
- Naštěstí existuje řešení, které nám umožní dekódovat slovo posunuté o libovolný počet bitů. A to dokonce bez nutnosti samotného vytvoření posunutého kódového slova.

Rychlý Fibonacciho shift

Theorem (V. Snášel, 2007)

Mějme Fibonacciho kód $\mathcal{F}(n)$ čísla n. Potom

$$V(\mathcal{F}(n) <<_F k) = F_k \cdot V(\mathcal{F}(n)) + F_{k-1} \cdot V(\mathcal{F}(n) >>_F 1).$$

Poznámka: $>>_F$ ... Fibonacciho levý shift (analogický pravému)

Platí, že:

- Všechny hodnoty mohou být předpočítány do tabulek, jejichž počet řádků je meší než 2^8.
- Výsledkem je Fibonacciho dekódování, které vůbec nepracuje s jednotlivými bity kódového slova.

Rychlý Fibonacciho shift

Theorem (V. Snášel, 2007)

Mějme Fibonacciho kód $\mathcal{F}(n)$ čísla n. Potom

$$V(\mathcal{F}(n) <<_F k) = F_k \cdot V(\mathcal{F}(n)) + F_{k-1} \cdot V(\mathcal{F}(n) >>_F 1).$$

Poznámka: $>>_F$ ... Fibonacciho levý shift (analogický pravému)

Platí, že:

- Všechny hodnoty mohou být předpočítány do tabulek, jejichž počet řádků je meší než 2^8.
- Výsledkem je Fibonacciho dekódování, které vůbec nepracuje s jednotlivými bity kódového slova.

Rychlý Fibonacciho shift

Theorem (V. Snášel, 2007)

Mějme Fibonacciho kód $\mathcal{F}(n)$ čísla n. Potom

$$V(\mathcal{F}(n) <<_F k) = F_k \cdot V(\mathcal{F}(n)) + F_{k-1} \cdot V(\mathcal{F}(n) >>_F 1).$$

Poznámka: $>>_F$ ... Fibonacciho levý shift (analogický pravému)

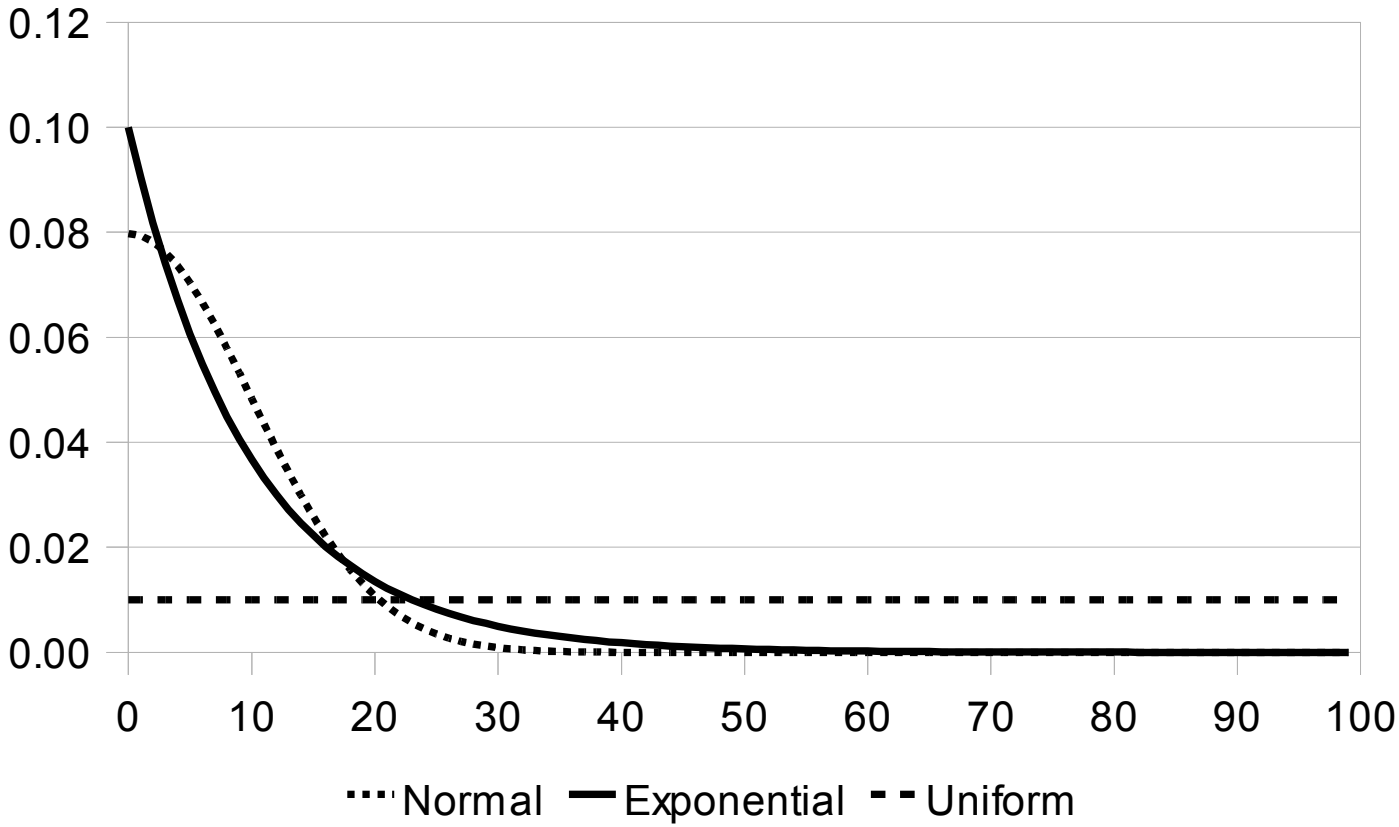
Platí, že:

- Všechny hodnoty mohou být předpočítány do tabulek, jejichž počet řádků je meší než 2^8.
- Výsledkem je Fibonacciho dekódování, které vůbec nepracuje s jednotlivými bity kódového slova.

Nastavení a data

- AMD Opteron 1,8 GHz, Windows Server 2008
- Datové kolekce:
  - Rovnoměrné rozložení čísel pro 8, 16, 32 a 64 bitová čísla.
  - Normální distribuce 32 bitových čísel.
  - Exponenciální distribuce 32 bitových čísel
- Každá kolekce obsahovala 10 miliónů čísel.

Nastavení a data



Porovnání času dekomprese (J. Walder, 2007)

Kolekce	Kopírování dat [s]	Dekódování bit po bitu [s]	Rychlé dekódování [s]
8-bit		3,32	0,8
16-bit		6,35	1,42
32-bit		12,9	2,74
64-bit		22,23	5,33
Exponenciální	0,41	6,66	1,49
Normální	0,41	6,49	1,47

TABULKA: Rychlost dekódování

V prvním sloupci můžeme pro porovnání nalézt dobu kopírování nekomprimovaných dat z paměti do paměti.

Díky za pozornost!