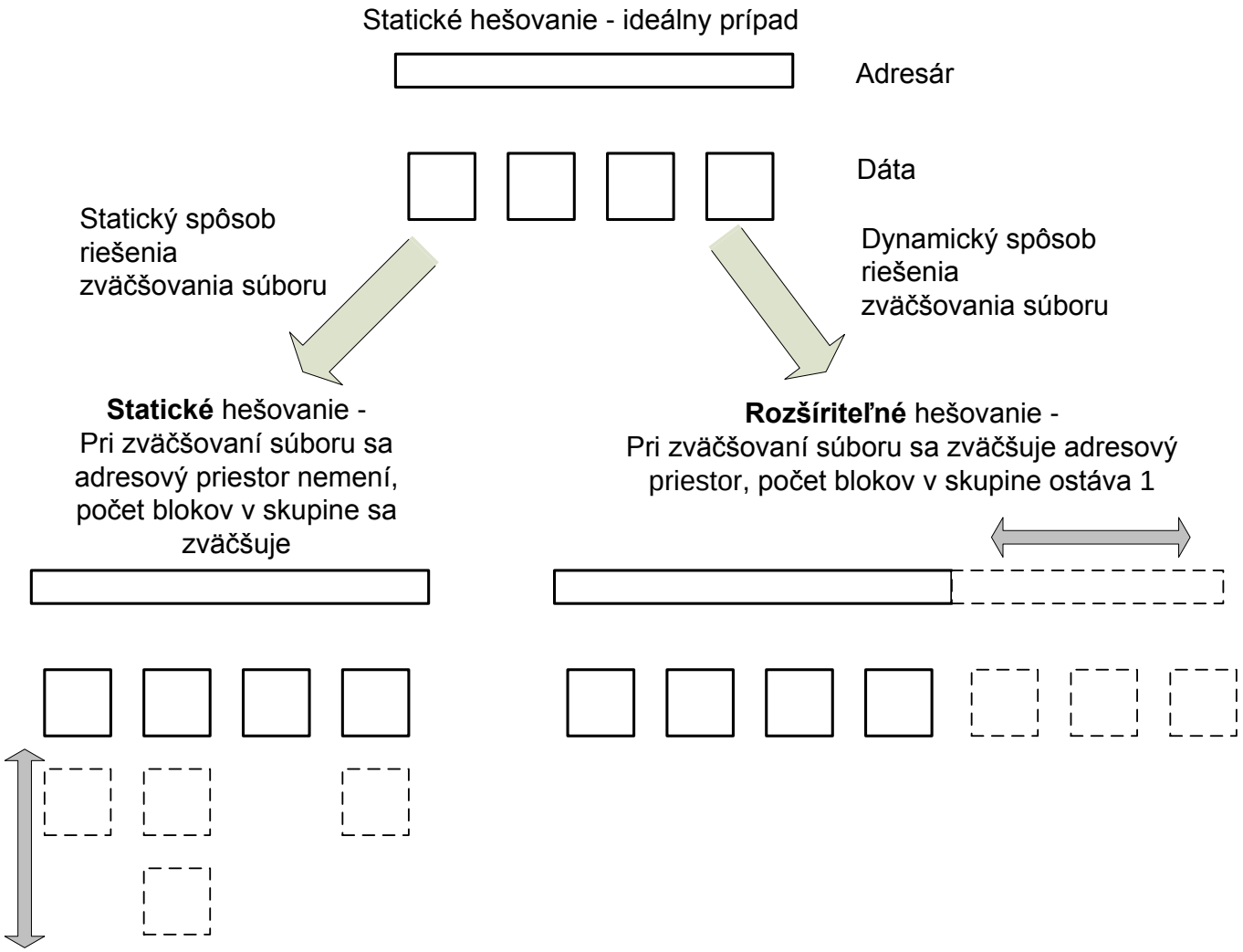
Rozšíriteľné (extendible) hešovanie

Obmedzenie doteraz známeho (statického) hešovania:

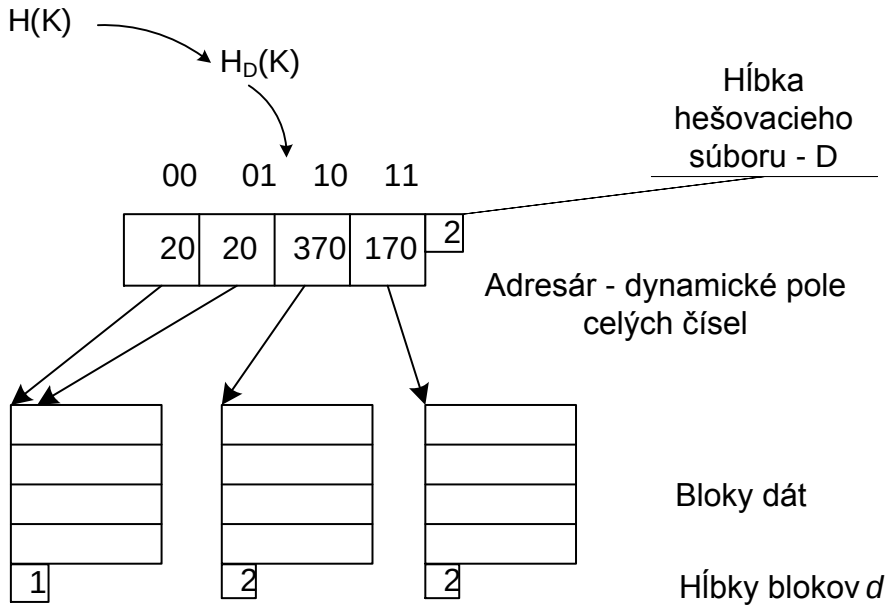
- **Vyhovuje iba pre súbory pevnej (vopred známej) veľkosti, lebo s touto veľkosťou je spätá hešovacia funkcia. Pri dynamickom raste efektívnosť degraduje, rekonštrukcia je ťažká až často aj nemožná.**
- **Pevná veľkosť súboru bez ohľadu na reálne vložené dáta.**

Myšlienka rozšíriteľného hešovania:

- pri raste a zmenšovaní súboru sa uloženie dát automaticky reorganizuje – veľkosť súboru sa mení podľa skutočne vložených dát
- zachováva okamžitý prístup k dátam v súbore (rovnako ako statické hešovanie)
- využitie iba istého počtu (postupne rastie) bitov z výsledku hešovacej funkcie
- použitie adresára blokov, ktorý sa nachádza v operačnej pamäti



Štruktúra:



Operácia nájdi záznam s kľúčom K:

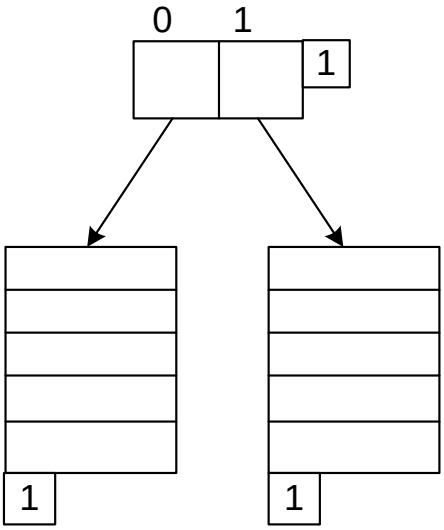
1. **Vypočítaj I = h_D (K) (prvých D bitov hodnoty hešovacej funkcie),**
2. **Pomocou adresára sprístupni blok P[i] (prvých D bitov I preveď na celé číslo, toto je index v adresári na ktorom sa nachádza adresa miesta v súbore, kde sa nachádza blok, ktorý by mal obsahovať hľadaný záznam),**
3. **V bloku P[i] nájdi záznam s kľúčom K.**

Adresár je krátky ➔ počas spracovania v operačnej pamäti ➔ vyhľadanie záznamu vyžaduje štandartne jeden blokový prenos (plus prípadný vyvolaný prenos z buffera do súboru). Adresár sa nachádza v operačnej pamäti. Je to jednorozmerné pole adries (celočíselných hodnôt).

Operácia vlož:

K	H(K)
Žilina	00000000
Košice	01100100
Martin	10010101
Levice	10111011
Trnava	10100101
Snina	10110110
Senica	10100000
Nitra	01101100
Poprad	00000000
Lučenec	01100100
Zvolen	11101001
Prešov	11110000
Púchov	10110111
Ilava	00001111
Brezno	00111100

Nech blokovací faktor *f* = 5.
V prázdnom súbore D = 1.
Tvar prázdneho súboru:



Základné idey:

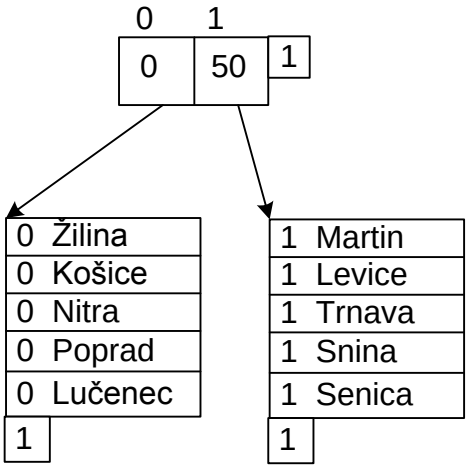
- **Ak sa blok preplní, nebuduj skupinu preplňujúcich blokov, namiesto toho zväčši adresový priestor tak, aby sa adresovali bloky a nie skupiny blokov.**
- **Pri preplnení alokuj nový blok a priraď mu adresu v adresári.**

- **Adresový priestor zväčši tak, že ho zdvojnásobíš (D = D + 1) a zaktualizuješ adresy v adresári. Zdvojnásobenie adresára znamená, že sa alokuje nové pole väčšie o 100% a každá hodnota z pôvodného poľa sa nakopíruje dva krát.**
- **V prípade, že dôjde k využitiu všetkých bitov z výsledku hešovacej funkcie a záznam nebolo možné vložiť dôjde ku kolízii. Na jej riešenie je možné použiť oblasť preplnenia blokov, alebo oblasť preplnenia súboru.**
- **Voľné bloky uprostred súboru je nutné využívať prednostne (rovnaký postup ako prázdne bloky v prelňujúcom súbore – pozri príslušný text o statickom hešovacom súbore).**

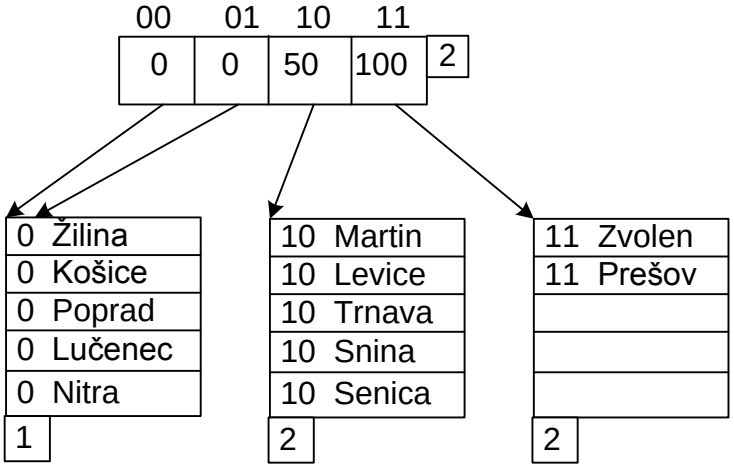
Algoritmus vkladania:

```
while niejevlozene do
begin
    VypočítajHash          // získame adresu bloku
    if BlokJePlný then
     begin
       if HĺbkaBloku=HĺbkaSúboru then
         begin
           ZdojnásobAdresár;
         end;
       RozdelenieBloku;  // Split – vytvorenie nového bloku
      end
    else
      begin
        VložZáznam;       //ukončenie cyklu while
      end;
  end;
```

Postup na príklade:

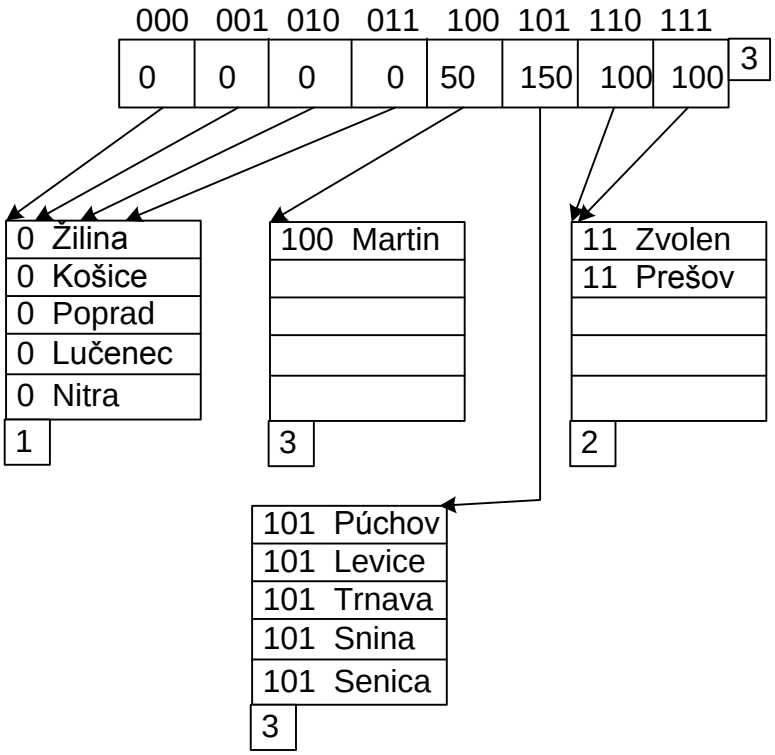


Stav po vložení prvých 10 záznamov. (Prefixy pred jednotlivými kľúčmi sú len ilustračné, nezapamätávajú sa ako súčasť záznamu). V adresári môžeme vidieť, že prvý blok začína na začiatku súboru a druhý na 50 bajte.



Teraz vložíme záznamy Zvolen, Prešov:

Po vložení Zvolen sa blok s adresou 1 sa rozdelí na dva (split) s adresami 10, 11. Adresár sa zdvojnásobí (D=2), adresy v adresári sa aktualizujú. Bloky po rozdelení majú hĺbku 2, prvý blok ostáva s hĺbkou 1 (obsahuje kľúče s prefixami 00 a 01.

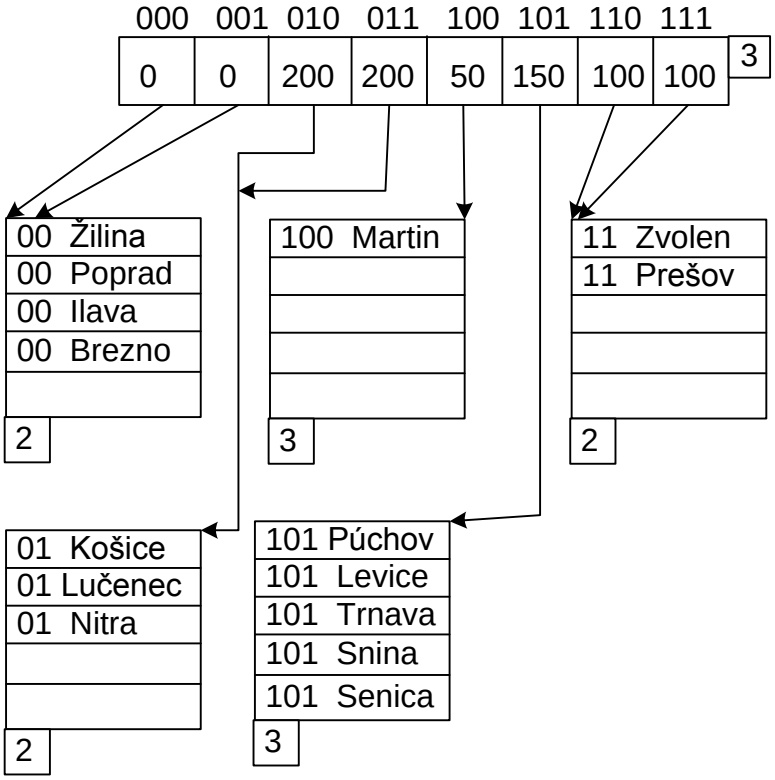


Vložíme Púchov:

Adresár sa zdvojnásobí, lebo vzrástla hĺbka súboru.

Blok s adresou 10 sa rozdelí po alokácii nového bloku na 100 a 101, každý s hĺbkou 3.

Upravia sa adresy blokov v adresári.



Nakoniec vložíme Ilava a Brezno:

Po vložení Ilava sa prvý blok zľava rozdelí a hĺbky dvoch vzniknutých blokov vzrastú na 2.

Hĺbka súboru nevzrástla, adresár sa nezdvojnásobí.

Vloženie Brezno je úplne bez reorganizácie.

Rekapitulácia vkladania do bloku b:

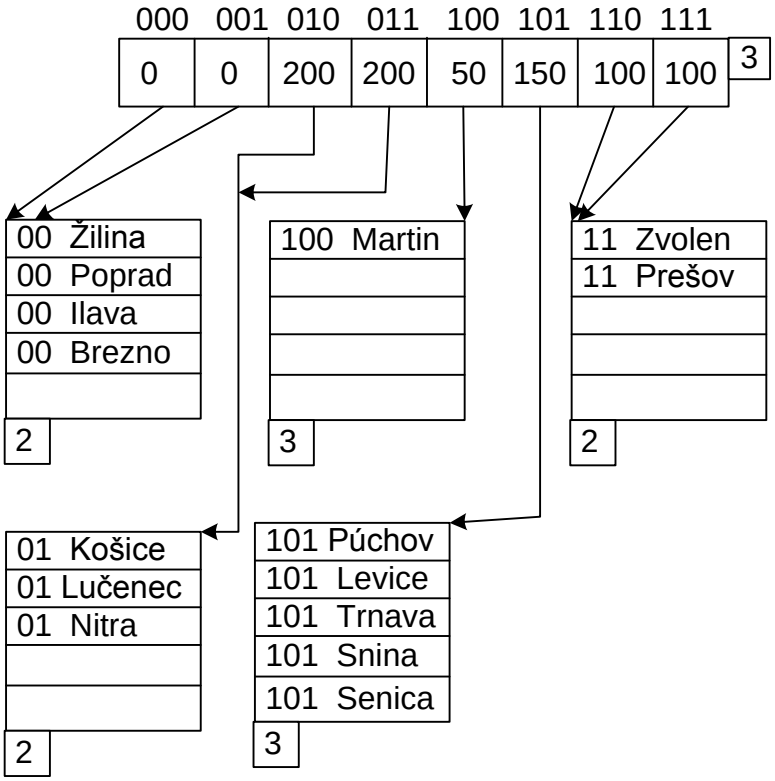
- **Nedošlo k preplneniu: vloží sa priamo.**
- **Došlo k preplneniu, ale d < D:**
  - **rozdeľ blok**
  - **vlož záznam**
  - **uprav adresy (lokálne) bez zväčšenia adresára**
- **Došlo k preplneniu a d = D:**
  - **zdvojnásob adresár**
  - **rozdeľ blok**
  - **vlož záznam**
  - **reorganizuj celý adresár**

<u>Operácia vymaž:</u>

Základné idey:

Ak po zrušení zostane v susedných blokov iba toľko záznamov, že sa zmestia do jediného bloku, presunú sa tam a voľný blok sa dealokuje. Bloku, ktorý ostal, sa zníži hĺbka. Ak to bol posledný blok s touto hĺbkou, znižuje sa hĺbka celého súboru a adresár sa zmenší na polovicu. Operácia prebieha cyklicky podobne ako vkladanie.

Algoritmus na príklade (stav ako na poslednom obrázku):



Pojem „sused“:

Susedom bloku B s hĺbkou d > 1 je blok s rovnakou hĺbkou pre ktorý platí, že prefixy záznamov tohto bloku sa zhodujú s prefixami bloku B v d-tom bite sprava.

Na obrázku sú susedia pod sebou. Posledný blok vpravo nemá suseda.

Zruš Poprad: po zrušení zostane spolu so susedom 6 > f záznamov ➔ bez reorganizácie.

Zruš Lučenec: (po zrušení Popradu): zostane spolu 5 = f záznamov ➔ spoja sa bloky a jeden sa dealokuje, zostávajúci bude mať hĺbku 1 a budú naňho ukazovať prvé 4 adresy adresára.

Zruš Zvolen: blok nemá suseda ➔ bez reorganizácie.

Zruš Prešov: blok nemá suseda s rovnakou hĺbkou ➔ bez reorganizácie a dealokácie.

Zruš Levice: spojenie blokov (100 a 101) a dealokácia voľného; hĺbka ostávajúceho klesne na 2; už nezostal žiaden s hĺbkou 3 ➔ mení sa stupeň súboru na 2, adresár sa zmenší na polovicu a nastavia sa jeho hodnoty.

<u>Efektívnosť operácií:</u>

Vyhľadanie: 1 prenos.

Vloženie: 1 prenos, v prípade preplnenia 2 prenosy.

Zrušenie: väčšinou 2 prenosy, pretože sa skúma i sused.

- **v oboch prípadoch prípadný zápis buffera pred prenosom naviac.**

Celkovo operácie: 1 - 3 prenosy (mimoriadne dobré).

Nevýhoda: pamäťová náročnosť na externú pamäť (pohybuje sa medzi 0.53 a 0.94, stredné využitie 69%).

Organizácia dát v bloku: ľubovoľná implementácia tabuľky v internej pamäti

Iný príklad:

K	H(K)=K mod 32	
	DEC	BIN
34	2	00010
24	24	11000
39	7	00111
46	14	01110
70	6	00110
7	7	00111
44	12	01100
60	28	11000
25	25	11001
61	29	11101