Níže uvedené úlohy představují přehled otázek, které se vyskytly v tomto nebo v minulých semestrech ve cvičení nebo v minulých semestrech u zkoušky. Mezi otázkami semestrovými a zkouškovými není žádný rozdíl, předpokládáme, že připravený posluchač dokáže zdárně zodpovědět většinu z nich.

Tento dokument je k dispozici ve variantě převážně s řešením a bez řešení.

Je to pracovní dokument a nebyl soustavně redigován, tým ALG neručí za překlepy a jazykové prohřešky, většina odpovědí a řešení je ale pravděpodobně správně :-).

-------------------------------------- HASHING GENERAL ---

1.
Hashovací (=rozptylovací) funkce

a) převádí adresu daného prvku na jemu příslušný klíč
b) vrací pro každý klíč jedinečnou hodnotu
c) pro daný klíč vypočte adresu
d) vrací pro dva stejné klíče různou hodnotu

Toto je elementární otázka z rozptylování. Hashovaci funkce podle své definice provádí činnost popsanou ve variantě c). Vytváří synonyma, takže varianta b) neplatí, a pro dva sejné klíče musí vrátit stejnou hodnotu, takže ani d) neplatí. Varianta a) obsahuje jen klíčová slova beze smyslu naházená do věty — neplatí také.

2.
Kolize u hashovací (rozptylovací) funkce h(k)
a) je situace, kdy pro dva různé klíče k vrátí h(k) stejnou hodnotu
b) je situace, kdy pro dva stejné klíče k vrátí h(k) různou hodnotu
c) je situace, kdy funkce h(k) při výpočtu havaruje
d) je situace, kdy v otevřeném rozptylování dojde dynamická paměť

Definitorická otázka, viz přednášky/literaturu. Platí varianta a).

3.
Hashovací (=rozptylovací) funkce

e) převádí adresu daného prvku na jemu příslušný klíč
f) vrací pro každý klíč jedinečnou hodnotu
g) pro daný klíč vypočte adresu
h) vrací pro dva stejné klíče různou hodnotu

Toto je elementární otázka z rozptylování. Hashovaci funkce podle své definice provádí činnost popsanou ve variantě c). Vytváří synonyma, takže varianta b) neplatí, a pro dva sejné klíče musí vrátit stejnou hodnotu, takže ani d) neplatí. Varianta a) obsahuje jen klíčová slova beze smyslu naházená do věty — neplatí také.

-------------------------------------- HASHING CHAINED ---

4.
Implementujte operace Init, Search, Insert a Delete pro rozptylovací tabulku se zřetězeným rozptylováním, do níž se ukládají celočíselné klíče. Předpokládejte, že rozptylovací funkce je již implementována a Vám stačí ji jen volat.

5.
A elsewhere
Hash table of size *m* in hashing with chaining contains *n* elements (keys). Its implementation optimizes the *Insert* operation. The worst case of insertion a new element has the complexity

a) $\Theta(n)$
b) $\Theta(m)$
c) $\Theta(m/n)$
d) $O(1)$
e) $\Theta(\log(n))$

6.
A where?
Linked list of synonyms
a) minimizes the overall cluster length in open address hashing method
b) solves the problem of collisions by inserting the key to the first empty space in the array
c) is a sequence of synonyms stored in continuous segment of addresses
d) does not exist in open address hashing

7.
Zřetězený seznam synonym
e) minimalizuje délku clusterů u metody otevřeného rozptylování
f) řeší kolize uložením klíče na první volné místo v poli
g) je posloupnost synonym uložená v souvislém úseku adres
h) u otevřeného rozptylování nevzniká

8.
Metoda hashování s vnějším zřetězením
  a) nemá problém s kolizemi, protože při ní nevznikají
  b) dokáže uložit pouze předem známý počet klíčů
  c) ukládá synonyma do samostatných seznamů v dynamické paměti
  d) ukládá synonyma spolu s ostatními klíči v poli

Každý alespoň elementární popis zřetězeného rozptylování vede na odpověď b). Kolize vznikají vždy, pole se tu nepoužívá a počet klíčů není teoreticky omezen.

9.
Metoda hashování s vnějším zřetězením
  a) nemá problém s kolizemi, protože nevznikají
  b) řeší kolize uložením klíče na první volné místo v poli
  c) dokáže uložit pouze předem známý počet klíčů
  d) dokáže uložit libovolný předem neznámý počet klíčů

10.
Metoda hashování s vnějším zřetězením
- nemá problém s kolizemi, protože při ní nevznikají
- dokáže uložit pouze předem známý počet klíčů
- ukládá synonyma do samostatných seznamů v dynamické paměti
- ukládá synonyma spolu s ostatními klíči v poli

11.
Rozptylovací tabulka o velikosti *m* se zřetězeným rozptylováním obsahuje *n* prvků. Nejhorší případ, který může při vložení dalšího prvku nastat, má složitost

- o $\Theta(n)$
- o $\Theta(m)$
- o $\Theta(m/n)$
- ✓ $O(1)$
- o $\Theta(\log(n))$

12.

Implementujte operace Init, Search, Insert a Delete pro rozptylovací tabulku se zřetězeným rozptylováním, do níž se ukládají celočíselné klíče. Předpokládejte, že rozptylovací funkce je již implementována a Vám stačí ji jen volat.

Zde – pokud se nevyskytne přímá žádost – řešení prozatím neuvádím, jedná se jen o přímou implementaci standardní situace popsané v přednášce i literatuře, nic se tu nemusí „vymýšlet“, předpokládáme tedy, že si zájemci mohou (příp. s knihou či obrazovkou) tamtéž uvedené kódy projít.

-------------------------------------- HASHING OPEN --

13.
Metoda otevřeného rozptylování

a) generuje vzájemně disjunktní řetězce synonym
b) dokáže uložit pouze předem známý počet klíčů
c) zamezuje vytváření dlouhých clusterů ukládáním synonym do samostatných seznamů v dynamické paměti
d) dokáže uložit libovolný předem neznámý počet klíčů

Varianty a), c), d) platí zřejmě pro zřetězené rozptylování, což vyplývá bezprostředně již z jakéhokoli jednoduchého popisu zřetězeného rozptylování. Zbývá jen správná možnost b).

14.
Metoda otevřeného rozptylování

a) dokáže uložit libovolný předem neznámý počet klíčů
b) nemá problém s kolizemi, protože nevznikají
c) ukládá prvky s klíči v dynamické paměti
d) ukládá prvky do pole pevné délky

15.
Metoda otevřeného rozptylování

- generuje vzájemně disjunktní řetězce synonym
- dokáže uložit pouze předem známý počet klíčů
- zamezuje vytváření dlouhých clusterů ukládáním synonym do samostatných seznamů v dynamické paměti
- dokáže uložit libovolný předem neznámý počet klíčů

Varianty a), c), d) platí zřejmě pro zřetězené rozptylování, což vyplývá bezprostředně již z jakéhokoli jednoduchého popisu zřetězeného rozptylování. Zbývá jen správná možnost b).

16.
Rozptylovací tabulka o velikosti *m* s otevřeným rozptylováním obsahuje *n* prvků. Při vložení (n+1)-ého prvku nastala kolize. To znamená, že

a) n = m
b) n > m
c) n = m mod n
d) m = n mod m
e) nic z předchozího

17.
Hash table of size *m* in open address hashing contains *n* elements (keys). While inserting the $(n+1)^{th}$ element a collision appeared. That means:

f) $n = m$

g) $n > m$
h) $n = m \bmod n$
i) $m = n \bmod m$
j) none of these answers

18.
A Where?
The hash table uses the hash function (x) = x mod 6 and it was originally empty. Then the following elements were inserted into the table and one collision occured. Which elements?

```
a)   6   12   24
b)  24    6   12
c)   1    7    6
d)   5    6    7
e)   2    3    4
```

19.
A Where?
The hash table uses the hash function (x) = x mod 5 and it was originally empty. Then the following elements were inserted into the table and one collision occured. Which elements?

a) 5 6 7
b) 10 15 20
c) 20 10 15
d) 5 6 11
e) 3 6 9

20.
A elsewhere
The word “cluster“ used in open hashing means the following

a) a sequence of synonyms stored in a continuous area of addresses
b) a sequence of keys stored in a continuous area of addresses
c) a sequence of synonyms stored in the dynamic memory
d) nothing, clusters does not appear in the open hashing

21.
A where
In open address hashing

a) unlimited number of synonyms can be stored
b) the range of keys must be defined
c) the array must be extended after a given number of collisions
d) number of stored elements is limited by the array size

22.
Kolize při vkládání klíče do rozptylovací tabulky s otevřeným rozptylováním znamená, že:
- o klíč nebude možno do tabulky vložit
- o klíč bude možno do tabulky vložit po jejím zvětšení
- ✓ místo pro klíč v poli je již obsazeno jiným klíčem
- o v paměti není dostatek místa pro zvětšení tabulky
- o kapacita tabulky je vyčerpána

23.
V otevřeném rozptylování
e) je nutno definovat rozsah hodnot klíčů
f) je počet uložených prvků omezen velikostí pole
g) je nutno po určitém počtu kolizí zvětšit velikost pole

h) je možno uložit libovolný počet synonym

V otevřeném rozptylování je maximální počet uložených prvků dán velikostí pole, varianta d) neplatí. Většinou se počítá s tím, že pole má danou velikost (podle charakteru a rozsahu dat), jeho velikost je tedy daná a nemění se. Varianta c) neplatí. Zároveň se potvrzuje platnost varianty b). Varianta a) neplatí, rozptylovací funkce má za úkol zpracovat jakýkoli klíč.

24.
Cluster (u metody otevřeného rozptylování)

a) je posloupnost synonym uložená v souvislém úseku adres
b) je posloupnost klíčů uložená v souvislém úseku adres
c) je posloupnost synonym uložená v dynamické paměti
d) u otevřeného rozptylování nevzniká

Definitorická otázka, viz přednášky/literaturu. Platí varianta b).

25.
Implementujte operace Init, Search, Insert pro rozptylovací tabulku s otevřeným rozptylováním, do níž se ukládají celočíselné klíče. Předpokládejte, že rozptylovací funkce je již implementována a Vám stačí ji jen volat. Použijte strategii „Linear probing".

Zde – pokud se nevyskytne přímá žádost – řešení prozatím neuvádím, jedná se jen o přímou implementaci standardní situace popsané v přednášce i literatuře, nic se tu nemusí „vymýšlet", předpokládáme tedy, že si zájemci mohou (příp. s knihou či obrazovkou) tamtéž uvedené kódy projít.

-------------------------------------- HASHING LINEAR ---

26.
Pole, ve kterém je uložena rozptylovací tabulka vypadá při použití rozptylovací funkce h(k) = k mod 5, lineárního prohledávání (linear probing) a vložení klíčů
8, 9, 4, 3 (vkládaných v pořadí zleva doprava) takto

a)

0	1	2	3	4
4	3		8	9

b)

0	1	2	3	4
8	9	4	3	

c)

0	1	2	3	4
8	9		3	4

d)

0	1	2	3	4
	9	8	3	4

27.
Pole, ve kterém je uložena rozptylovací tabulka vypadá při použití rozptylovací funkce h(k) = k mod 5, lineárního prohledávání (linear probing) a vložení klíčů 7, 1, 6, 2 (vkládaných v pořadí zleva doprava) takto

a)

0	1	2	3	4
7	1	6	2	

b)

0	1	2	3	4
6		7	1	2

c)

0	1	2	3	4
	1	7	6	2

d)

0	1	2	3	4
	6	2	1	7

28.
A hash table is stored in an array. The keys inserted into the originally empty table are 7, 1, 6, 2. The table uses hash function h(k) = k mod 5 and resolves collisïons by linear probing scheme. What is the resulting contents of the table?

a)

0	1	2	3	4
7	1	6	2	

b)

0	1	2	3	4
6		7	1	2

c)

0	1	2	3	4
	1	7	6	2

d)

0	1	2	3	4
	6	2	1	7

29.
Pole, ve kterém je uložena rozptylovací tabulka vypadá při použití rozptylovací funkce h(k) = k mod 5, lineárního prohledávání (linear probing) a vložení klíčů
5, 9, 4, 6 (vkládaných v pořadí zleva doprava) takto

	0	1	2	3	4
a)	5	6	4		9
b)	5	6	9		4
c)	5	4	6		9
d)	4	5	6		9

30. A
Hashing uses linear probing and a hash function *h*(*k*) = *k* mod 5. We insert the keys
5, 9, 4, 6 (in this order). The array used for storage of the hash table looks then as follows:

	0	1	2	3	4
a)	5	6	4		9
b)	5	6	9		4
c)	5	4	6		9
d)	4	5	6		9

31.
Pole, ve kterém je uložena rozptylovací tabulka vypadá při použití rozptylovací funkce h(k) = k mod 5, lineárního prohledávání (linear probing) a vložení klíčů
4, 5, 9, 6 (vkládaných v pořadí zleva doprava) takto

	0	1	2	3	4
a)	5	6	4		9
b)	5	9	6		4
c)	4	6	5		9
d)	4	5	6		9

32. A
Hashing uses linear probing and a hash function *h*(*k*) = *k* mod 5. We insert the keys
4, 5, 9, 6 (in this order). The array used for storage of the hash table looks then as follows:

	0	1	2	3	4
a)	5	6	4		9
b)	5	9	6		4
c)	4	6	5		9
d)	4	5	6		9

33.
Pole, ve kterém je uložena rozptylovací tabulka vypadá při použití rozptylovací funkce h(k) = k mod 5, lineárního prohledávání (linear probing) a vložení klíčů 6, 5, 9, 4 (vkládaných v pořadí zleva doprava) takto

	0	1	2	3	4
a)	5	6	4		9
b)	5	6	9		4
c)	4	6	5		9
d)	4	5	6		9

6 mod 5 = 1, takže hodnota 6 se vloží na pozici č. 1.
5 mod 5 = 0, takže hodnota 5 se vloží na pozici č. 0.
9 mod 5 = 4, takže hodnota 9 se vloží na pozici č. 4.
Tím získáme tabulka naznačenou vpravo.

0	1	2	3	4
5	6			9

Zbývá vložit hodnotu 4.
4 mod 5 = 4. Hodnota 4 by se tedy měla vložit pozici č. 4. Ta je však již obsazená hodnotou 9 a je tedy zapotřebí najít nejbližší volnou pozici směrem doprava. Za pozicí č. 4 bezprostředně následuje pozice č. 0. (tabulka je „zacyklená“) a pozice č. 1, jež jsou obě také obsazeny, Hodnota 4 se tedy vloží na pozici č. 2, jak ukazuje poslední tabulka, ekvivalentní s variantou a).

0	1	2	3	4
5	6	4		9

34.

Pole, ve kterém je uložena rozptylovací tabulka vypadá při použití rozptylovací funkce h(k) = k mod 5, lineárního prohledávání (linear probing) a vložení klíčů 6, 4, 5, 9 (vkládaných v pořadí zleva doprava) takto

a)

0	1	2	3	4
5	6	4		9

b)

0	1	2	3	4
5	6	9		4

c)

0	1	2	3	4
4	6	5		9

d)

0	1	2	3	4
4	5	6		9

6 mod 5 = 1, takže hodnota 6 se vloží na pozici č. 1.
4 mod 5 = 4, takže hodnota 4 se vloží na pozici č. 4.
5 mod 5 = 0, takže hodnota 5 se vloží na pozici č. 0.
Tím získáme tabulka naznačenou vpravo.
Zbývá vložit hodnotu 9.

0	1	2	3	4
5	6			4

9 mod 5 = 4. Hodnota 9 by se tedy měla vložit pozici č. 4. Ta je však již obsazená hodnotou 4 a je tedy zapotřebí najít nejbližší volnou pozici směrem doprava. Za pozicí č. 4 bezprostředně následuje pozice č. 0. (tabulka je „zacyklená“) a pozice č. 1, jež jsou obě také obsazeny, Hodnota 9 se tedy vloží na pozici č. 2, jak ukazuje poslední tabulka, ekvivalentní s variantou b).

0	1	2	3	4
5	6	9		4

35.
Implementujte operace Init, Search, Insert pro rozptylovací tabulku s otevřeným rozptylováním, do níž se ukládají celočíselné klíče. Předpokládejte, že rozptylovací funkce je již implementována a Vám stačí ji jen volat. Použijte strategii „Linear probing“.

Zde – pokud se nevyskytne přímá žádost – řešení prozatím neuvádím, jedná se jen o přímou implementaci standardní situace popsané v přednášce i literatuře, nic se tu nemusí „vymýšlet“, předpokládáme tedy, že si zájemci mohou (příp. s knihou či obrazovkou) tamtéž uvedené kódy projít.

--------------------------------------- HASHING DOUBLE --

36.
Double hashing
a) je metoda ukládání klíčů na dvě různá místa současně
b) je metoda minimalizace kolizí u metody otevřeného rozptylování
c) má vyšší pravděpodobnost vzniku kolizí než linear probing
d) je metoda minimalizace kolizí u metody rozptylování s vnějším zřetězením

Tady nepomůže asi nic jiného než dobrá paměť.

37.
Double hashing
a) má stejnou pravděpodobnost vzniku dlouhých clusterů jako linear probing
b) je metoda ukládání klíčů na dvě různá místa
c) je metoda minimalizace délky clusterů u metody otevřeného rozptylování
d) má vyšší pravděpodobnost vzniku dlouhých clusterů než linear probing

--------------------------------------- HASHING COALESCED ---

38.
Uložte dané klíče v daném pořadí postupně do rozptylovací tabulky. Porovnejte počet kolizí při ukládání klíčů do tabulek různé velikosti a použití různých strategií pro srůstání řetězců kolidujících klíčů: LISCH, LICH, EISCH, EICH.

Postupná demonstrace, poslední řádek s tečkami představuje pole referencí, pomocí nějž se udržuje struktura jednotlivých (srůstajících) seznamů synonym.:

```
LISCH - Late Insert Standard Coalesced Hashing
Keys to insert:  9 11 18 27 29 36 43 45
Table size: 9
Hash function: h(k) = k % 9

  0  1  2  3  4  5  6  7  8   Insert(9)
  9  -  -  -  -  -  -  -  -   Index: [0]
  .  .  .  .  .  .  .  .  .   Collisions: 0

  0  1  2  3  4  5  6  7  8   Insert(11)
  9  - 11  -  -  -  -  -  -   Index: [2]
  .  .  .  .  .  .  .  .  .   Collisions: 0

  0  1  2  3  4  5  6  7  8   Insert(18)
  9  - 11  -  -  -  -  - 18   Index: [0]->[8]
  8  .  .  .  .  .  .  .  .   Collisions: 1

  0  1  2  3  4  5  6  7  8   Insert(27)
  9  - 11  -  -  -  - 27 18   Index: [0]->[8]->[7]
  8  .  .  .  .  .  .  .  7   Collisions: 2

  0  1  2  3  4  5  6  7  8   Insert(29)
  9  - 11  -  -  - 29 27 18   Index: [2]->[6]
  8  .  6  .  .  .  .  .  7   Collisions: 1

  0  1  2  3  4  5  6  7  8   Insert(36)
  9  - 11  -  - 36 29 27 18   Index: [0]->[8]->[7]->[5]
  8  .  6  .  .  .  .  5  7   Collisions: 3

  0  1  2  3  4  5  6  7  8   Insert(43)
  9  - 11  - 43 36 29 27 18   Index: [7]->[5]->[4]
  8  .  6  .  .  4  .  5  7   Collisions: 2

  0  1  2  3  4  5  6  7  8   Insert(45)
  9  - 11 45 43 36 29 27 18   Index: [0]->[8]->[7]->[5]->[4]->[3]
  8  .  6  .  3  4  .  5  7   Collisions: 5
--------------------------------  Total collisions 14


LISCH - Late Insert Standard Coalesced Hashing
Keys to insert:  10 12 20 23 32 39 40
Table size: 10
Hash function: h(k) = k % 10

  0  1  2  3  4  5  6  7  8  9   Insert(10)
 10  -  -  -  -  -  -  -  -  -   Index: [0]
  .  .  .  .  .  .  .  .  .  .   Collisions: 0

  0  1  2  3  4  5  6  7  8  9   Insert(12)
 10  - 12  -  -  -  -  -  -  -   Index: [2]
  .  .  .  .  .  .  .  .  .  .   Collisions: 0

  0  1  2  3  4  5  6  7  8  9   Insert(20)
 10  - 12  -  -  -  -  -  - 20   Index: [0]->[9]
  9  .  .  .  .  .  .  .  .  .   Collisions: 1

  0  1  2  3  4  5  6  7  8  9   Insert(23)
```

```
 10  - 12 23  -  -  -  -  - 20   Index: [3]
  9  .  .  .  .  .  .  .  .  .   Collisions: 0

  0  1  2  3  4  5  6  7  8  9   Insert(32)
 10  - 12 23  -  -  -  - 32 20   Index: [2]->[8]
  9  .  8  .  .  .  .  .  .  .   Collisions: 1

  0  1  2  3  4  5  6  7  8  9   Insert(39)
 10  - 12 23  -  -  - 39 32 20   Index: [9]->[7]
  9  .  8  .  .  .  .  .  .  7   Collisions: 1

  0  1  2  3  4  5  6  7  8  9   Insert(40)
 10  - 12 23  -  - 40 39 32 20   Index: [0]->[9]->[7]->[6]
  9  .  8  .  .  .  .  6  .  7   Collisions: 3
--------------------------------  Total collisions 6
```

39.
Oba předchozí případy zopakujeme pro stejná data, pouze použíjeme tabulku se „sklepem“ o velikosti 2, přičemž celková velikost tabulky se nezmění.

```
LICH - Late Insert Coalesced Hashing
Keys to insert:  9 11 18 27 29 36 43 45
Table size: 7    Cellar size: 2
Hash function: h(k) = k % 7

  0  1  2  3  4  5  6 |  7  8   Insert(9)
  -  -  9  -  -  -  - |  -  -   Index: [2]
  .  .  .  .  .  .  . |  .  .   Collisions: 0

  0  1  2  3  4  5  6 |  7  8   Insert(11)
  -  -  9  - 11  -  - |  -  -   Index: [4]
  .  .  .  .  .  .  . |  .  .   Collisions: 0

  0  1  2  3  4  5  6 |  7  8   Insert(18)
  -  -  9  - 11  -  - |  - 18   Index: [4]->[8]
  .  .  .  .  8  .  . |  .  .   Collisions: 1

  0  1  2  3  4  5  6 |  7  8   Insert(27)
  -  -  9  - 11  - 27 |  - 18   Index: [6]
  .  .  .  .  8  .  . |  .  .   Collisions: 0

  0  1  2  3  4  5  6 |  7  8   Insert(29)
  - 29  9  - 11  - 27 |  - 18   Index: [1]
  .  .  .  .  8  .  . |  .  .   Collisions: 0

  0  1  2  3  4  5  6 |  7  8   Insert(36)
  - 29  9  - 11  - 27 | 36 18   Index: [1]->[7]
  .  7  .  .  8  .  . |  .  .   Collisions: 1

  0  1  2  3  4  5  6 |  7  8   Insert(43)
  - 29  9  - 11 43 27 | 36 18   Index: [1]->[7]->[5]
  .  7  .  .  8  .  . |  5  .   Collisions: 2

  0  1  2  3  4  5  6 |  7  8   Insert(45)
  - 29  9 45 11 43 27 | 36 18   Index: [3]
  .  7  .  .  8  .  . |  5  .   Collisions: 0
--------------------------------  Total collisions 4


LICH - Late Insert Coalesced Hashing
```

```
Keys to insert:  10 12 20 23 32 39 40
Table size: 8    Cellar size: 2
Hash function: h(k) = k % 8

  0  1  2  3  4  5  6  7 |  8  9   Insert(10)
  -  - 10  -  -  -  -  - |  -  -   Index: [2]
  .  .  .  .  .  .  .  . |  .  .   Collisions: 0

  0  1  2  3  4  5  6  7 |  8  9   Insert(12)
  -  - 10  - 12  -  -  - |  -  -   Index: [4]
  .  .  .  .  .  .  .  . |  .  .   Collisions: 0

  0  1  2  3  4  5  6  7 |  8  9   Insert(20)
  -  - 10  - 12  -  -  - |  - 20   Index: [4]->[9]
  .  .  .  .  9  .  .  . |  .  .   Collisions: 1

  0  1  2  3  4  5  6  7 |  8  9   Insert(23)
  -  - 10  - 12  -  - 23 |  - 20   Index: [7]
  .  .  .  .  9  .  .  . |  .  .   Collisions: 0

  0  1  2  3  4  5  6  7 |  8  9   Insert(32)
 32  - 10  - 12  -  - 23 |  - 20   Index: [0]
  .  .  .  .  9  .  .  . |  .  .   Collisions: 0

  0  1  2  3  4  5  6  7 |  8  9   Insert(39)
 32  - 10  - 12  -  - 23 | 39 20   Index: [7]->[8]
  .  .  .  .  9  .  .  8 |  .  .   Collisions: 1

  0  1  2  3  4  5  6  7 |  8  9   Insert(40)
 32  - 10  - 12  - 40 23 | 39 20   Index: [0]->[6]
  6  .  .  .  9  .  .  8 |  .  .   Collisions: 1
--------------------------------  Total collisions 3
```

V souladu s teoriíí, „sklep“ pomáhá snížit počet kolizí.

40.
Oba předchozí případy zopakujeme pro stejná data, použijme metodu EISCH , přičemž celková velikost tabulky se nezmění.

```
EISCH - Early Insert Standard Coalesced Hashing
Keys to insert:  9 11 18 27 29 36 43 45
Table size: 9
Hash function: h(k) = k % 9

  0  1  2  3  4  5  6  7  8   Insert(9)
  9  -  -  -  -  -  -  -  -   Index: [0]
  .  .  .  .  .  .  .  .  .   Collisions: 0

  0  1  2  3  4  5  6  7  8   Insert(11)
  9  - 11  -  -  -  -  -  -   Index: [2]
  .  .  .  .  .  .  .  .  .   Collisions: 0

  0  1  2  3  4  5  6  7  8   Insert(18)
  9  - 11  -  -  -  -  - 18   Index: [0]->[8]
  8  .  .  .  .  .  .  .  .   Collisions: 1

  0  1  2  3  4  5  6  7  8   Insert(27)
  9  - 11  -  -  -  - 27 18   Index: [0]->[7]
  7  .  .  .  .  .  .  8  .   Collisions: 1
```

```
  0  1  2  3  4  5  6  7  8   Insert(29)
  9  - 11  -  -  - 29 27 18   Index: [2]->[6]
  7  .  6  .  .  .  .  8  .   Collisions: 1

  0  1  2  3  4  5  6  7  8   Insert(36)
  9  - 11  -  - 36 29 27 18   Index: [0]->[5]
  5  .  6  .  .  7  .  8  .   Collisions: 1

  0  1  2  3  4  5  6  7  8   Insert(43)
  9  - 11  - 43 36 29 27 18   Index: [7]->[4]
  5  .  6  .  8  7  .  4  .   Collisions: 1

  0  1  2  3  4  5  6  7  8   Insert(45)
  9  - 11 45 43 36 29 27 18   Index: [0]->[3]
  3  .  6  5  8  7  .  4  .   Collisions: 1
-------------------------------  Total collisions 6

EISCH - Early Insert Standard Coalesced Hashing
Keys to insert:  10 12 20 23 32 39 40
Table size: 10
Hash function: h(k) = k % 10

  0  1  2  3  4  5  6  7  8  9   Insert(10)
 10  -  -  -  -  -  -  -  -  -   Index: [0]
  .  .  .  .  .  .  .  .  .  .   Collisions: 0

  0  1  2  3  4  5  6  7  8  9   Insert(12)
 10  - 12  -  -  -  -  -  -  -   Index: [2]
  .  .  .  .  .  .  .  .  .  .   Collisions: 0

  0  1  2  3  4  5  6  7  8  9   Insert(20)
 10  - 12  -  -  -  -  -  - 20   Index: [0]->[9]
  9  .  .  .  .  .  .  .  .  .   Collisions: 1

  0  1  2  3  4  5  6  7  8  9   Insert(23)
 10  - 12 23  -  -  -  -  - 20   Index: [3]
  9  .  .  .  .  .  .  .  .  .   Collisions: 0

  0  1  2  3  4  5  6  7  8  9   Insert(32)
 10  - 12 23  -  -  -  - 32 20   Index: [2]->[8]
  9  .  8  .  .  .  .  .  .  .   Collisions: 1

  0  1  2  3  4  5  6  7  8  9   Insert(39)
 10  - 12 23  -  -  - 39 32 20   Index: [9]->[7]
  9  .  8  .  .  .  .  .  .  7   Collisions: 1

  0  1  2  3  4  5  6  7  8  9   Insert(40)
 10  - 12 23  -  - 40 39 32 20   Index: [0]->[6]
  6  .  8  .  .  .  9  .  .  7   Collisions: 1
-------------------------------  Total collisions 4
```

Oba předchozí případy nakonec zopakujeme pro stejná data, použijme metodu EICH a tabulku se „sklepem“ o velikosti 2, přičemž celková velikost tabulky se nezmění.

```
EICH - Early Insert Coalesced Hashing
Keys to insert:  9 11 18 27 29 36 43 45
Table size: 7    Cellar size: 2
Hash function: h(k) = k % 7

  0  1  2  3  4  5  6 |  7  8   Insert(9)
```

```
  -  -  9  -  -  -  - |  -  -   Index: [2]
  .  .  .  .  .  .  . |  .  .   Collisions: 0

  0  1  2  3  4  5  6 |  7  8   Insert(11)
  -  -  9  - 11  -  - |  -  -   Index: [4]
  .  .  .  .  .  .  . |  .  .   Collisions: 0

  0  1  2  3  4  5  6 |  7  8   Insert(18)
  -  -  9  - 11  -  - |  - 18   Index: [4]->[8]
  .  .  .  .  8  .  . |  .  .   Collisions: 1

  0  1  2  3  4  5  6 |  7  8   Insert(27)
  -  -  9  - 11  - 27 |  - 18   Index: [6]
  .  .  .  .  8  .  . |  .  .   Collisions: 0

  0  1  2  3  4  5  6 |  7  8   Insert(29)
  - 29  9  - 11  - 27 |  - 18   Index: [1]
  .  .  .  .  8  .  . |  .  .   Collisions: 0

  0  1  2  3  4  5  6 |  7  8   Insert(36)
  - 29  9  - 11  - 27 | 36 18   Index: [1]->[7]
  .  7  .  .  8  .  . |  .  .   Collisions: 1

  0  1  2  3  4  5  6 |  7  8   Insert(43)
  - 29  9  - 11 43 27 | 36 18   Index: [1]->[5]
  .  5  .  .  8  7  . |  .  .   Collisions: 1

  0  1  2  3  4  5  6 |  7  8   Insert(45)
  - 29  9 45 11 43 27 | 36 18   Index: [3]
  .  5  .  .  8  7  . |  .  .   Collisions: 0
-------------------------------  Total collisions 3

EICH - Early Insert Coalesced Hashing
Keys to insert:  10 12 20 23 32 39 40
Table size: 8    Cellar size: 2
Hash function: h(k) = k % 8

  0  1  2  3  4  5  6  7 |  8  9   Insert(10)
  -  - 10  -  -  -  -  - |  -  -   Index: [2]
  .  .  .  .  .  .  .  . |  .  .   Collisions: 0

  0  1  2  3  4  5  6  7 |  8  9   Insert(12)
  -  - 10  - 12  -  -  - |  -  -   Index: [4]
  .  .  .  .  .  .  .  . |  .  .   Collisions: 0

  0  1  2  3  4  5  6  7 |  8  9   Insert(20)
  -  - 10  - 12  -  -  - |  - 20   Index: [4]->[9]
  .  .  .  .  9  .  .  . |  .  .   Collisions: 1

  0  1  2  3  4  5  6  7 |  8  9   Insert(23)
  -  - 10  - 12  -  - 23 |  - 20   Index: [7]
  .  .  .  .  9  .  .  . |  .  .   Collisions: 0

  0  1  2  3  4  5  6  7 |  8  9   Insert(32)
 32  - 10  - 12  -  - 23 |  - 20   Index: [0]
  .  .  .  .  9  .  .  . |  .  .   Collisions: 0

  0  1  2  3  4  5  6  7 |  8  9   Insert(39)
 32  - 10  - 12  -  - 23 | 39 20   Index: [7]->[8]
  .  .  .  .  9  .  .  8 |  .  .   Collisions: 1
```

```
  0  1  2  3  4  5  6  7 |  8  9   Insert(40)
 32  - 10  - 12  - 40 23 | 39 20   Index: [0]->[6]
  6  .  .  .  9  .  .  8 |  .  .   Collisions: 1
-------------------------------  Total collisions 3
```

41.

Pro data

```
 9 11 18 27 29 36 43 45
```

jsme použitím metod LISCH, LICH, EISCH, EICH získali čtyři různé tabulky stejné velikosti, které pro přehled opakujeme níže. Předpokládejme, že v tabulce budeme vzhledávat vždy pouze klíče, které tam jsou uloženy, přičemž frekvence hledání budou pro všechny klíče stejné (= všechny klíče budeme vyhledávat stejně často). Která z uvedených tabulek je z tohoto hlediska nejvýhodnější?

```
LISCH - Late Insert Standard Coalesced Hashing
  0  1  2  3  4  5  6  7  8
  9  - 11 45 43 36 29 27 18
  8  .  6  .  3  4  .  5  7

LICH - Late Insert Coalesced Hashing
  0  1  2  3  4  5  6 |  7  8
  - 29  9 45 11 43 27 | 36 18
  .  7  .  .  8  .  . |  5  .

EISCH - Early Insert Standard Coalesced Hashing
  0  1  2  3  4  5  6  7  8
  9  - 11 45 43 36 29 27 18
  3  .  6  5  8  7  .  4  .

EICH - Early Insert Coalesced Hashing
  0  1  2  3  4  5  6 |  7  8
  - 29  9 45 11 43 27 | 36 18
  .  5  .  .  8  7  . |  .  .
```

Pro každou tabulku musíme sečíst počet porovnání klíčů při hledání každého jednotlivého klíče.
Search cost: [key no_of_checks]:
LISCH
[9 1] [11 1] [18 2] [27 3] [29 2] [36 4] [43 3] [45 6]   Total checks = 22
LICH
[9 1] [11 1] [18 2] [27 1] [29 1] [36 2] [43 3] [45 1]   Total checks = 12
EISCH
[9 1] [11 1] [18 6] [27 4] [29 2] [36 3] [43 2] [45 2]   Total checks = 21
EICH
[9 1] [11 1] [18 2] [27 1] [29 1] [36 3] [43 2] [45 1]   Total checks = 12
Nejvýhodnější (a to zřetelně nejvýhodnější) jsou tabulky využívající „sklep“, tj LICH a EICH.

42.

Předchozí úlohu zopakujeme pro data

```
10 12 20 23 32 39 40
```

a jim příslušné čtyři tabulky o velikosti 10 a případné velikosti “sklepa“ 2.
Získáme:
LISCH
[10 1] [12 1] [20 2] [23 1] [32 2] [39 2] [40 4]   Total checks = 13
LICH
[10 1] [12 1] [20 2] [23 1] [32 1] [39 2] [40 2]   Total checks = 10
EISCH
[10 1] [12 1] [20 3] [23 1] [32 2] [39 2] [40 2]   Total checks = 12
EICH

[10 1] [12 1] [20 2] [23 1] [32 1] [39 2] [40 2]   Total checks = 10

Opět jsou mírně výhodnější tabulky LICH a EICH.