Tabulka

Často jde nejenom o **dynamickou** množinu, ale i **statickou** množinu

Matematické tabulky –
statická množina

x	cos x
0,0	1,000
0,1	0,995
0,2	0,980
0,3	0,955
0,4	0,921
0,5	0,878
0,6	0,825
0,7	0,765
0,8	0,697
0,9	0,622
1,0	0,540
1,1	0,454
1,2	0,362
1,3	0,267
1,4	0,170
1,5	0,071
1,6	-0,029
1,7	-0,129
1,8	-0,227
1,9	-0,323

(x, cos x) – *prvek tabulky*
x – *klíč*

Startovní listina –
dynamická množina

Příjmení	Startovní číslo
ADAM	1
DOLEJŠ	2
HESOUN	3
JANDÁK	4
ŠANTALA	5
ŠEFLOVÁ	6
ŠINDEL	7
ŠNAJDR	8
ŠONKA	9
ŠOTOLA	10
ŠTĚPÁN	11
TICHÝ	12
TICHÝ	13
ŤOUPALÍK	14
VÁPENÍK	15
VINTÍŘ	16
VYDRÁK	17
	18
ZDVOŘILÝ	19
ŽEBRÁK	20

// závodník se startovním číslem 18 nenastoupil

(příjmení, ... *jméno, věk, nejlepší výkon,...* startovní číslo) - ***prvek***
startovní číslo - ***klíč***

Jak se jmenuje závodník se startovním číslem 11?

Implementace

- ***Spojový seznam - O(n)***
- ***Seřazené pole podle klíče - O(log n)***
- ***BVS - O(log n)***

Jde to rychleji ?

O(1) ?

Prvek uložíme do prvku pole s indexem rovným klíči ukládaného prvku

O(1)

1	ADAM	1
2	DOLEJŠ	2
3	HESOUN	3
4	JANDÁK	4
5	ŠANTALA	5
6	ŠEFLOVÁ	6
7	ŠINDEL	7
8	ŠNAJDR	8
9	ŠONKA	9
10	ŠOTOLA	10
11	ŠTĚPÁN	11
12	TICHÝ	12
13	TICHÝ	13
14	ŤOUPALÍK	14
15	VÁPENÍK	15
16	VINTÍŘ	16
17	VYDRÁK	17
18		18
19	ZDVOŘILÝ	19
20	ŽEBRÁK	20

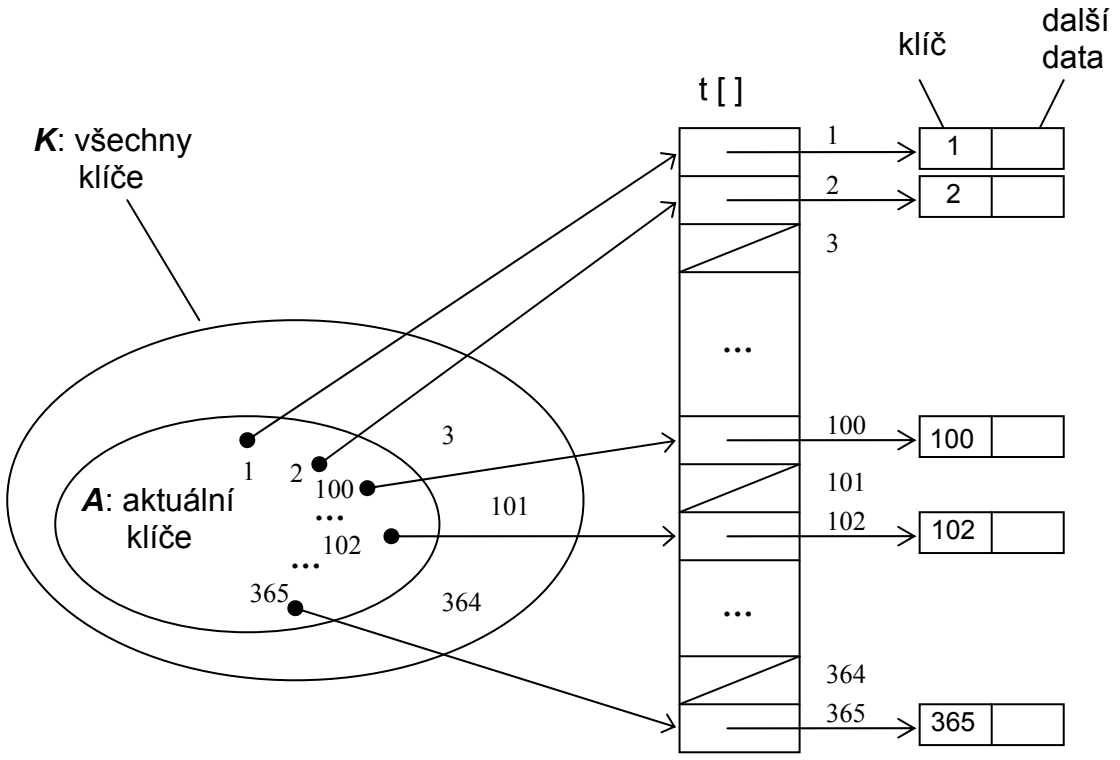
Tabulka s přímým adresováním – *O(1)*

- **klíče** všech prvků jsou **různé**

- **klíče** prvků jsou z množiny ***K*** o velikosti $|K|$
- každému klíči odpovídá prvek pole `Prvek[] t`
  `t.length` $= |K|$, v tabulce je místo pro každý „klíč“
- množina klíčů aktuálně uložených prvků $A \subseteq K$, $|A| \leq |K|$

Příklad:

startující se startovními čísly 1 až 365,
- $K = \{1,2,...,365\}$, $|K| = 365$
tabulka
- `t.length` $= |K| = 365$
startující s čísly 3, 101 a 364 ke startu nenastoupili,
- $A = \{1,2,4,...,363,365\}$, $|A| = 362$



Implementace

```
class Prvek {
  int klic;
  String data;
  …
  Prvek(int klic, String data) {
    this.klic = klic;
    this.data = data;
  }
}
```

```
class Tabulka {
  // rozhrani ADT
  Tabulka()
  Prvek hledej(int)
  vloz(Prvek)
  vyber(Prvek)
}
```

```
class Tabulka {
  Prvek[] t = new Prvek[|K|];

  Tabulka() {
    for(int i = 0; i < t.length; i++) {
      t[i] = null; // prázdná tabulka
  }

  Prvek hledej(int k) {
    return t[k];
  }

  void vloz (Prvek x) {
    t[x.klic] = x;
  }

  void vyber (Prvek x) {
    t[x.klic] = null;
  }

}
```

- všechny operace jsou rychlé, čas je ***O(1)***
- vhodné, není-li velikost $|K|$ množiny všech klíčů ***K*** velká

Poznámky:

Alternativní implementace by mohla namísto referencí na prvky dynamické množiny v poli implementujícím tabulku obsahovat přímo samotné prvky.

Klíč prvků známe na základě indexu a nemusíme jej uchovávat v prvku samotném. Musíme ovšem být schopni poznat, že prvek pole je prázdný.

Operace **`vloz`** a **`vyber`** mají jako **parametr ukazatel na prvek množiny**. Důvodem takové konstrukce může být **existence prvků** množiny **před jejich organizací v tabulce**.

Příkladem může být pole všech přihlášených závodníků, ale do tabulky jejich umístnění a dosáhnutého výkonu vložíme jenom ty závodníky, kteří nastoupili a závod dokončili. Obdobně, byl-li některý závodník diskvalifikován nebude se nacházet v tabulce výsledků.

Zákon 133/2000

Informační systém evidence obyvatel
§ 3

(1) Evidence obyvatel je vedena v informačním systému, jehož správcem[5)] je ministerstvo.

(2) V informačním systému se vedou o občanech tyto údaje:

a) jméno, příjmení, případně jejich změna, rodné příjmení,
b) datum narození,
c) pohlaví a jeho změna,
d) obec a okres narození a u občana, který se narodil v cizině, pouze stát narození,
e) **rodné číslo,**
f) státní občanství,
g) adresa místa trvalého pobytu (§ 10 odst. 1), včetně předchozích adres místa trvalého pobytu,
h) počátek trvalého pobytu, popřípadě datum zrušení údaje o místu trvalého pobytu,
i) zbavení nebo omezení způsobilosti k právním úkonům,
j) zákaz pobytu a doba jeho trvání,
k) rodné číslo otce, matky, popřípadě jiného zákonného zástupce, v případě, že jeden z rodičů nemá rodné číslo, vede se datum narození,
l) rodinný stav, datum a místo jeho změny,
m) rodné číslo manžela, nebo jméno a příjmení manžela, je-li manželem cizinec, který nemá přiděleno rodné číslo,
n) rodné číslo dítěte,
o) osvojení dítěte [§ 7 písm. b)],
p) záznam o poskytnutí údajů (§ 8 odst. 6),
r) datum, místo a okres úmrtí a u občana, který zemřel v cizině, pouze stát úmrtí.

občan (údaje o něm) – *prvek v tabulce*
rodné číslo - *klíč*

§ 13

(1) Rodné číslo je desetimístné číslo, které je dělitelné jedenácti beze zbytku. ...

Jaké má jméno občan s rodným číslem **9012150004*?***

velikost množiny klíčů $|K| = 10^{10} / 11$ = počet míst v tabulce
(velikost pole `t[]`)

index v tabulce = rodné číslo / 11

velikost aktuální množiny občanů $|A| \approx 10\,000\,000$

„využití“ míst v tabulce občanů s přímým adresováním

$|A| / |K| = 1.1\%$

- přímé adresování nevhodné, je-li množina klíčů *K* rozsáhlá

a může být hůř ...

Informační systém klientů banky

BANKA	ROK 2000	ROK 2006
Finanční skupina ČS	4 755 000[1)]	5 277 000
KB	1 246 000	1 467 000[2)]
GE Money Bank	470 060	844 214
eBanka	30 700	119 800
HVB Bank	57 000[3)]	111 000
Volksbank	16 588	35 716
Poznámky: [1)] údaj za rok 2001, [2)] údaj za rok 2005, [3)] údaj za rok 2002		
Zdroj: ČTK		

klient – *prvek v tabulce*
rodné číslo – *klíč*

uvažujme 100 000 klientů $|\boldsymbol{A}| = 100\,000$

„využití“ míst v tabulce klientů s přímým adresováním

$|\boldsymbol{A}| / |\boldsymbol{K}| = 0.011\%$

Rozptylová (hash) tabulka

Problém

je-li $|A| << |K|$
za přístup ***O(1)*** k prvkům (nalezení, vložení, vybrání)
platíme nízkým využitím paměti $|A| / |K|$

Řešení

namísto **`t.length`** = $|K|$

volíme **`t.length`** ≈ $|A|$

využití paměti kolem 100%

- označme ***m*** **= `t.length`,** indexy 0, 1, ... , m-1, ***m*** ≈ $|A|$

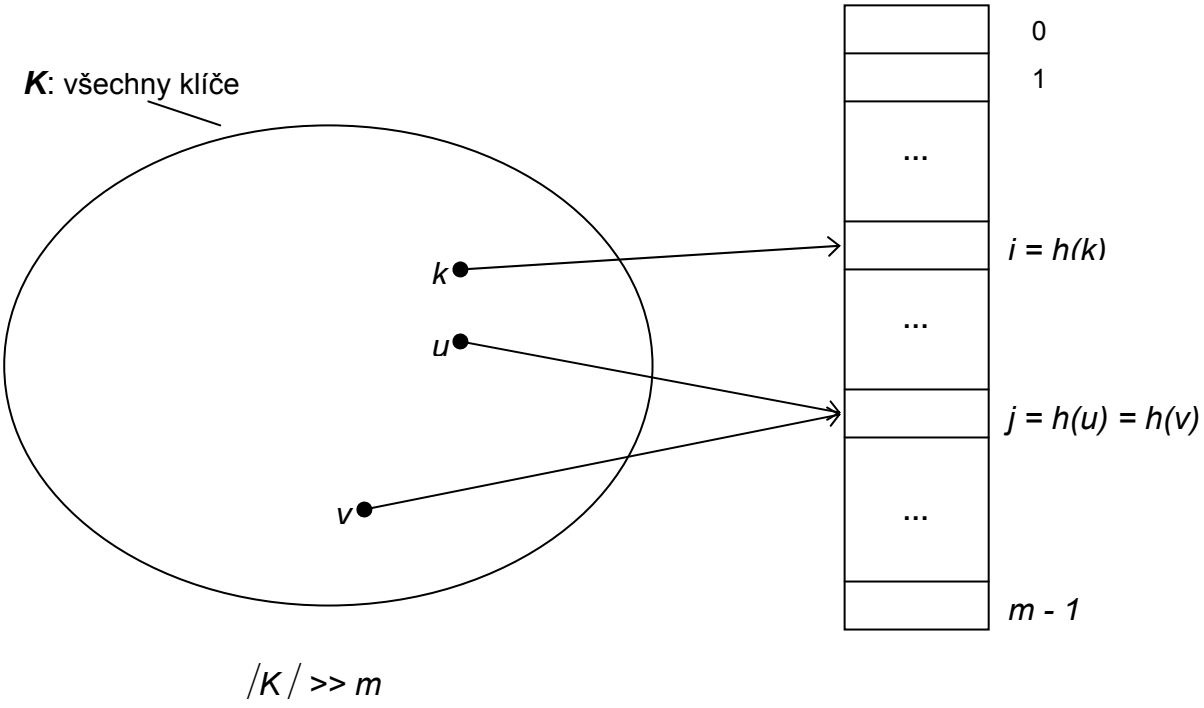
- neznáme hodnoty klíčů v $A \subseteq K$ **,** nejspíš nebudou 0, 1, ... , m-1

- pro přístup ***O(1)*** musíme určit hodnotu indexu v tabulce **`t[]`**
  pro každý klíč $k \in K$:

$$h: K \rightarrow \{0,1, \ldots m-1\}$$

h - rozptylovací funkce (hash function)

h(k) - index klíče ***k*** v tabulce



$m \approx |A| << |K|$

rozptylová funkce **může** zobrazit alespoň dva různé klíče na stejný index - **kolize**

pro $u \neq v$ a $u,v \in K$ bude $h(u) = h(v)$

ideálně

všechny klíče prvků množiny ***A*** jsou zobrazeny **na různé hodnoty indexů**, musí být $m \geq |A|$

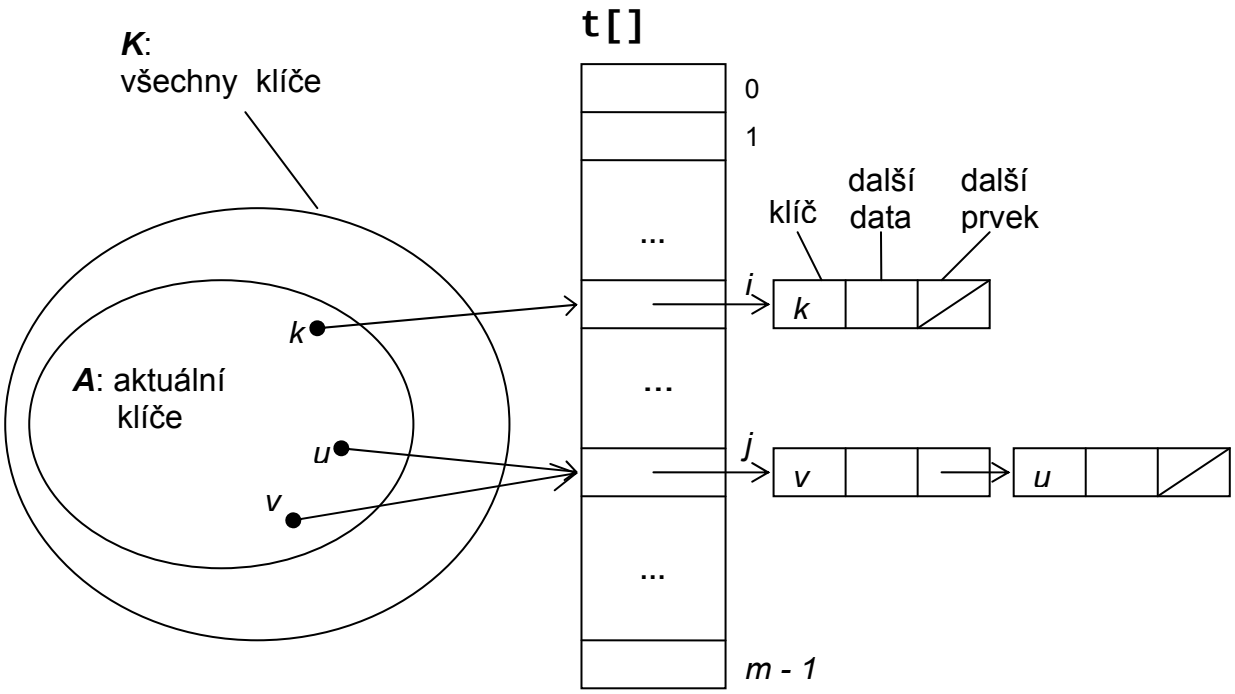
nejhorším případ

všechny klíče množiny *A* prvků jsou zobrazeny **na jeden index**

Řešení kolize

- vnější řetězení *(separate chaining), ...*

vytvoří se seznam prvků, jejichž klíče jsou zobrazeny na stejnou pozici v tabulce (index)



h(k) = j
h(u) = h(v) = j

Implementace

```
class Prvek {
  int klic;
  String data;
  Prvek dalsi;
  Prvek predch; //ulehčí výběr prvku

  Prvek(int klic, String data) {
    this.klic = klic;
    this.data = data;
  }
}
```

operace **hledej, vloz, vyber** jsou převedeny na operace nad seznamem s hlavičkou **t[h(k)]**

```
  Prvek hledej(int k) {
    hledej prvek s klíčem k v seznamu t[h(k)]
  }

  void vloz (Prvek x) {
    vlož  prvek x  na začátek seznamu t[h(x.klic)]
  }

  void vyber (Prvek x) {
    vyber prvek x ze seznamu t[h(x.klic)]
  }
```

Operace vložení prvku do tabulky je ***O(1).***

- prvek vkládáme **na začátek** odpovídajícího seznamu
- výhodné pro aplikace s vnořenými strukturami

```
{int identifikator1 ...
  {int identifikator2 ...
  } ...
}
```

Operace odebrání prvku je *O(1)*.

Operace hledej vyžaduje analýzu – kolik je prvků v seznamu před hledaným klíčem.

Analýza

Vznik kolizí

Pro $|A| = 1$, kolize nemůže vzniknout.

Pro $|A| \leq m$,
- kolize nemusí vzniknout
- všechny prvky mohou být vloženy **na stejnou pozici** v tabulce a **vytvoří seznam délky** $|A|$.

Pro $|A| > m$, bude alespoň jeden seznam obsahovat více než jeden prvek a kolize musí nastat.

Dirichletův princip *(Johann Peter Gustav Lejeune Dirichlet, 1834):*
Pokud do m přihrádek umístíme n > m kuliček, alespoň v jedné přihrádce musí být více než jedna kulička. (Důkaz sporem.)

Zkoumejme $|A| \leq m$

Jaká je **pravděpodobnost vzniku kolize** pro $1 < |A| \leq m$?

Předpoklad – jednoduché rovnoměrné rozptylování

Možnost umístnění libovolného prvku do kterékoliv pozice v tabulce je stejná a nezávislá od umístnění ostatních prvků.

Narozeninový problém (Richard von Mises, 1939)

***Kolik lidí se musí nacházet v místnosti, aby ignorujíc 29. únor alespoň dva z nich měli narozeniny ve stejný den roku s pravděpodobností* 50%.**

K – rodná čísla
A – rodná čísla lidí v místnosti
m = 365

h(rodné číslo) = den narození v roku (*1 – 365*)

Q(n) - pravděpodobnost, že ve skupině ***n*** lidí **žádní dva nemají narozeniny** ve stejný den

n=2,
dva lidé nemají narozeniny ve stejný den, jsou-li **narozeniny druhého** z nich v jednom **ze zbývajících 364 dnů**

$Q(2) = 364/365$

n=3,
tři lidé mají narozeniny v různé dny, když dva mají narozeniny v různé dny a třetí má narozeniny v jednom ze zbývajících 363 dnů

$Q(3) = Q(2) * 363/365 = Q(2) * (365 – 2)/365$
...
$Q(n) = Q(n – 1) * [365 – (n – 1)] / 365$

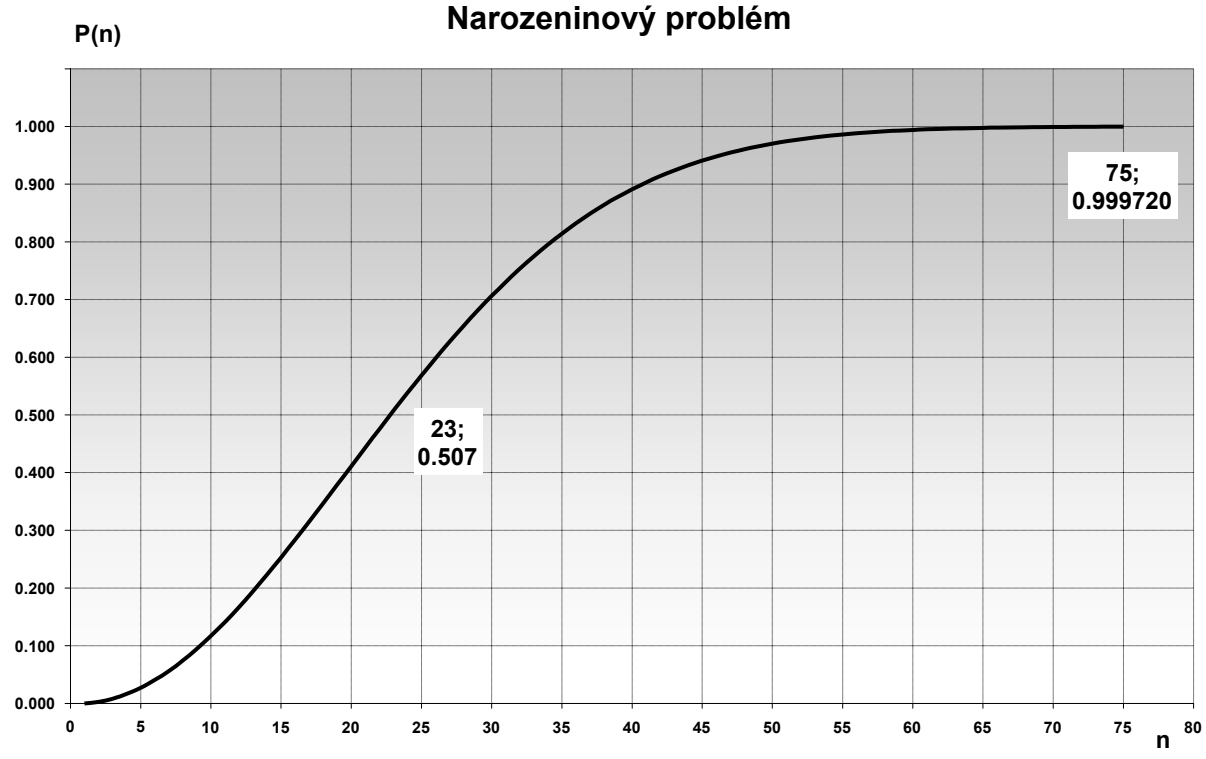
$$Q(n) = \frac{364*363*...*[365 – (n – 1)]}{365^{n – 1}} = \frac{365!}{365^{n}\,(365 – n)!}$$

Pravděpodobnost *P(n)*, že ve skupině *n* lidí alespoň dva z nich mají narozeniny ve stejný den je

$$P(n) = 1 - Q(n) = 1 - \frac{365!}{365^{n}\,(365-n)!}$$

Pro jaké n je $P(n) = 0.5$ **?**

n	*P(n)*
5	0.027
10	0.117
15	0.253
20	0.411
22	0.476
23	0.507
25	0.569
30	0.706
40	0.891
50	0.970
60	0.994



n = 23

- pro velikost rozptylové tabulky *m* = *365*, je 50% pravděpodobnost vzniku kolize již po vložení 23 prvků

- to je 6,3% z počtu kdy kolize vzniknout musí, tj. 366 aktuálních prvků

- kolize budou poměrně časté, i když velikost aktuální množiny prvků $|A|$ bude o dost menší než velikost tabulky ***m***

Na zabránění kolizím, nemá význam volit velké *m* vzhledem k $|A|$.

Zkoumejme $|A| > m$

- kolize musí vzniknout

Jaký je průměrný počet vložených prvků, když už při každém dalším vložení vznikne kolize?

- v každém seznamu **`t[i]`**, i=*0,1,* ..., *m-1,* bude alespoň jeden prvek

Problém sběratele kupónů

Mějme neomezené množství kuponů m typů. Vybereme-li náhodně vždycky jeden kupón, jaká je průměrná hodnota počtu výběrů W na to, abychom získali alespoň jeden kupón z každého typu.

kupón – prvek
typ kupónu – index

máme-li alespoň jeden kupón každého typu, v každém seznamu **`t[i]`**, i=*0,1,* ..., *m-1* je alespoň jeden prvek

- mějme získané kupony *i* různých typů

- pravděpodobnost, získání typu kupónu, který ještě **nemáme** je

$$p_i = \frac{m - i}{m} \qquad i = 0, 1, ..., m-1$$

- pravděpodobnost *P(k)* získání kupónu, který ještě **nemáme** po *k* výběrech, je dána součinem pravděpodobností *k-1* neúspěšných výběrů a pravděpodobnosti úspěšného výběru

$$P(k) = p_i(1 - p_i)^{k-1}$$

- kupón typu, který ještě **nemáme**, získáme průměrně po W_i výběrech

$$W_i = \sum_{k=1}^{\infty} k\, p_i(1 - p_i)^{k-1} = - p_i \Big[\sum_{k=0}^{\infty} (1 - p_i)^k \Big]' = 1/ p_i = m / (m-i)$$

Poznámky:

$$\sum_{k=0}^{\infty} q^k = 1/(1-q)$$

$$(1/q)' = -1/q^2$$

- průměrný počet výběrů kuponů na získání všech ***m*** typů je

$$W = \sum_{i=0}^{m-1} W_i = m \sum_{i=0}^{m-1} 1 / (m-i) = m \sum_{j=1}^{m} 1 / j$$

- W je průměrný počet vložených prvků, kdy už v každém seznamu `t[i]`, i = *0,1,* ..., *m-1* bude alespoň jeden prvek a při každém dalším vložení prvku vznikne kolize

- pro m = 365 je W = 2364

- kolize vzniká při každém vložení prvku po průměrném vložení 6.5 krát více prvků než je pozic v tabulce

Situace se dramaticky nezhorší, stane-li se *m* „rozumně“ menší než $|A|$.

Časová náročnost operace hledání

m – velikost tabulky
n = $|A|$

α = n/m - koeficient obsazení („využití“)

d_i - délka seznamu **`t[i]`**, i = *0,1,* ..., *m-1*

$n = d_0 + d_1 + \ldots + d_{m-1}$

průměrná délka ***α = n/m***

Čas hledání prvku v tabulce závisí na délce prohledávaného seznamu.

Neúspěšné hledání

- klíč se může se stejnou pravděpodobností zobrazit **na libovolnou pozici**

- průměrná doba hledání v seznamu je stejná a **rovna *α***

- celková doba hledání včetně času výpočtu rozptylovací funkce a přístupu k seznamu je ***Θ(1 + α)***

Úspěšné hledání

- nechť byl hledaný prvek vložen jako *j-tý* v pořadí z *n* vložených prvků, *j =1, 2, …, n*

- po něm bylo vloženo *n-j* prvků

- klíč každého z nich se zobrazil na libovolnou pozici v tabulce se stejnou pravděpodobností *1/m*

- průměrná délka seznamů, které takto vznikly z uvažovaných *n-j* prvků, bude tedy *(n-j)/m*

- při hledání projdeme tento seznam + hledaný prvek

Průměrný počet prvků, které nutno projít pro libovolný vložený prvek je

$$1/n \sum_{j=1}^{n} [\ 1 + (n-j)/m] \ = 1 + 1/(nm)\sum_{j=1}^{n}(n-j) = 1 + 1/(nm)\left(\sum_{j=1}^{n} n - \sum_{j=1}^{n} j\right) =$$

$$1 + [n^2 - n(n+1)/2]/nm = 1 + (n-1)/2m = 1 + \alpha/2 - \alpha/2n.$$

Čas hledání existujícího prvku v tabulce včetně času výpočtu rozptylovací funkce a přístupu k seznamu je

Θ(2 + α/2 – α/2n) = Θ(1 + α)

Je-li počet prvků ***n*** aktuální množiny ***A*** úměrný počtu pozic v tabulce ***m***, ***m*** ≈ $|A|$, tj. ***n/m = konstanta***, potom je **průměrný čas hledání prvku** v tabulce ***O(1)***.

Známe-li, nebo umíme alespoň odhadnout, počet prvků *n*, které chceme uložit v tabulce, jak zvolíme její velikost *m*?

- pokud je kritická rychlost hledání volíme *m* > *n*, za cenu větších nároků na paměť, když alespoň *m-n* pozic v tabulce bude nevyužitých

- v opačném případě volíme ***m* < *n*, kdy se úměrně prodlouží průměrný čas hledání**

Obecně se **doporučuje zvolit *m* = *n*** a i když nakonec bude nutno vložit více prvků, vlastnosti se výrazně nezhorší.

Poznámka:

JAVA

Rozptylovací funkce

h: K → {0,1, ... m -1}

- očekáváme, že **každý klíč zobrazí (přibližně) stejně pravděpodobně na libovolnou z *m* hodnot**, nezávisle na tom, kam je zobrazen jakýkoliv jiný klíč

- musíme znát charakteristiku výskytu klíčů v aplikaci, což není vždycky splněno

- **klíče mohou mít různý typ**, od jednoduchých datových typů až ke strukturovaným datovým typům, jakými jsou například objekty

- každý typ klíče je v počítači reprezentován **řetězcem 0** a **1**, který můžeme vyjádřit j**ako** (možná velice veliké) **celé číslo**

Modulární metoda

klíčem je číslo

- hodnoty ***h(k) = k mod m*** (v Javě **k % m**) jsou právě z množiny {0,1, ... m -1}.

- *m* **nemůžeme volit libovolně,** je-li $m = 2^p$, potom *h(k)* závisí jenom na *p* nejnižších bitech klíče *k*

- pokud nevíme, že všechny *p* - bitové permutace nejnižších bitů jsou stejně pravděpodobné, volíme jako *m* prvočíslo, které není blízké mocnině 2

- nastáva například při správě paměťových struktur operačního systému, kterých velikost je obvykle 2^p

klíčem je řetězec ASCII znaků

- ASCII kód zobrazuje znak řetězcem 7 bitů, rozeznává tedy 128 různých hodnot, které můžeme interpretovat jako celá čísla 0 až 127

- řetězec $\alpha_1\alpha_2...\alpha_n$ potom můžeme interpretovat jako zápis čísla se základem 128, kterého hodnota je

$$z_1.128^{n-1} + z_2.128^{n-2} + ... + z_n.128^0$$

kde z_i je celočíselná interpretace znaku α_i

- řetězce znaků mohou být tak dlouhé, že uvedená číselná reprezentace přesáhne rozsah zobrazitelných celočíselných hodnot
- potřebujeme

$$h(\alpha_1\alpha_2...\alpha_n) = (z_1.128^{n-1} + z_2.128^{n-2} + ... + z_n.128^0) mod\ m$$

- užitím Hornerova schématu pro výpočet $h(\alpha_1\alpha_2...\alpha_n)$ dostaneme

$$h(\alpha_1\alpha_2...\alpha_n) = (z_1.128^{n-1} + z_2.128^{n-2} + ... + z_n.128^0) \bmod m =$$

$$(((z_1.128 + z_2).128 + ... + z_{n-1}).128 + z_n) \bmod m$$

- využijeme toho, že pro výpočet *mod m* nebudeme uvažovat násobky *m* ve „velkých“ členech, tj.

$$(x.c + y) \bmod m = ((x \bmod m).c + y) \bmod m$$

$$x = a.m + b$$

$$((a.m + b).c + y) \bmod m = (a.c.m + b.c + y) \bmod m = (b.c + y) \bmod m$$
$$(((a.m + b) \bmod m).c + y) \bmod m = (b.c + y) \bmod m$$

$$(((z_1.128 + z_2).128 + ... + z_{n-1}).128 + z_n) \bmod m =$$

$$((((0.128 + z_1) \bmod m .128 + z_2) \bmod m .128 + ... + z_{n-1}) \bmod m .128 + z_n) \bmod m$$

```
static int h(String klic, int m) {
  int h = 0;
  for (int i = 0; i < klic.length(); i++)
    h = (h * 128 + klic.charAt(i)) % m;
  return h;
}
```

Multiplikativní metoda

- celou část reálného čísla *r* označíme $\lfloor r \rfloor$

- je-li *c* reálné číslo z intervalu *0* ≤ *c* < *1*, potom $\lfloor m.c \rfloor \in$ *{0,1, ... m -1}*

- jsou-li hodnoty klíčů ***k*** omezeny, *a* ≤ *k* < *b*, potom

c = (k – a) / (b – a)

***K** = {1, 2, ..., 10 000}*
***m** = 100*

a = 1, b = 10 001
h(k) = $\lfloor m.c \rfloor = \lfloor m(k-1)/10000 \rfloor = \lfloor (k-1)/100 \rfloor$

klíče *k = 1, 2, ..., 100* jsou zobrazeny na pozici 0
klíče *k = 101, 102, ..., 200* jsou zobrazeny na pozici 1, atd.

Transformace klíče *k* na multiplikativní koeficient *c*, musí být „znáhodněna“ a přitom její výpočet musí zůstat efektivní.

- zvolíme **konstantu *A*** z intervalu *0 < A < 1*
- jako ***c*** vezmeme **zlomkovou část** součinu ***kA***, *c* = $kA - \lfloor kA \rfloor$

h(k) = $\lfloor m(kA - \lfloor kA \rfloor) \rfloor$

implementace

- zvolíme $m = 2^p$

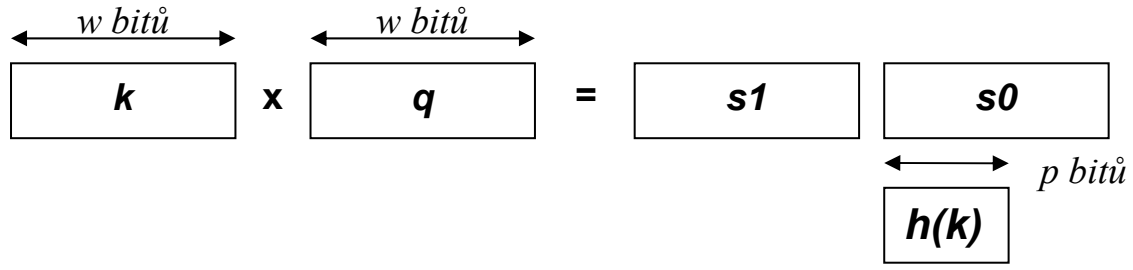
- nechť celočíselný klíč ***k*** je zobrazen slovem s ***w* bity**

- ***A*** zvolme ve tvaru $q/2^w$, ***q*** je celé číslo zobrazené také jako slovo s ***w* bity**, $0 < q < 2^w$, $k.A = k.q/2^w$

- celočíselným násobením ***k.q*** získáme hodnotu s ***2w*** bity uloženou ve dvou slovech ***s0*** a ***s1***, přičemž ***s0*** označuje slovo s bity nižších řádů součinu, $k.q = s1 \,.\, 2^w + s0$

- $k.A = k.q/2^w = (k.q) \,.\, 2^{-w} = s1 + s0 \,.\, 2^{-w}$, ***w*** bitů slova ***s0*** obsahuje **zlomkovou** část součinu ***kA***, tj. $kA - \lfloor kA \rfloor$

- ***h(k)*** = $\lfloor m(kA - \lfloor kA \rfloor) \rfloor = \left\lfloor 2^p (kA - \lfloor kA \rfloor) \right\rfloor$, ***h(k)*** tvoří ***p*** nejvýznamnějších bitů slova ***s0***



- jako konstanta ***A*** se doporučuje hodnota zlatého poměru $\varphi = 0.618033$
- ***q*** volíme tak, aby $q/2^w$ bylo co nejblíže φ, $q \approx 0.618033 \times 2^w$