Rozšiřitelné hašování (Extendible Hashing)

Standardní metoda hašování vyžaduje na začátku zvolit velikost hašovací tabulky. Pokud ji zvolíme nedostatečnou, u otevřeného adresování dojde k vyčerpání kapacity tabulky, u zřetězení sice tento problém nenastává, ale roste časová složitost vyhledání prvku. Metoda hašování nazývaná *extendible hashing* nevyžaduje počáteční stanovení velikosti hašovací tabulky a navíc poskytuje konstantní časovou složitost vyhledání prvku bez ohledu na to, kolik prvků je v hašovací struktuře uloženo.

Hašovací datová struktura se skládá ze dvou částí:

- adresáře
- přihrádek

Adresář je pole odkazů na přihrádky. Velikost pole adresáře je 2^d odkazů. Hodnota *d* se dynamicky mění v závislosti na počtu prvků uložených v hašovací struktuře. Na začátku je zpravidla *d* malé. Při postupném zvyšování počtu ukládaných prvků do hašovací struktury se v určitých okamžicích hodnota *d* zvyšuje.

Přihrádka je místo pro uložení pevně stanoveného počtu prvků. Velikost všech přihrádek je stejná (například pro 4 prvky).

Základem metody je opět hašovací funkce, která datový prvek zobrazí na celé číslo. Posledních *d* bitů tohoto čísla je indexem v poli adresáře, kde najdeme odkaz na přihrádku, ve které je uložen daný prvek.

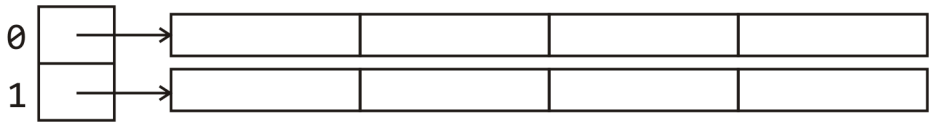
Příklad. Máme navrhnout rozšiřitelné hašování pro uložení řetězců. Hašovací funkci zvolíme:

$$h(z_1 z_2 ... z_k) = 7 * asc(z_1) + 3 * asc(z_2) + asc(z_k) + k \quad .$$

Velikost přihrádek zvolíme 4 prvky.

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Na začátku zvolíme velikost adresáře 2 (d=1).



Do hašovací struktury uložíme jména

h(*Eva*) = 7*69+3*118+97+3 = 937 = 1110101001

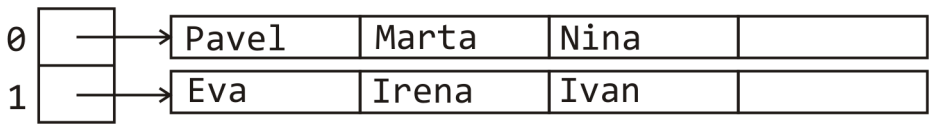
h(*Irena*) = 7*73+3*114+97+5 = 955 = 1110111011

h(*Pavel*) = 7*80+3*97+108+5 = 964 = 1111000100

h(*Marta*) = 7*77+3*97+97+5 = 932 = 1110100100

h(*Ivan*) = 7*73+3*118+110+4 = 979 = 1111010011

h(*Nina*) = 7*78+3*105+97+4 = 962 = 1111000010



Vyhledání prvku v hašovací struktuře

Jednotlivé bity hodnoty hašovací funkce si označíme:

… $b_{d+1}b_d b_{d-1}$ … $b_3 b_2 b_1$ (b_1 je nejméně významný bit)

◊ Vypočítáme hodnotu hašovací funkce hledaného prvku.

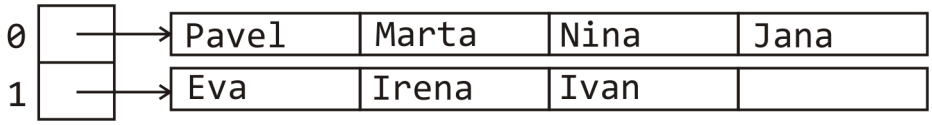
◊ Vezmeme posledních d bitů $b_d..b_1$ vypočtené hodnoty hašovací funkce. Ty jsou indexem v poli adresáře. V prvku pole adresáře, který odpovídá tomuto indexu, zjistíme odkaz na příslušnou přihrádku.

◊ V přihrádce procházíme v ní uložené prvky a srovnáváme je s hledaným prvkem, dokud nenalezneme prvek shodný s hledaným prvkem nebo je všechny neprojdeme (pak hledaný prvek není nalezen).

Přidání prvku do hašovací struktury

- Vypočítáme hodnotu hašovací funkce přidávaného prvku.
- Vezmeme posledních d bitů $b_d..b_1$ vypočtené hodnoty hašovací funkce. Ty jsou indexem v poli adresáře. V prvku pole adresáře, který odpovídá tomuto indexu, zjistíme odkaz na příslušnou přihrádku.
- Je-li v přihrádce místo pro uložení přidávaného prvku, prvek do přihrádky vložíme. Jinak je nutné přihrádku rozdělit.

h(*Jana*) = 7*74+3*97+97+4 = 910 = 1110001110

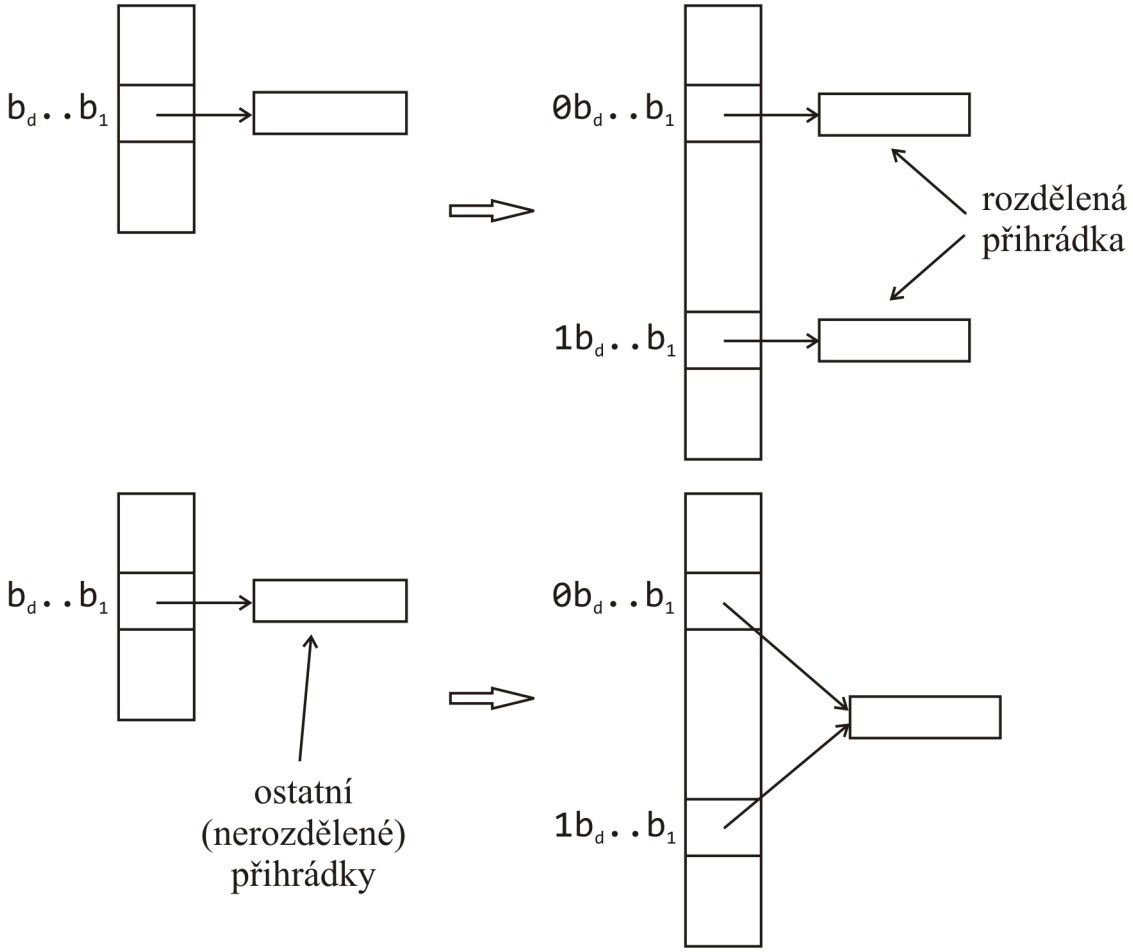


h(*Hana*) = 7*72+3*97+97+4 = 896 = 1110000000

Toto jméno se už do přihrádky nevejde, je nutné ji rozdělit.

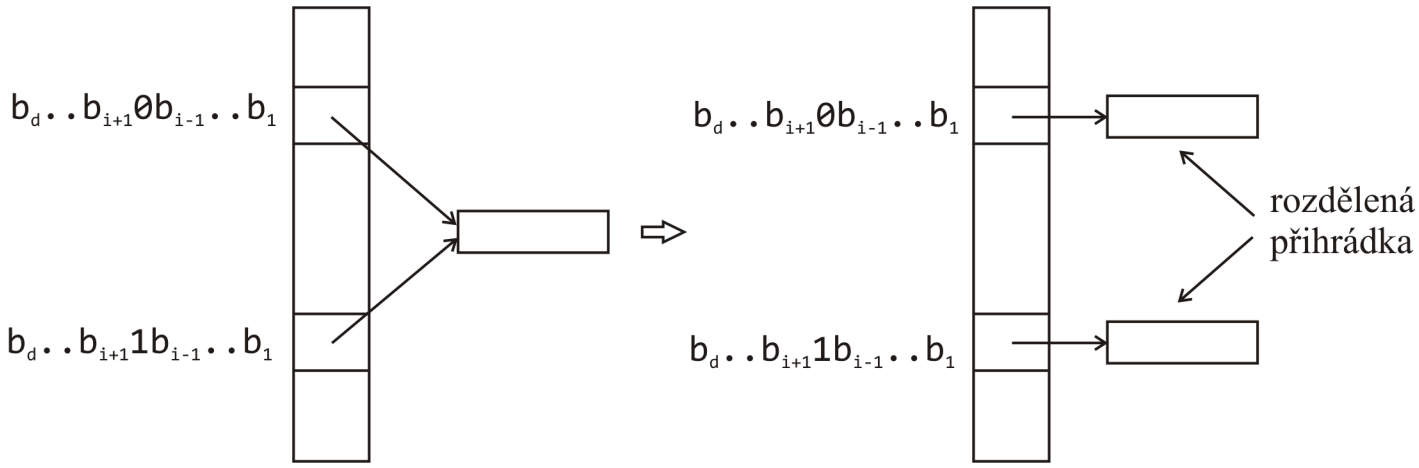
Rozdělení přihrádky

- Pokud mají všechny prvky uložené v přihrádce stejnou hodnotu posledních d bitů $b_d..b_1$ hodnot svých hašovacích funkcí, přihrádku rozdělíme na dvě a do první dáme prvky, jejichž hašovací funkce má hodnotu dalšího bitu b_{d+1} rovnu 0. Do druhé dáme prvky, jejichž hodnota bitu b_{d+1} je 1. Po rozdělení přihrádek následně i zdvojnásobíme velikost adresář (hodnotu d zvýšíme o 1). Při zdvojnásobení velikosti adresáře zachováme všechny odkazy na přihrádky, které se nerozdělily.



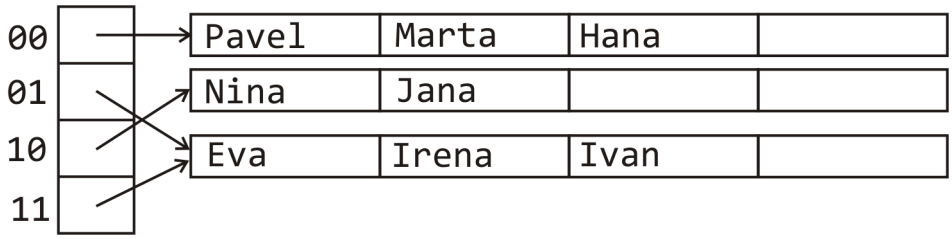
Pokud by bit b_{d+1} byl u všech prvků stejný, vzali bychom další bit b_{d+2} a adresář bychom zvětšili čtyřikrát (hodnotu d bychom zvýšili celkově o 2). Atd.

- Jestliže prvky uložené v přihrádce nemají shodnou hodnotu posledních d bitů hodnot svých hašovacích funkcí, rozdělíme je do dvou přihrádek podle prvního (bráno zleva) z d bitů, který není u všech prvků identický. Nechť je to bit b_i, kde $i \leq d$ a i je největší index takový, že hodnota tohoto bitu hašovacích funkcí všech prvků není stejná. Následně v adresáři příslušně upravíme odkazy na tyto přihrádky.



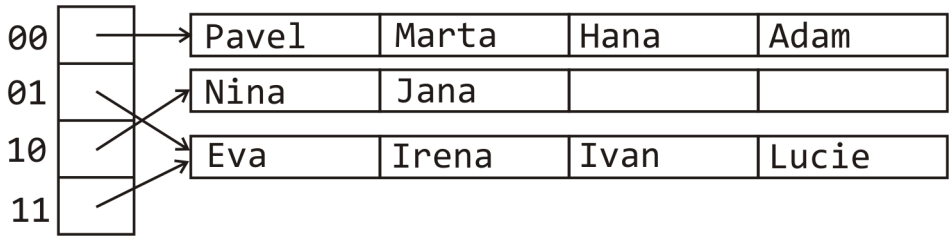
Příklad – pokračování.

Pavel	1111000100
Marta	1110100100
Nina	1111000010
Jana	1110001110
Hana	1110000000



h(*Lucie*) = 7*76+3*117+101+5 = 989 = 1111011101

h(*Adam*) = 7*65+3*100+109+4 = 868 = 1101100100



h(*Jitka*) = 7*74+3*105+97+5 = 868 = 1110100111

Eva	1110101001
Irena	1110111011
Ivan	1111010011
Lucie	1111011101
Jitka	1110100111

h(*Marek*) = 7*77+3*97+107+5 = 942 = 1110101110

h(*Martin*) = 7*77+3*97+110+6 = 946 = 1110110010

h(*Lenka*) = 7*76+3*101+97+5 = 937 = 1110101001

h(*Radek*) = 7*82+3*97+107+5 = 977 = 1111010001

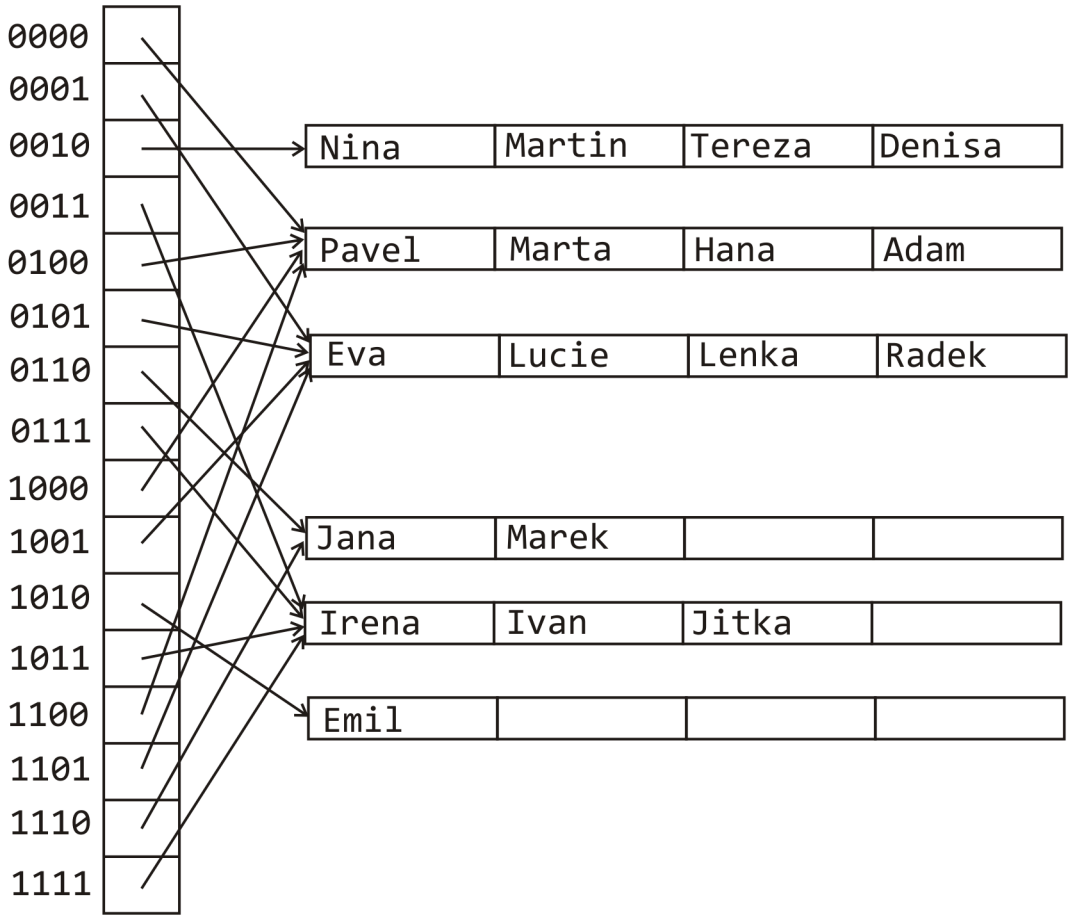
h(*Tereza*) = 7*84+3*101+97+6 = 994 = 1111100010

Nina	1111000010
Jana	1110001110
Marek	1110101110
Martin	1110110010
Tereza	1111100010

h(*Denisa*) = 7*68+3*101+97+6 = 882 = 1101110010

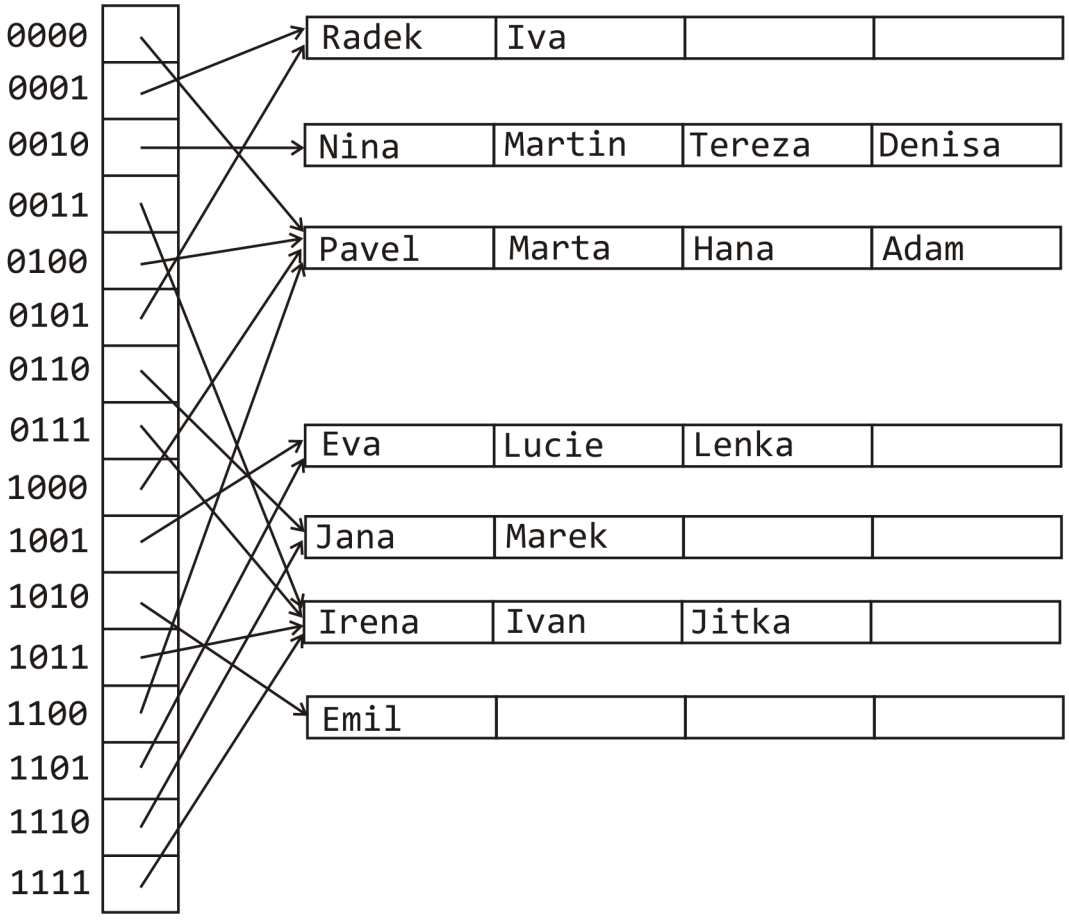
h(*Emil*) = 7*69+3*109+108+4 = 922 = 1110011010

Nina	1111000010
Martin	1110110010
Tereza	1111100010
Denisa	1101110010
Emil	1110011010



h(*Iva*) = 7*73+3*118+97+3 = 965 = 1111000101

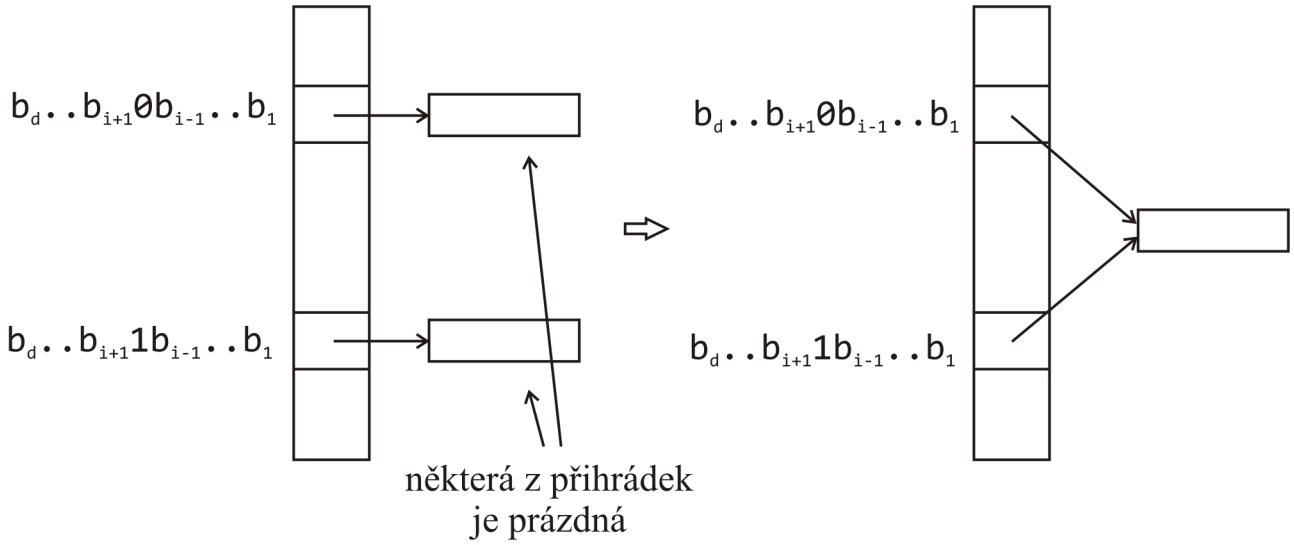
Eva	1110101001
Lucie	1111011101
Lenka	1110101001
Radek	1111010001
Iva	1111000101



Odebrání prvku z hašovací struktury

- Vyhledáme odebíraný prvek.
- Pokud byl nalezen, odstraníme ho z dané přihrádky.

Při větším odebírání, kdy vznikne více prázdných přihrádek, lze některé přihrádky sloučit. Případně po větším sloučení je někdy možné zmenšit velikost adresáře na polovinu. Jednoduchým způsobem lze sloučit prázdnou přihrádku, na kterou je jen jeden odkaz z adresáře, s přihrádkou, na kterou je opět jen jeden odkaz z adresáře a zároveň hodnoty indexů těchto odkazů v adresáři se liší jen v jednom bitu.



Bylo by možné sloučit i v případě, že na obě slučované přihrádky je stejný počet 2^m odkazů a přitom hodnoty indexů jednotlivých odkazů v adresáři se liší právě v m bitech.
V praxi se ale při odebírání zmenšování rozsahu adresáře nebo slučování přihrádek nedělá.

Složitost hašování

Složitost operace vyhledání je dána velikostí přihrádek. Vyhledání prvku vyžaduje výpočet jedné hodnoty hašovací funkce a nejvýše tolik srovnání hledaného prvku s jinými prvky, kolik je kapacita přihrádek. Složitost operace přidání prvku závisí na tom, zda dojde k dělení přihrádek a jak rozsáhlé dělení to je. Závisí to na volbě vhodné hašovací funkce. Složitost operace odebrání je stejná jako složitost operace vyhledání (pokud bychom nedělali slučování přihrádek).

Uložení na vnější paměti

Použití této metody je vhodné i v případech, kdy standardní hašovací tabulka by byla tak rozsáhlá, že by se nevešla do paměti. U rozšiřitelného hašování stačí, aby v paměti byl uložen jen adresář. Přihrádky mohou být uloženy na vnější paměti.