ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ

ΚΑΤΑΚΕΡΜΑΤΙΣΜΟΣ

6.2 Χειρισμός Συγκρούσεων

Αλυσιδωτή σύνδεση (chaining)

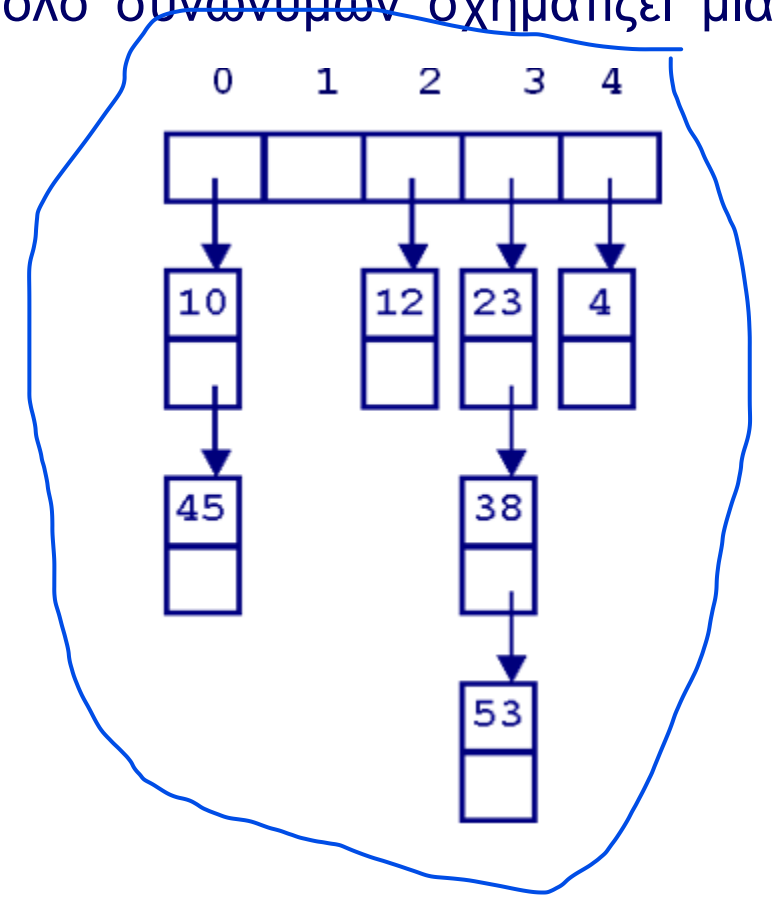
Η μέθοδος αυτή βασίζεται στις συναρτήσεις κατακερματισμού και στις συνδεδεμένες λίστες.

Κάθε μια από τις τιμές κατακερματισμού αντιστοιχεί σε μια συνδεδεμένη υπολίστα συνωνύμων, δηλαδή κάθε σύνολο συνωνύμων σχηματίζει μια συνδεδεμένη υπολίστα.

Παράδειγμα: αν έχουμε τιμές κατακερματισμού από 0 έως 4 και κλειδιά με τιμές

23, 38, 10, 12, 53, 45 και 4

τότε οι υπολίστες συνωνύμων που σχηματίζονται είναι όπως δείχνει το διπλανό σχήμα:



Αλυσιδωτή σύνδεση (chaining)

Όταν η υπολίστα έχει περιορισμένο αριθμό εγγραφών, ονομάζεται πολλές φορές **κάδος** (**bucket**).

Σε ορισμένες περιπτώσεις το μέγεθος του κάδου είναι μία μόνο εγγραφή, αλλά συνήθως ένας κάδος έχει χωρητικότητα πολλών εγγραφών.

Ανεξάρτητα από το αν το μέγεθος της υπολίστας είναι σταθερό ή όχι, το μεγαλύτερο **μειονέκτημα** του κατακερματισμού είναι η αδυναμία δημιουργίας σταθερού αριθμού εγγραφών σε κάθε υπολίστα.

Αλυσιδωτή σύνδεση (chaining): *αποδοτικότητα*

Η αποδοτικότητα αυτής της μεθόδου εξαρτάται από πολλές παραμέτρους:

- Το πρώτο πράγμα που θα πρέπει να σκεφτούμε είναι **το πλήθος των υπολιστών** που θα χρησιμοποιήσουμε.
- Για να καθορίσουμε, όμως, τον αριθμό των υπολιστών, πρέπει πρώτα να αποφασίσουμε **πόσες εγγραφές θα περιέχει κάθε υπολίστα**.

Αν επιθυμούμε να έχουμε **k εγγραφές το πολύ** σε κάθε υπολίστα και έχουμε συνολικά **N εγγραφές στη λίστα**, τότε ο **συνολικός αριθμός υπολιστών ή κάδων** που θα έχουμε είναι

$$b = \lceil N/k \rceil$$

και η σχέση υπολογισμού της τιμής κατακερματισμού είναι:

HashValue = KeyValue % b

όπου HashValue είναι η τιμή κατακερματισμού και KeyValue η τιμή του κλειδιού που κατακερματίζεται.

Αλυσιδωτή σύνδεση (chaining): *αποδοτικότητα*

- **Αν οι τιμές κλειδιών σχηματίζουν ένα πυκνό σύνολο** από ακέραιες τιμές (δηλαδή, αν υπάρχει μια εγγραφή με μια τιμή κλειδιού για κάθε πιθανή τιμή κλειδιού του διαστήματος των κλειδιών) και αν δεν υπάρχουν διπλότυπα, τότε οποιαδήποτε τιμή κι αν επιλέξουμε για το b θα μας δώσει μια όσο γίνεται πιο τέλεια ομοιόμορφη κατανομή.

- **Αν το N είναι πολλαπλάσιο του b**, η κατανομή θα είναι ακριβώς τέλεια.

- **Αν**, όμως, **οι τιμές κλειδιών δεν σχηματίζουν ένα πυκνό σύνολο**, αλλά ούτε και μπορούν εύκολα να μετατραπούν σε ένα πυκνό σύνολο από ακέραιες τιμές, τότε η επιλογή του b απαιτεί προσοχή.

Αλυσιδωτή σύνδεση (chaining): *αποδοτικότητα*

- Είναι προφανές ότι δεν θα πρέπει να επιλεγεί ένας διαιρέτης ο οποίος να περιέχει έναν παράγοντα που να διαιρείται με τις περισσότερες τιμές κλειδιών.
- Αν, για παράδειγμα, όλες οι τιμές των κλειδιών είναι άρτιοι αριθμοί και επιλέξουμε άρτιο διαιρέτη, τότε δεν θα προκύψει ποτέ περιττός αριθμός ως υπόλοιπο.
- Αυτό θα έχει ως συνέπεια όλες οι τιμές κατακερματισμού να είναι άρτιοι αριθμοί και να χρησιμοποιούνται μόνο οι μισές υπολίστες.
- Στην πραγματικότητα, σε καμία περίπτωση **δεν είναι καλό να επιλέξουμε άρτιο αριθμό για το b**, όταν οι τιμές των κλειδιών δεν είναι πυκνό σύνολο.
- Αυτό ισχύει, γιατί διαιρώντας με άρτιο αριθμό προκύπτει υπόλοιπο περιττός ή άρτιος αριθμός ανάλογα με το αν ο διαιρετέος είναι περιττός ή άρτιος αριθμός.
- Έτσι, αν οι περισσότερες τιμές κλειδιών είναι περιττοί αριθμοί, τότε και τα περισσότερα υπόλοιπα θα είναι περιττοί αριθμοί, ενώ, αν οι περισσότερες τιμές κλειδιών είναι άρτιοι αριθμοί, τότε τα περισσότερα υπόλοιπα θα είναι άρτιοι αριθμοί.

Αλυσιδωτή σύνδεση (chaining): *αποδοτικότητα*

Εμπειρικά έχει αποδειχθεί ότι **το b πρέπει να είναι ο μικρότερος πρώτος αριθμός που είναι ίσος ή μεγαλύτερος από b=N/k**.

Επίσης, καλή επιλογή για το b είναι να πάρουμε **τον πρώτο περιττό αριθμό που είναι ίσος ή μεγαλύτερος από b=N/k**.

Αλυσιδωτή σύνδεση (chaining): *υλοποίηση*

Η υλοποίηση της παραπάνω μεθόδου γίνεται με τη βοήθεια ενός πίνακα, που έχει τόσες θέσεις όσες είναι και οι διαφορετικές τιμές κατακερματισμού.

Ο πίνακας αυτός, που μπορεί να ονομαστεί HashTable, περιέχει δείκτες προς τις συνδεδεμένες υπολίστες συνωνύμων, δηλαδή κάθε καταχώρησή του είναι ένας δείκτης που δείχνει στην αρχή της αντίστοιχης υπολίστας.

Έτσι, λοιπόν, έχουμε μια δομή που αποτελείται από:

- μια **λίστα εγγραφών** και
- έναν **πίνακα δεικτών προς υπολίστες συνωνύμων**, τα οποία κατευθύνονται στη συγκεκριμένη θέση με βάση τη συνάρτηση κατακερματισμού.

Αλυσιδωτή σύνδεση (chaining): *δηλώσεις*

```
#define HMax ...;                /*όριο μεγέθους του πίνακα κατακερματισμού*/
#define VMax ...;                /*όριο μεγέθους της λίστας εγγραφών*/
#define EndOfList -1;            /*σημαία που σηματοδοτεί το τέλος της λίστας και της
                                   κάθε υπολίστας συνωνύμων*/

typedef int ListElementType;     /*τύπος δεδομένων για τα στοιχεία της λίστας*/

typedef int KeyType;             /*τύπος δεδομένων για το κλειδί */
typedef struct {
        KeyType RecKey;
        ListElementType Data;
        int Link;
} ListElm;

typedef struct  {
        int HashTable[HMax];     // πίνακας δεικτών προς τις υπο-λίστες συνωνύμων
        int Size;                // πλήθος εγγραφών
        int SubListPtr;          // δείκτης προς 1η υπολίστα συνωνύμων
        int StackPtr;            // δείκτης στη 1η (ελεύθερη) θέση της λίστας εγγραφών List
        ListElm List[VMax];
} HashListType;
```

Αλυσιδωτή σύνδεση (chaining): *δηλώσεις*

Στο στάδιο της δημιουργίας της δομής αυτής, **κάθε συνώνυμη εγγραφή**, που ανήκει σε ένα συγκεκριμένο σύνολο συνωνύμων, **προστίθεται στην αντίστοιχη συνδεδεμένη υπολίστα**.

Μετά τη δημιουργία της δομής, μια ***αναζήτηση*** για μια συγκεκριμένη εγγραφή περιλαμβάνει:

- πρώτα *κατακερματισμό για τον εντοπισμό της αρχής της κατάλληλης συνδεδεμένης υπολίστας* και,
- εν συνεχεία, *γραμμική αναζήτηση της συνδεδεμένης υπολίστας* αυτής για εντοπισμό της ζητούμενης εγγραφής, αν υπάρχει.

Διαδικασία δημιουργίας κενής δομής

```
void CreateHashList(HashListType *HList)
/*Λειτουργία:     Δημιουργεί μια δομή HList.
  Επιστρέφει:    Την δομή HList.*/
{
        int index;
        HList ->Size = 0;
        HList ->StackPtr = 0;
        index = 0;
       while (index<Vmax-1) {
                        HList ->List[index].Link = index+1;
                        index++;
         }
        HList -> List[index].Link = EndOfList;
        index = 0;
        while (index < HMax) {
                        HList->HashTable[index] = EndOfList;
                        index++;
        }
}
```

Παράδειγμα

Έστω ότι το πλήθος των συνδεδεμένων υπολιστών είναι 5 και ότι η συνάρτηση κατακερματισμού είναι αυτή που χρησιμοποιείται στην διαίρεση, δηλαδή: **HValue= Key % HMax**

HList

HashTable					Size	SubListPtr	StackPtr	List								
								0			.....			VMAX-1		
0	1	2	3	4				Reckey	Data	Link	Reckey	Data	Link	Reckey	Data	Link
-1	-1	-1	-1	-1	0	?	0	?	?	1	?	?	VMAX-2	?	?	-1

```
void CreateHashList(HashListType *HList)
/*Λειτουργία: Δημιουργεί μια δομή HList.
  Επιστρέφει: Την δομή HList.*/
{
        int index;
        HList ->Size = 0;
        HList ->StackPtr = 0;
        index = 0;
       while (index<Vmax-1) {
                HList ->List[index].Link = index+1;
                index++;
         }
        HList -> List[index].Link = EndOfList;
        index = 0;
        while (index < HMax) {
                HList->HashTable[index] = EndOfList;
                index++;
        }
}
```

HashTable	
0	-1
1	-1
2	-1
3	-1
4	-1

	List		
	key	Data	Link
0			1
1			2
2			3
3			4
4			5
5			6
6			7
7			8
8			9
9			10
10			11
11			12
12			13
13			14
14			-1

Διαδικασία ελέγχου γεμάτης λίστας

Για να εισάγουμε μια εγγραφή στη δομή HashList, πρέπει πρώτα να ελέγξουμε αν η συνδεδεμένη λίστα List είναι γεμάτη με μια συνάρτηση FullHashList, που επιστρέφει TRUE, αν η λίστα είναι γεμάτη, και FALSE, διαφορετικά:

```
boolean  FullHashList(HashListType HList)
/*Δέχεται:        Μια δομή HList.
  Λειτουργία:     Ελέγχει αν η λίστα List της δομής HList είναι γεμάτη.
  Επιστρέφει:     TRUE αν η λίστα List είναι γεμάτη, FALSE διαφορετικά.*/
{
  return (HList.Size == VMax);
}

int HashKey(KeyType Key)
{
        /*Σε περίπτωση που το KeyType δεν είναι ακέραιος
        θα πρέπει να μετατρέπεται κατάλληλα το κλειδί σε αριθμό*/
        return Key%HMax;
}
```

Διαδικασία αναζήτησης τιμής στη δομή *HList*

```
void SearchHashList(HashListType HList, KeyType KeyArg, int *Loc, int
*Pred)
/*Δέχεται:      Μια δομή HList και μια τιμή κλειδιού KeyArg.
  Λειτουργία:   Αναζητά μια εγγραφή με κλειδί KeyArg στη δομή HList.
  Επιστρέφει:   Τη θέση Loc της εγγραφής και τη θέση Pred της προηγούμενης
                εγγραφής της υπολίστας στην οποία ανήκει. Αν δεν υπάρχει
                εγγραφή με κλειδί KeyArg τότε Loc=Pred=-1.*/
{
    int HVal;

    Hval = HashKey(KeyArg);
    if (HList.HashTable[HVal] == EndOfList) {  //δεν υπάρχει η εγγραφή
          *Pred = -1;
          *Loc = -1;
    }
    else {          // ενδεχόμενα να υπάρχει η εγγραφή στα συνώνυμα της
                HList.SubListPtr = HList.HashTable[HVal];
                SearchSynonymList(HList, KeyArg, &(*Loc), &(*Pred));
    }
}
```

Διαδικασία αναζήτησης σε υπολίστα συνωνύμων

```
void SearchSynonymList(HashListType HList, KeyType KeyArg, int *Loc, int *Pred)
/*Δέχεται: Μια δομή HList και μια τιμή κλειδιού KeyArg.
  Λειτουργία:      Αναζητά μια εγγραφή με κλειδί KeyArg στην υπολίστα συνωνύμων.
  Επιστρέφει:      Τη θέση Loc της εγγραφής και τη θέση Pred της προηγούμενης
                   εγγραφής στην υπολίστα.*/
{
   int Next;       //η θέση της επόμενης εγγραφής στην υπολίστα

    Next = HList.SubListPtr;
   *Loc = -1;
   *Pred = -1;
   while (Next != EndOfList) {     //διάσχιση της υπολίστας συνωνύμων
         if (HList.List[Next].RecKey == KeyArg) {  //το κλειδί βρέθηκε
               *Loc = Next;          // επιστρέφω μέσω της *Loc τη θέση που βρέθηκε το κλειδί
               Next = EndOfList;     // τερματίζω την αναζήτηση στην υπολίστα συνωνύμων
         }
         else                  //συνεχίζω την αναζήτηση με το επόμενο της υπολίστας συνωνύμων
            {
               *Pred = Next;
               Next = HList.List[Next].Link;
            }
      }
}
```

Διαδικασία εισαγωγής

```
void AddRec(HashListType *HList, ListElm InRec)
/*Δέχεται:      Μια δομή HList και μια εγγραφή InRec.
  Λειτουργία:   Εισάγει την εγγραφή InRec στη λίστα List, αν δεν είναι γεμάτη, και
                ενημερώνει τη δομή HList.
  Επιστρέφει:   Την τροποποιημένη δομή HList.
  Έξοδος:       Μήνυμα γεμάτης λίστας, αν η List είναι γεμάτη, διαφορετικά, αν
                υπάρχει ήδη εγγραφή με το ίδιο κλειδί, εμφάνιση αντίστοιχου
                μηνύματος.*/
{
   int Loc, Pred, New, HVal;

   if (!FullHashList(*HList)) {
         Loc = -1;
         Pred = -1;
         SearchHashList(*HList, InRec.RecKey, &Loc, &Pred);
```

Διαδικασία εισαγωγής

```
        if (Loc == -1) {                //το κλειδί δεν υπάρχει μπορεί να καταχωρηθεί
                HList->Size ++;
                New = HList->StackPtr;
                HList->StackPtr = HList->List[New].Link;
                HList->List[New] = InRec; // το κλειδί καταχωρείται στη θέση New
                if (Pred == -1)  { //Δεν υπάρχει υπολίστα συνωνύμων
                        HVal = HashKey(InRec.RecKey);
                        HList->HashTable[HVal] =New;
                        HList->List[New].Link = EndOfList;
                }
                else {                  //Υπάρχει υπολίστα συνωνύμων
                        HList->List[New].Link = HList->List[Pred].Link;
                        HList->List[Pred].Link = New;
                 }
         }
        else
             printf("YPARXEI HDH EGGRAFH ME TO IDIO KLEIDI \n");
    }
else
    printf("Full list...\n");
}
```

Παράδειγμα

Έστω ότι το πλήθος των συνδεδεμένων υπολιστών είναι 5 και ότι η συνάρτηση κατακερματισμού είναι αυτή που χρησιμοποιείται στην διαίρεση, δηλαδή:

HValue= Key % HMax

HMax=5

Ο πίνακας HashTable θα έχει 5 θέσεις σε καθεμιά από τις οποίες καταχωρούμε αρχικά την τιμή -1, που λειτουργεί ως μηδενικός δείκτης.

HashTable

0	-1
1	-1
2	-1
3	-1
4	-1

VMax=15

Το μέγεθος του πίνακα List θεωρούμε ότι είναι 15, δηλαδή

και ο πίνακας List θα είναι ως εξής:

List

	key	Data	Link
0			1
1			2
2			3
3			4
4			5
5			6
6			7
7			8
8			9
9			10
10			11
11			12
12			13
13			14
14			-1

Παράδειγμα

- Δηλαδή, η εγγραφή που θα αποθηκευτεί στην θέση 0 του πίνακα List θα έχει ως επόμενη αυτήν που θα αποθηκευτεί στη θέση 1 (List[0].link=1), κ.ο.κ.

- Η αρχικοποίηση των τιμών του πίνακα HashTable όσο και του πίνακα List γίνεται με τη βοήθεια της διαδικασίας **CreateHashList**.

- Κατ' αρχήν θεωρούμε ότι **τα στοιχεία του πίνακα List σχηματίζουν συνδεδεμένη λίστα**, όπου το πρώτο στοιχείο θα αποθηκευτεί στη θέση 0 του πίνακα List, το δεύτερο στοιχείο θα αποθηκευτεί στη θέση 1 του πίνακα List, κ.ο.κ. και **η αρχή της συνδεδεμένης λίστας αποθηκεύεται στη μεταβλητή StackPtr**.

- Η συνδεδεμένη αυτή λίστα **λειτουργεί ως στοίβα**.

Παράδειγμα

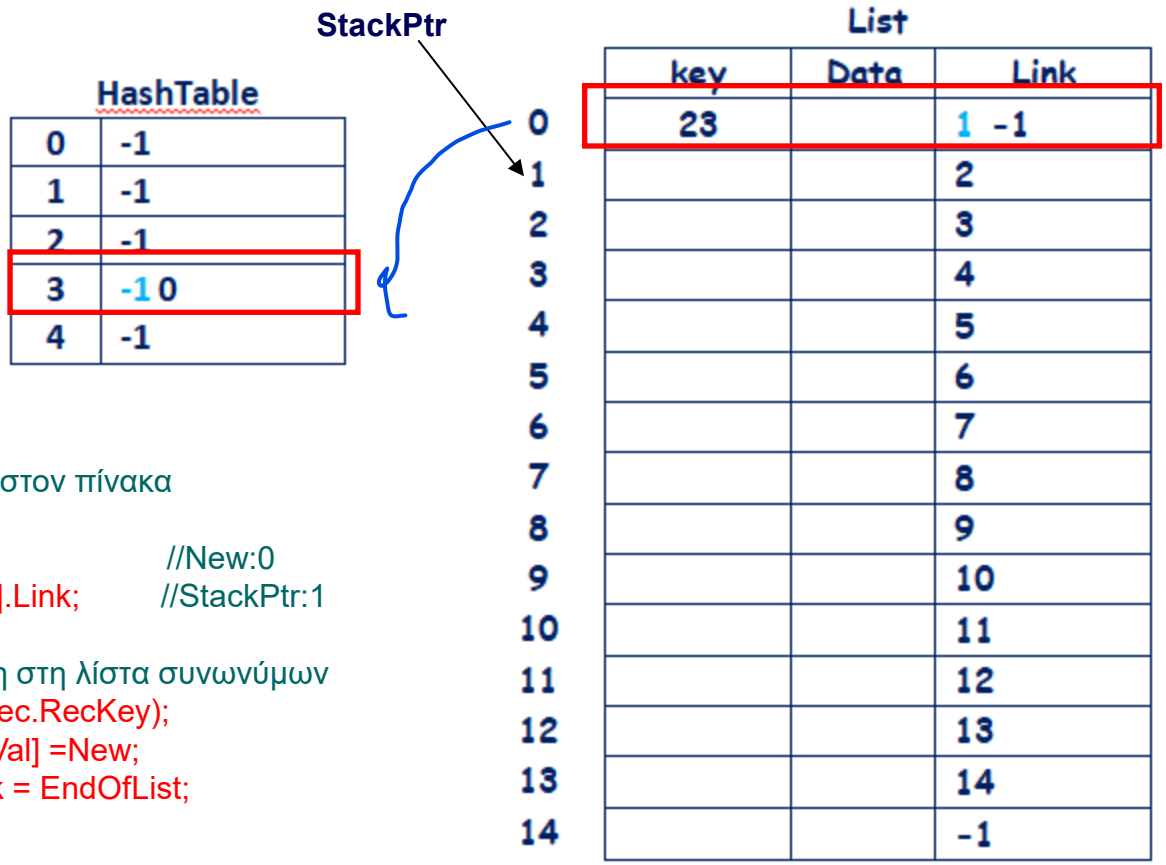
Αρχικά, η συνδεδεμένη λίστα List της δομής HashList είναι κενή και έστω ότι επιθυμούμε να εισάγουμε σ' αυτήν μια εγγραφή με τιμή κλειδιού 23.

Η συνάρτηση κατακερματισμού μας δίνει τιμή κατακερματισμού

HValue=(23 % 5) = 3

δηλαδή η εγγραφή θα προστεθεί στη υπολίστα νούμερο 3.

Η List δεν είναι γεμάτη ούτε και υπάρχει ήδη εγγραφή με κλειδί 23, επομένως η νέα εγγραφή θα είναι το πρώτο στοιχείο της λίστας List και στην θέση HValue=3 του πίνακα HashTable θα καταχωρηθεί η τιμή 0:




```
if (Loc == -1) {              //Η τιμή δεν βρέθηκε στον πίνακα
        HList->Size ++;
         New = HList->StackPtr;                          //New:0
        HList->StackPtr = HList->List[New].Link;        //StackPtr:1
        HList->List[New] = InRec;
       if (Pred == -1)  {    //1η καταχώρηση στη λίστα συνωνύμων
                        HVal = HashKey(InRec.RecKey);
                        HList->HashTable[HVal] =New;
                        HList->List[New].Link = EndOfList;
       }
       else {
                        HList->List[New].Link = HList->List[Pred].Link;
                        HList->List[Pred].Link = New;
        }
   }
```

Παράδειγμα

Έστω ότι η επόμενη εγγραφή που πρόκειται να εισάγουμε έχει τιμή κλειδιού 40.

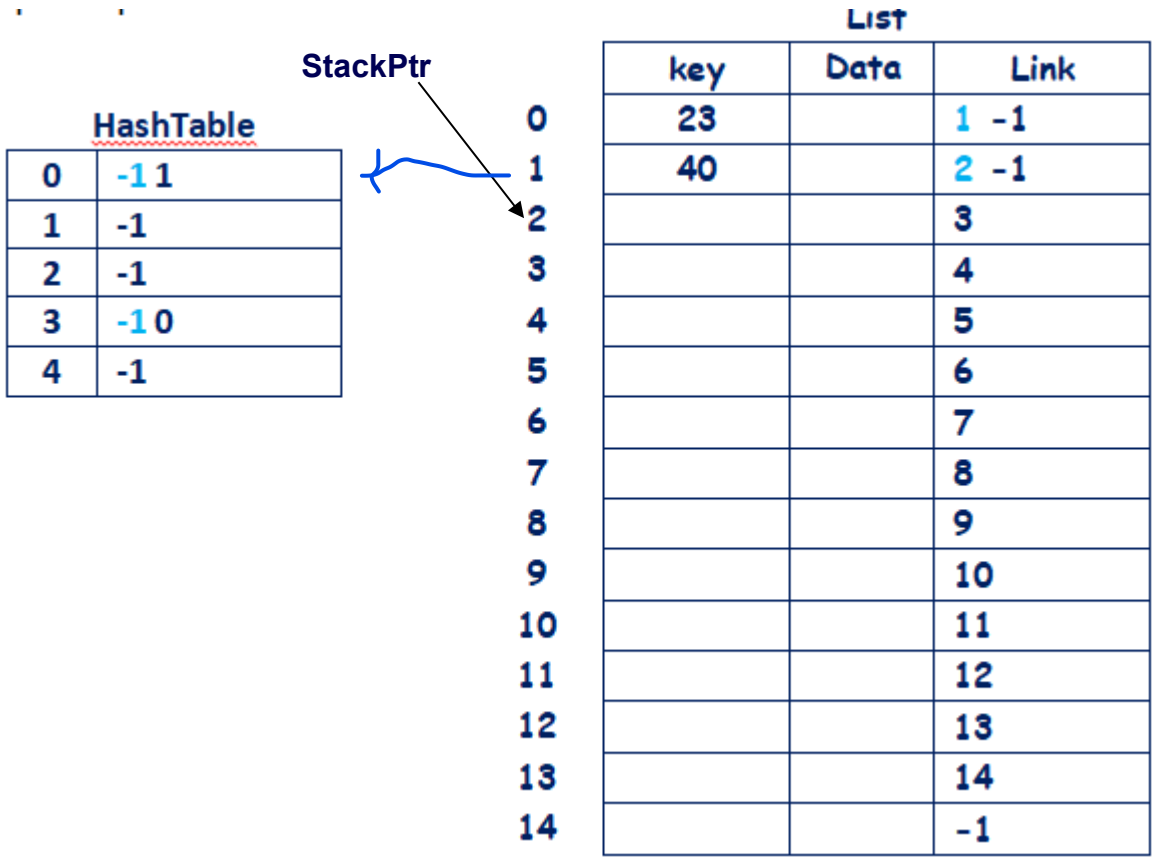
Η τιμή κατακερματισμού είναι

HValue=(40 % 5) = 0

δηλαδή η εγγραφή θα προστεθεί στην $0^{η}$ υπολίστα και στη θέση

HValue=0

του πίνακα HashTable θα καταχωρηθεί η τιμή 1:



Παράδειγμα

Αν τώρα εισαγάγουμε μια εγγραφή με κλειδί 71

η εισαγωγή αυτή θα γίνει στην υπολίστα νούμερο 1, γιατί

HValue= 71 % 5 = 1

και στην θέση

HValue=1

του πίνακα HashTable θα καταχωρηθεί η τιμή 2, γιατί η εισαγωγή γίνεται στη θέση 2 της λίστας List:

HashTable

0	-1 1
1	-1 2
2	-1
3	-1 0
4	-1

StackPtr

List

	key	Data	Link
0	23		1 -1
1	40		2 -1
2	71		3 -1
3			4
4			5
5			6
6			7
7			8
8			9
9			10
10			11
11			12
12			13
13			14
14			-1

Παράδειγμα

Στη συνέχεια, επιθυμούμε να εισάγουμε μια νέα εγγραφή με κλειδί 86,
δηλαδή η τιμή κατακερματισμού είναι
τώρα

HValue=86 % 5 = 1

και η εισαγωγή θα γίνει στη θέση 3 της λίστας List.

Επειδή *έχει ήδη εισαχθεί εγγραφή με αυτήν την τιμή κατακερματισμού* (η εγγραφή με κλειδί 71), *θα πρέπει να συνδεθούν αυτές οι δύο συνώνυμες εγγραφές για να σχηματιστεί η αντίστοιχη υπολίστα*.

HashTable

0	-1 1
1	-1 2
2	-1
3	-1 0
4	-1

List

StackPtr → 4

	key	Data	Link
0	23		1 -1
1	40		2 -1
2	71		3 -1 3
3	86		4 -1
4			5
5			6
6			7
7			8
8			9
9			10
10			11
11			12
12			13
13			14
14			-1

```
if (Loc == -1) {            //Η τιμή δεν βρέθηκε στον πίνακα
        HList->Size ++;
        New = HList->StackPtr;                          //New:3
        HList->StackPtr = HList->List[New].Link;        //StackPtr:4
        HList->List[New] = InRec;
        if (Pred == -1) {
                HVal = HashKey(InRec.RecKey);
                HList->HashTable[HVal] =New;
                HList->List[New].Link = EndOfList;
        }
        else {      //Το προηγούμενο στοιχείο στη λίστα συνωνύμων Pred:2
                HList->List[New].Link = HList->List[Pred].Link;
                HList->List[Pred].Link = New;
         }
    }
```

Η σύνδεση αυτή γίνεται αλλάζοντας την τιμή του πεδίου Link της 3ης εγγραφής (θέση 2) σε 3 ώστε να δείχνει στην θέση 3 του List:

Όπως φαίνεται από το παραπάνω σχήμα, η τιμή HashTable[1] δεν αλλάζει, γιατί δείχνει στην αρχή της 1ης υπολίστας, που είναι η θέση 2 του List.

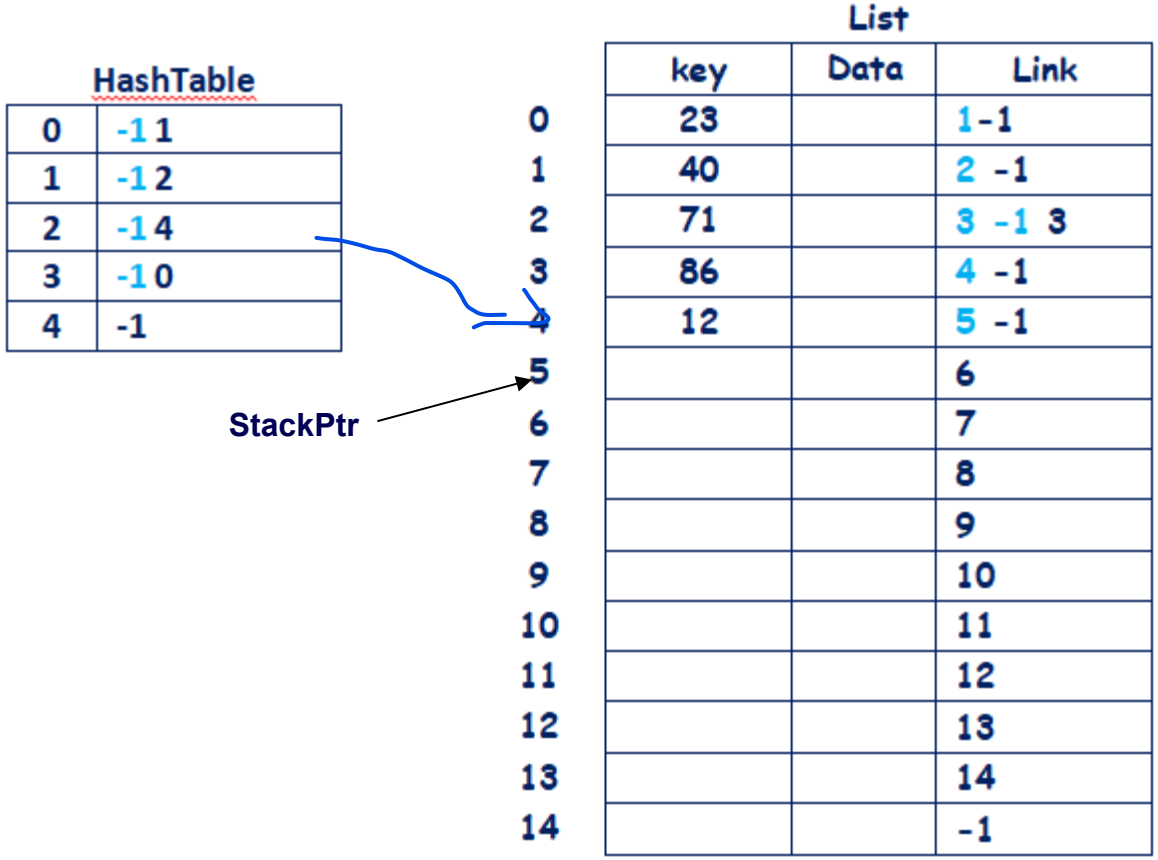
Παράδειγμα

Εν συνεχεία, έστω ότι θέλουμε να εισαγάγουμε μια εγγραφή με τιμή κλειδιού 12.

Επειδή

HValue=(12 % 5) = 2

και η εγγραφή θα τοποθετηθεί στην θέση 4 της λίστας List, στην θέση 2 του πίνακα καταχωρούμε την τιμή 4, όπως δείχνει και το διπλανό σχήμα:



Παράδειγμα

Έστω ότι ακολουθεί νέα εισαγωγή με τιμή κλειδιού 45.

Η εγγραφή αυτή θα τοποθετηθεί στην θέση 5 της λίστας List και θα ανήκει στην 0η υπολίστα συνωνύμων, αφού

HValue=(45 % 5) = 0

Επομένως, το πεδίο Link της εγγραφής με κλειδί 40 θα πάρει τιμή 5 για να δείχνει στην έκτη εγγραφή:

HashTable

0	-1 1
1	-1 2
2	-1 4
3	-1 0
4	-1

StackPtr → 6

List

	key	Data	Link
0	23		1 -1
1	40		2 -1 5
2	71		3 -1 3
3	86		4 -1
4	12		5 -1
5	45		6 -1
6			7
7			8
8			9
9			10
10			11
11			12
12			13
13			14
14			-1

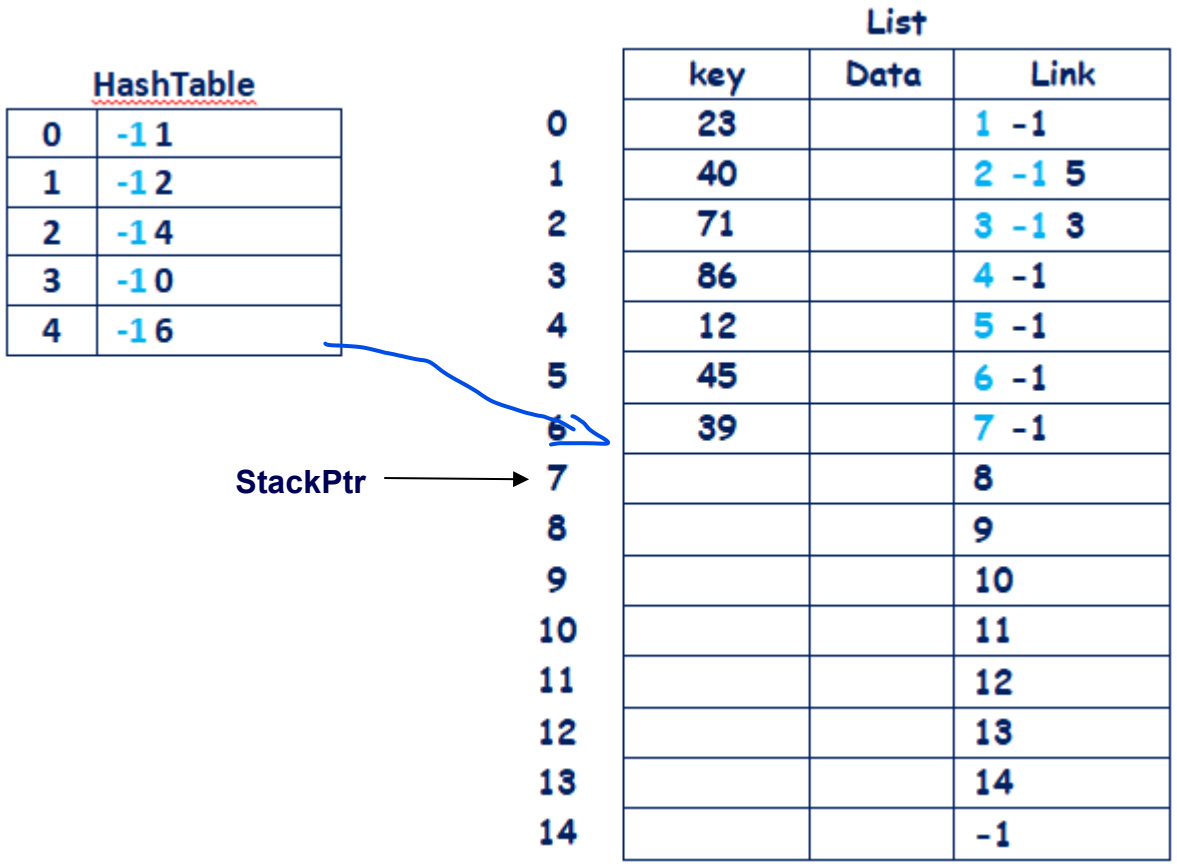
Παράδειγμα

Αν τώρα αποφασίσουμε να εισαγάγουμε μια εγγραφή με κλειδί 39,

θα πρέπει να αποθηκεύσουμε στην θέση

HValue=(39 % 5) = 4

του πίνακα HashTable την τιμή 6, γιατί η νέα εγγραφή θα τοποθετηθεί στην θέση 6 της λίστας List:



Παράδειγμα

Έστω ότι η επόμενη εγγραφή που θα εισαχθεί είναι μια εγγραφή με κλειδί 68

δηλαδή με τιμή κατακερματισμού

HValue=(68 % 5) = 3

Η εγγραφή αυτή θα βρίσκεται στην θέση 7 της λίστας List και θα ανήκει στην 3η υπολίστα συνωνύμων, οπότε θέτουμε την τιμή 7 στο πεδίο Link της εγγραφής με τιμή κλειδιού 23 (23, 68 συνώνυμα):

HashTable

0	-1 1
1	-1 2
2	-1 4
3	-1 0
4	-1 6

List

	key	Data	Link
0	23		1 -1 7
1	40		2 -1 5
2	71		3 -1 3
3	86		4 -1
4	12		5 -1
5	45		6 -1
6	39		7 -1
7	68		8 -1
StackPtr → 8			9
9			10
10			11
11			12
12			13
13			14
14			-1

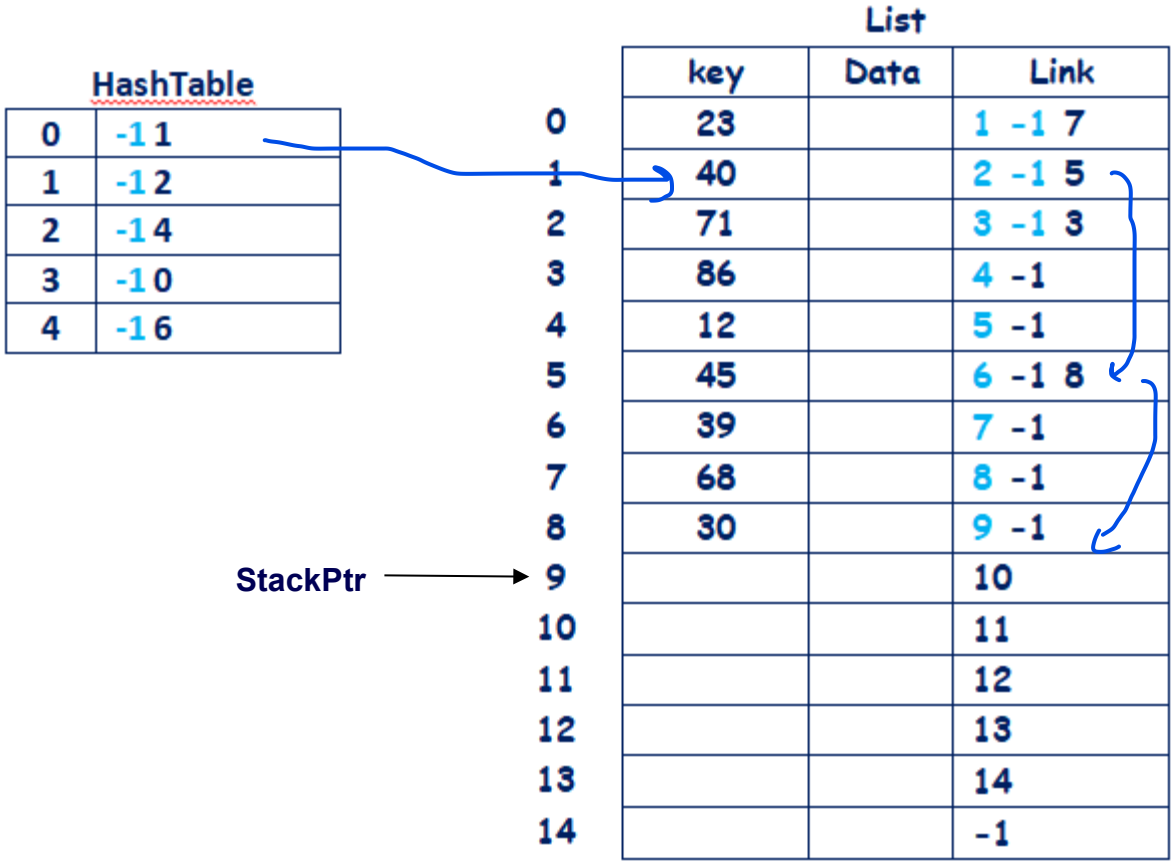
Παράδειγμα

Αν υποθέσουμε ότι ακολουθεί εισαγωγή μιας εγγραφής με κλειδί 30,

τότε η τιμή κατακερματισμού

HValue=(30 % 5) = 0

μας οδηγεί στο να αλλάξουμε την τιμή του πεδίου Link της εγγραφής με τιμή κλειδιού 45 σε 8, αφού η εισαγωγή θα γίνει στην θέση 8 της λίστας List:



Παράδειγμα

Κατά τον ίδιο τρόπο μπορούν να γίνουν και οι εισαγωγές άλλων στοιχείων.

Στα σχήματα που ακολουθούν φαίνονται οι εισαγωγές εγγραφών με κλειδιά 22, 3 και 54 αντίστοιχα:

HashTable

0	-1 1
1	-1 2
2	-1 4
3	-1 0
4	-1 6

Εισαγωγή εγγραφής με κλειδί 22

StackPtr →

List

	key	Data	Link
0	23		1 -1 7
1	40		2 -1 5
2	71		3 -1 3
3	86		4 -1
4	12		5 -1 9
5	45		6 -1 8
6	39		7 -1
7	68		8 -1
8	30		9 -1
9	22		10 -1
10			11
11			12
12			13
13			14
14			-1

Παράδειγμα

Κατά τον ίδιο τρόπο μπορούν να γίνουν και οι εισαγωγές άλλων στοιχείων.

Στα σχήματα που ακολουθούν φαίνονται οι εισαγωγές εγγραφών με κλειδιά 22, 3 και 54 αντίστοιχα:

HashTable

0	-1 1
1	-1 2
2	-1 4
3	-1 0
4	-1 6

Εισαγωγή εγγραφής με κλειδί 3

StackPtr →

List

	key	Data	Link
0	23		1 -1 7
1	40		2 -1 5
2	71		3 -1 3
3	86		4 -1
4	12		5 -1 9
5	45		6 -1 8
6	39		7 -1
7	68		8 -1 10
8	30		9 -1
9	22		10 -1
10	3		11 -1
11			12
12			13
13			14
14			-1

Παράδειγμα

Κατά τον ίδιο τρόπο μπορούν να γίνουν και οι εισαγωγές άλλων στοιχείων.

Στα σχήματα που ακολουθούν φαίνονται οι εισαγωγές εγγραφών με κλειδιά 22, 3 και 54 αντίστοιχα:

HashTable

0	-1 1
1	-1 2
2	-1 4
3	-1 0
4	-1 6

List

	key	Data	Link
0	23		1 -1 7
1	40		2 -1 5
2	71		3 -1 3
3	86		4 -1
4	12		5 -1 9
5	45		6 -1 8
6	39		7 -1 11
7	68		8 -1 10
8	30		9 -1
9	22		10 -1
10	3		11 -1
11	54		12 -1
12			13
13			14
14			-1

Εισαγωγή εγγραφής με κλειδί 54

StackPtr

Παράδειγμα - Αναζήτηση 39

Έστω ο πίνακας κατακερματισμού που δημιουργήθηκε μετά την διαδοχική εισαγωγή των κλειδιών: 23, 40, 71, 86, 12, 45, 39, 68, 30, 22 και 3.

HashTable	
0	1
1	2
2	4
3	0
4	6

	List		
	key	Data	Link
0	23		7
1	40		5
2	71		3
3	86		-1
4	12		9
5	45		8
6	39		-1
7	68		10
8	30		-1
9	22		-1
10	3		-1
11			12
12			13
13			14
14			-1

```
void SearchHashList(HashListType HList, KeyType KeyArg,
int *Loc, int *Pred)
{
    int HVal;

    HVal = HashKey(KeyArg);    //Hval:4 (=39%5)
     if (HList.HashTable[HVal] == EndOfList) {
          *Pred = -1;
          *Loc = -1;
    }
    else {
           HList.SubListPtr = HList.HashTable[HVal];
         SearchSynonymList(HList, KeyArg, &(*Loc), &(*Pred));
    }
}
```

Παράδειγμα - Αναζήτηση 39

Έστω ο πίνακας κατακερματισμού που δημιουργήθηκε μετά την διαδοχική εισαγωγή των κλειδιών: 23, 40, 71, 86, 12, 45, 39, 68, 30, 22 και 3.

HashTable	
0	1
1	2
2	4
3	0
4	6

HList.SubListPtr=6

	List		
	key	Data	Link
0	23		7
1	40		5
2	71		3
3	86		-1
4	12		9
5	45		8
6	39		-1
7	68		10
8	30		-1
9	22		-1
10	3		-1
11			12
12			13
13			14
14			-1

```
void SearchHashList(HashListType HList, KeyType KeyArg, int
*Loc, int *Pred)
{
    int HVal;

    Hval = HashKey(KeyArg);
    if (HList.HashTable[HVal] == EndOfList) {
        *Pred = -1;
        *Loc = -1;
    }
    else {
        HList.SubListPtr = HList.HashTable[HVal];
        SearchSynonymList(HList, KeyArg, &(*Loc), &(*Pred));
    }
}
```

Παράδειγμα - Αναζήτηση 39

Έστω ο πίνακας κατακερματισμού που δημιουργήθηκε μετά την διαδοχική εισαγωγή των κλειδιών: 23, 40, 71, 86, 12, 45, 39, 68, 30, 22 και 3.

HashTable	
0	1
1	2
2	4
3	0
4	6

	List		
	key	Data	Link
0	23		7
1	40		5
2	71		3
3	86		-1
4	12		9
5	45		8
6	39		-1
7	68		10
8	30		-1
9	22		-1
10	3		-1
11			12
12			13
13			14
14			-1

HList.SubListPtr=6

```
void SearchHashList(HashListType HList, KeyType KeyArg,
int *Loc, int *Pred)
{
    int HVal;

    Hval = HashKey(KeyArg);
    if (HList.HashTable[HVal] == EndOfList) {
          *Pred = -1;
          *Loc = -1;
    }
    else {
          HList.SubListPtr = HList.HashTable[HVal];
          SearchSynonymList(HList, KeyArg, &(*Loc), &(*Pred));
    }
}
```

Παράδειγμα - Αναζήτηση 39

Έστω ο πίνακας κατακερματισμού που δημιουργήθηκε μετά την διαδοχική εισαγωγή των κλειδιών: 23, 40, 71, 86, 12, 45, 39, 68, 30, 22 και 3.

HashTable	
0	1
1	2
2	4
3	0
4	6

HList.SubListPtr=6

	List		
	key	Data	Link
0	23		7
1	40		5
2	71		3
3	86		-1
4	12		9
5	45		8
6	39		-1
7	68		10
8	30		-1
9	22		-1
10	3		-1
11			12
12			13
13			14
14			-1

```
void SearchSynonymList(HashListType HList, KeyType KeyArg,
int *Loc, int *Pred)
{
   int Next;

    Next = HList.SubListPtr;   //Next=6
   *Loc = -1;
   *Pred = -1;
   while (Next != EndOfList) {
         if (HList.List[Next].RecKey == KeyArg) {
               *Loc = Next;
               Next = EndOfList;
         }
         else
            {
               *Pred = Next;
               Next = HList.List[Next].Link;
            }
      }
}
```

Παράδειγμα - Αναζήτηση 39

Έστω ο πίνακας κατακερματισμού που δημιουργήθηκε μετά την διαδοχική εισαγωγή των κλειδιών: 23, 40, 71, 86, 12, 45, 39, 68, 30, 22 και 3.

HashTable	
0	1
1	2
2	4
3	0
4	6

	List		
	key	Data	Link
0	23		7
1	40		5
2	71		3
3	86		-1
4	12		9
5	45		8
6	39		-1
7	68		10
8	30		-1
9	22		-1
10	3		-1
11			12
12			13
13			14
14			-1

```
void SearchSynonymList(HashListType HList, KeyType
KeyArg, int *Loc, int *Pred)
{
   int Next;

    Next = HList.SubListPtr;   //Next=6
   *Loc = -1;
   *Pred = -1;
   while (Next != EndOfList) {
         if (HList.List[Next].RecKey == KeyArg) {
               *Loc = Next;
               Next = EndOfList;
         }
         else
            {
               *Pred = Next;
               Next = HList.List[Next].Link;
            }
      }
}
```

*Επιστρέφει *Loc=6 & *Pred= -1, δηλαδή το 1ο στην αλυσίδα συνωνύμων*

Παράδειγμα - Αναζήτηση 22

Έστω ο πίνακας κατακερματισμού που δημιουργήθηκε μετά την διαδοχική εισαγωγή των κλειδιών: 23, 40, 71, 86, 12, 45, 39, 68, 30, 22 και 3.

HList.SubListPtr=4

HashTable	
0	1
1	2
2	4
3	0
4	6

```
void SearchHashList(HashListType HList, KeyType KeyArg, int
*Loc, int *Pred)
{
    int HVal;

    Hval = HashKey(KeyArg);   //Hval:2 (=22%5)
    if (HList.HashTable[HVal] == EndOfList) {
          *Pred = -1;
          *Loc = -1;
    }
    else {
          HList.SubListPtr = HList.HashTable[HVal];
          SearchSynonymList(HList, KeyArg, &(*Loc), &(*Pred));
    }
}
```

	List		
	key	Data	Link
0	23		7
1	40		5
2	71		3
3	86		-1
4	12		9
5	45		8
6	39		-1
7	68		10
8	30		-1
9	22		-1
10	3		-1
11			12
12			13
13			14
14			-1

Παράδειγμα - Αναζήτηση 22

Έστω ο πίνακας κατακερματισμού που δημιουργήθηκε μετά την διαδοχική εισαγωγή των κλειδιών: 23, 40, 71, 86, 12, 45, 39, 68, 30, 22 και 3.

HList.SubListPtr=4

HashTable	
0	1
1	2
2	4
3	0
4	6

	List		
	key	Data	Link
0	23		7
1	40		5
2	71		3
3	86		-1
4	12		9
5	45		8
6	39		-1
7	68		10
8	30		-1
9	22		-1
10	3		-1
11			12
12			13
13			14
14			-1

```
void SearchHashList(HashListType HList, KeyType KeyArg, int
*Loc, int *Pred)
{
    int HVal;

    Hval = HashKey(KeyArg);
    if (HList.HashTable[HVal] == EndOfList) {
          *Pred = -1;
          *Loc = -1;
    }
    else {
           HList.SubListPtr = HList.HashTable[HVal];
           SearchSynonymList(HList, KeyArg, &(*Loc), &(*Pred));
    }
}
```

Παράδειγμα - Αναζήτηση 22

Έστω ο πίνακας κατακερματισμού που δημιουργήθηκε μετά την διαδοχική εισαγωγή των κλειδιών: 23, 40, 71, 86, 12, 45, 39, 68, 30, 22 και 3.

HList.SubListPtr=4

HashTable	
0	1
1	2
2	4
3	0
4	6

	List		
	key	Data	Link
0	23		7
1	40		5
2	71		3
3	86		-1
4	12		9
5	45		8
6	39		-1
7	68		10
8	30		-1
9	22		-1
10	3		-1
11			12
12			13
13			14
14			-1

```
void SearchSynonymList(HashListType HList, KeyType
KeyArg, int *Loc, int *Pred)
{
   int Next;

    Next = HList.SubListPtr;   //Next=4
   *Loc = -1;
   *Pred = -1;
   while (Next != EndOfList) {
         if (HList.List[Next].RecKey == KeyArg) {
               *Loc = Next;
               Next = EndOfList;
         }
         else
            {
               *Pred = Next;
               Next = HList.List[Next].Link;
            }
      }
}
```

Παράδειγμα - Αναζήτηση 22

Έστω ο πίνακας κατακερματισμού που δημιουργήθηκε μετά την διαδοχική εισαγωγή των κλειδιών: 23, 40, 71, 86, 12, 45, 39, 68, 30, 22 και 3.

HashTable	
0	1
1	2
2	4
3	0
4	6

	List		
	key	Data	Link
0	23		7
1	40		5
2	71		3
3	86		-1
4	12		9
5	45		8
6	39		-1
7	68		10
8	30		-1
9	22		-1
10	3		-1
11			12
12			13
13			14
14			-1

```
void SearchSynonymList(HashListType HList, KeyType
KeyArg, int *Loc, int *Pred)
{
   int Next;

    Next = HList.SubListPtr;
    *Loc = -1;
   *Pred = -1;
   while (Next != EndOfList) {
         if (HList.List[Next].RecKey == KeyArg) {
               *Loc = Next;
               Next = EndOfList;
         }
         else
            {
               *Pred = Next;    //Pred=4
               Next = HList.List[Next].Link;   //Next=9
             }
      }
}
```

Παράδειγμα - Αναζήτηση 22

Έστω ο πίνακας κατακερματισμού που δημιουργήθηκε μετά την διαδοχική εισαγωγή των κλειδιών: 23, 40, 71, 86, 12, 45, 39, 68, 30, 22 και 3.

HashTable	
0	1
1	2
2	4
3	0
4	6

	List		
	key	Data	Link
0	23		7
1	40		5
2	71		3
3	86		-1
4	12		9
5	45		8
6	39		-1
7	68		10
8	30		-1
9	22		-1
10	3		-1
11			12
12			13
13			14
14			-1

```
void SearchSynonymList(HashListType HList, KeyType
KeyArg, int *Loc, int *Pred)
{
   int Next;

    Next = HList.SubListPtr;
    *Loc = -1;
   *Pred = -1;
   while (Next != EndOfList) {
         if (HList.List[Next].RecKey == KeyArg) {
               *Loc = Next;
               Next = EndOfList;
         }
         else
            {
               *Pred = Next;              //Pred=4
               Next = HList.List[Next].Link;         //Next=9
             }
      }
}
```

Παράδειγμα - Αναζήτηση 22

Έστω ο πίνακας κατακερματισμού που δημιουργήθηκε μετά την διαδοχική εισαγωγή των κλειδιών: 23, 40, 71, 86, 12, 45, 39, 68, 30, 22 και 3.

HashTable	
0	1
1	2
2	4
3	0
4	6

	List		
	key	Data	Link
0	23		7
1	40		5
2	71		3
3	86		-1
4	12		9
5	45		8
6	39		-1
7	68		10
8	30		-1
9	22		-1
10	3		-1
11			12
12			13
13			14
14			-1

```
void SearchSynonymList(HashListType HList, KeyType
KeyArg, int *Loc, int *Pred)
{
   int Next;

    Next = HList.SubListPtr;
    *Loc = -1;
   *Pred = -1;
   while (Next != EndOfList) {
         if (HList.List[Next].RecKey == KeyArg) {
               *Loc = Next;              //*Loc=9
               Next = EndOfList;         // Next=-1
         }
         else
            {
               *Pred = Next;             //Pred=4
               Next = HList.List[Next].Link;        //Next=9
             }
    }
}
```

*Επιστρέφει *Loc=9 & *Pred= 4, έχει συνώνυμο και είναι στην 4η θέση του List*

Παράδειγμα - Αναζήτηση 31

Έστω ο πίνακας κατακερματισμού που δημιουργήθηκε μετά την διαδοχική εισαγωγή των κλειδιών: 23, 40, 71, 86, 12, 45, 39, 68, 30, 22 και 3.

HList.SubListPtr=2

HashTable	
0	1
1	2
2	4
3	0
4	6

```
void SearchHashList(HashListType HList, KeyType KeyArg, int
*Loc, int *Pred)
{
    int HVal;

    Hval = HashKey(KeyArg);   //Hval:1 (=31%5)
    if (HList.HashTable[HVal] == EndOfList) {
          *Pred = -1;
          *Loc = -1;
    }
    else {
          HList.SubListPtr = HList.HashTable[HVal];
          SearchSynonymList(HList, KeyArg, &(*Loc), &(*Pred));
    }
}
```

	List		
	key	Data	Link
0	23		7
1	40		5
2	71		3
3	86		-1
4	12		9
5	45		8
6	39		-1
7	68		10
8	30		-1
9	22		-1
10	3		-1
11			12
12			13
13			14
14			-1

Παράδειγμα - Αναζήτηση 31

Έστω ο πίνακας κατακερματισμού που δημιουργήθηκε μετά την διαδοχική εισαγωγή των κλειδιών: 23, 40, 71, 86, 12, 45, 39, 68, 30, 22 και 3.

HList.SubListPtr=2

HashTable	
0	1
1	2
2	4
3	0
4	6

```
void SearchHashList(HashListType HList, KeyType KeyArg, int
*Loc, int *Pred)
{
    int HVal;

    Hval = HashKey(KeyArg);
    if (HList.HashTable[HVal] == EndOfList) {
          *Pred = -1;
          *Loc = -1;
    }
    else {
           HList.SubListPtr = HList.HashTable[HVal];
           SearchSynonymList(HList, KeyArg, &(*Loc), &(*Pred));
    }
}
```

	List		
	key	Data	Link
0	23		7
1	40		5
2	71		3
3	86		-1
4	12		9
5	45		8
6	39		-1
7	68		10
8	30		-1
9	22		-1
10	3		-1
11			12
12			13
13			14
14			-1

Παράδειγμα - Αναζήτηση 31

Έστω ο πίνακας κατακερματισμού που δημιουργήθηκε μετά την διαδοχική εισαγωγή των κλειδιών: 23, 40, 71, 86, 12, 45, 39, 68, 30, 22 και 3.

HList.SubListPtr=2

HashTable	
0	1
1	2
2	4
3	0
4	6

	List		
	key	Data	Link
0	23		7
1	40		5
2	71		3
3	86		-1
4	12		9
5	45		8
6	39		-1
7	68		10
8	30		-1
9	22		-1
10	3		-1
11			12
12			13
13			14
14			-1

```
void SearchSynonymList(HashListType HList, KeyType
KeyArg, int *Loc, int *Pred)
{
   int Next;

    Next = HList.SubListPtr;   //Next=2
   *Loc = -1;                  // *Loc = -1
   *Pred = -1;                 // *Pred = -1
   while (Next != EndOfList) {
         if (HList.List[Next].RecKey == KeyArg) {
               *Loc = Next;
               Next = EndOfList;
         }
         else
            {
               *Pred = Next;
               Next = HList.List[Next].Link;
            }
      }
}
```

Παράδειγμα - Αναζήτηση 31

Έστω ο πίνακας κατακερματισμού που δημιουργήθηκε μετά την διαδοχική εισαγωγή των κλειδιών: 23, 40, 71, 86, 12, 45, 39, 68, 30, 22 και 3.

HashTable	
0	1
1	2
2	4
3	0
4	6

	List		
	key	Data	Link
0	23		7
1	40		5
2	71		3
3	86		-1
4	12		9
5	45		8
6	39		-1
7	68		10
8	30		-1
9	22		-1
10	3		-1
11			12
12			13
13			14
14			-1

```
void SearchSynonymList(HashListType HList, KeyType
KeyArg, int *Loc, int *Pred)
{
   int Next;

    Next = HList.SubListPtr;
    *Loc = -1;              // *Loc = -1
   *Pred = -1;
   while (Next != EndOfList) {
         if (HList.List[Next].RecKey == KeyArg) {
               *Loc = Next;
               Next = EndOfList;
         }
         else
            {
               *Pred = Next;    //Pred=2
               Next = HList.List[Next].Link;   //Next=3
             }
      }
}
```

Παράδειγμα - Αναζήτηση 31

Έστω ο πίνακας κατακερματισμού που δημιουργήθηκε μετά την διαδοχική εισαγωγή των κλειδιών: 23, 40, 71, 86, 12, 45, 39, 68, 30, 22 και 3.

HashTable	
0	1
1	2
2	4
3	0
4	6

	List		
	key	Data	Link
0	23		7
1	40		5
2	71		3
3	86		-1
4	12		9
5	45		8
6	39		-1
7	68		10
8	30		-1
9	22		-1
10	3		-1
11			12
12			13
13			14
14			-1

```
void SearchSynonymList(HashListType HList, KeyType
KeyArg, int *Loc, int *Pred)
{
   int Next;

    Next = HList.SubListPtr;
    *Loc = -1;              // *Loc = -1
   *Pred = -1;
   while (Next != EndOfList) {
         if (HList.List[Next].RecKey == KeyArg) {
               *Loc = Next;
               Next = EndOfList;
         }
         else
            {
               *Pred = Next;    //Pred=3
               Next = HList.List[Next].Link;   //Next=-1
             }
      }
}
```

*Επιστρέφει *Loc=-1 άρα δεν υπάρχει το στοιχείο στη ΔΔ*

Διαγραφή εγγραφής

Όταν πρόκειται να διαγράψουμε μια εγγραφή, **καλούμε πρώτα τη διαδικασία SearchHashList για να την εντοπίσουμε** και, εφόσον υπάρχει, τη διαγράφουμε διακρίνοντας δύο περιπτώσεις:

(α) η εγγραφή έχει προηγούμενη και

(β) η εγγραφή είναι η πρώτη της υπολίστας στην οποία ανήκει.

Στην

περίπτωση (α):

χρειάζεται να αλλάξουμε την τιμή του πεδίου Link της προηγούμενης εγγραφής ώστε να δείχνει στην επόμενη αυτής που θα διαγραφεί,

ενώ στην

περίπτωση (β):

- βρίσκουμε την τιμή κατακερματισμού HValue και
- αλλάζουμε την καταχώρηση HashTable[HValue] ώστε να δείχνει στην θέση της δεύτερης εγγραφής της αντίστοιχης υπολίστας.

Διαγραφή εγγραφής

Και στις δυο περιπτώσεις:

- χρειάζεται να θέσουμε την τιμή StackPtr στο πεδίο Link της διαγραμμένης εγγραφής και
- ο δείκτης StackPtr να δείχνει στη θέση της διαγραμμένης εγγραφής, ώστε να γίνει αυτή η πρώτη διαθέσιμη θέση της λίστας List.

Η διαδικασία της διαγραφής είναι η εξής:

Διαδικασία διαγραφής εγγραφής

void DeleteRec(HashListType *HList, KeyType DelKey)

/*Δέχεται: Μια δομή *HList* και το κλειδί *DelKey* της εγγραφής που πρόκειται να διαγραφεί.

Λειτουργία: Διαγράφει την εγγραφή με κλειδί *DelKey* από τη λίστα *List*, αν υπάρχει, και ενημερώνει τη δομή *HList*.

Επιστρέφει: Την τροποποιημένη δομή *HList*.

Έξοδος: Αν δεν υπάρχει εγγραφή με αυτό το κλειδί, εμφάνιση αντίστοιχου μηνύματος.*/

Διαδικασία διαγραφής εγγραφής

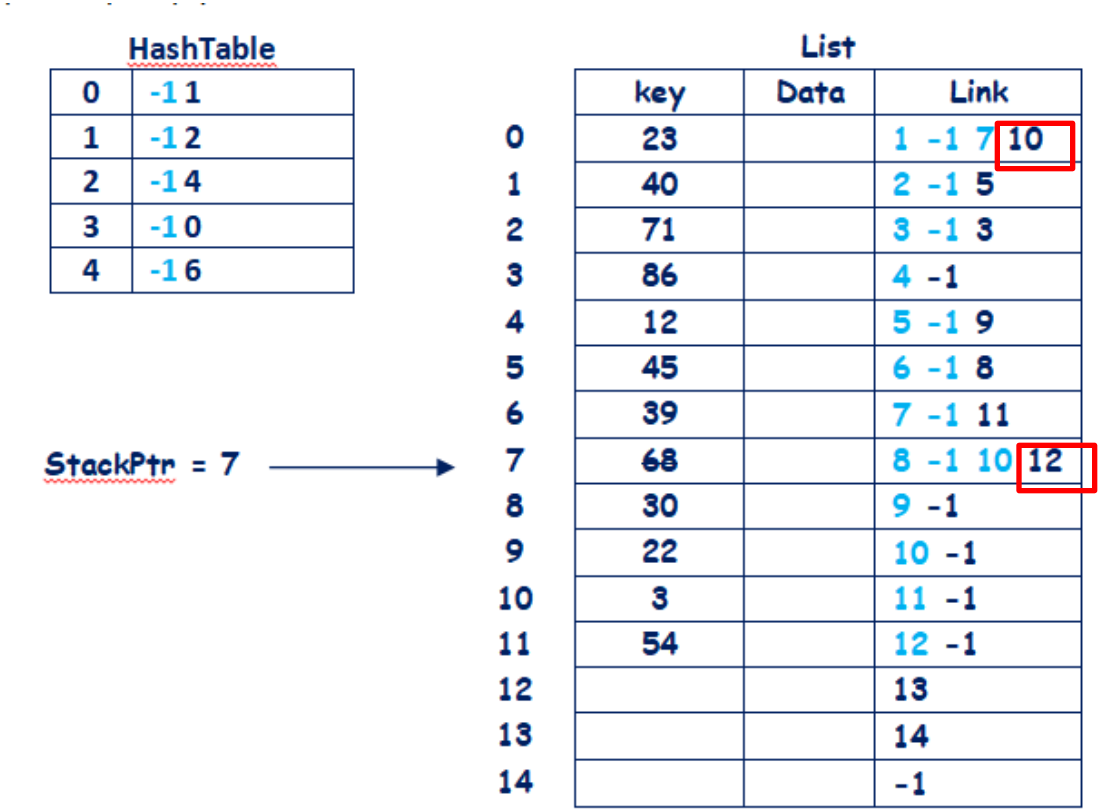
```
{
   int Loc, Pred, New, HVal;

   SearchHashList(*HList, DelKey, &Loc, &Pred);
   if (Loc != -1) {                /*Η εγγραφή υπάρχει στη λίστα*/
         if (Pred != -1)           /*Η εγγραφή έχει προηγούμενη*/
                  HList->List[Pred].Link = HList->List[Loc].Link;
         else {                    /*Η εγγραφή δεν έχει προηγούμενη*/
                  HVal  = HashKey(DelKey);
                  HList ->HashTable[HVal] = HList->List[Loc]. Link;
          }
         HList ->List[Loc].Link = HList ->StackPtr;
         HList ->StackPtr = Loc;
         HList ->Size--;
   }
   else
         printf("DEN YPARXEI  EGGRAFH ME KLEIDI %d \n", DelKey);
}
```

Διαδικασία διαγραφής εγγραφής: *παράδειγμα*

Έστω ότι θέλουμε να διαγράψουμε την εγγραφή με κλειδί 68.

Η εγγραφή αυτή ανήκει στην 3η υπολίστα συνωνύμων, αφού HValue=(68 % 5) = 3 και δεν είναι η πρώτη εγγραφή της υπολίστας αυτής. Επομένως:

1. θέτουμε το πεδίο Link της εγγραφής με κλειδί 23, δηλαδή της προηγούμενης εγγραφής, ίσο με τη θέση της επόμενης εγγραφής, δηλαδή 10.
2. Ο δείκτης StackPtr έχει τιμή 12, επομένως στο πεδίο Link της εγγραφής με κλειδί 68 θέτουμε την τιμή 12, ενώ
3. ο δείκτης StackPtr παίρνει τώρα τιμή 7:



Διαδικασία διαγραφής εγγραφής: *παράδειγμα*

Έστω ότι θέλουμε να διαγράψουμε την εγγραφή με κλειδί 68.

Η εγγραφή αυτή ανήκει στην 3η υπολίστα συνωνύμων, αφού

HValue=(68 % 5) = 3 και δεν είναι η πρώτη εγγραφή της υπολίστας αυτής. Επομένως:

1. θέτουμε το πεδίο Link της εγγραφής με κλειδί 23, δηλαδή της προηγούμενης εγγραφής, ίσο με τη θέση της επόμενης εγγραφής, δηλαδή 10.
2. Ο δείκτης StackPtr έχει τιμή 12, επομένως στο πεδίο Link της εγγραφής με κλειδί 68 θέτουμε την τιμή 12, ενώ
3. ο δείκτης StackPtr παίρνει τώρα τιμή 7:

HashTable

0	-1 1
1	-1 2
2	-1 4
3	-1 0
4	-1 6

(3)

StackPtr = 7

List

	key	Data	Link	
0	23		1 -1 7 10	(1)
1	40		2 -1 5	
2	71		3 -1 3	
3	86		4 -1	
4	12		5 -1 9	
5	45		6 -1 8	
6	39		7 -1 11	
7	68		8 -1 10 12	(2)
8	30		9 -1	
9	22		10 -1	
10	3		11 -1	
11	54		12 -1	
12			13	
13			14	
14			-1	

```
if (Loc != -1) {          /Η εγγραφή υπάρχει στη λίστα      Loc : 7 */
    if (Pred != -1)       /*Η εγγραφή έχει προηγούμενη     Pred : 0*/
        HList->List[Pred].Link = HList->List[Loc].Link;          (1)
    else {                /*Η εγγραφή δεν έχει προηγούμενη*/
        HVal  = HashKey(DelKey);
        HList ->HashTable[HVal] = HList->List[Loc]. Link;
    }
    HList ->List[Loc].Link = HList ->StackPtr;                   (2)
    HList ->StackPtr = Loc;                                      (3)
    HList ->Size--;
```

Διαδικασία διαγραφής εγγραφής: *παράδειγμα*

Αν επιθυμούμε να διαγράψουμε την εγγραφή με τιμή κλειδιού 12 (12 % 5) = 2,

δηλαδή την πρώτη εγγραφή της 2ης υπολίστας συνωνύμων, τότε χρειάζεται:

1. να θέσουμε την τιμή 9 στην θέση 2 του πίνακα HashTable, γιατί η πρώτη εγγραφή της 2ης υπολίστας θα είναι τώρα η εγγραφή με τιμή κλειδιού 22, που βρίσκεται στη θέση 9 της λίστας List.
2. Στο πεδίο Link της διαγραμμένης εγγραφής θέτουμε την τιμή StackPtr=7 και
3. ο δείκτης StackPtr παίρνει τιμή 4:

HashTable

0	-1 1
1	-1 2
2	-1 4 9
3	-1 0
4	-1 6

(1)

(3)

StackPtr = 4

List

	key	Data	Link
0	23		1 -1 7 10
1	40		2 -1 5
2	71		3 -1 3
3	86		4 -1
4	~~12~~		5 -1 9 7
5	45		6 -1 8
6	39		7 -1 11
7	~~68~~		8 -1 10 12
8	30		9 -1
9	22		10 -1
10	3		11 -1
11	54		12 -1
12			13
13			14
14			-1

(2)

```
if (Loc != -1) {            /*Η εγγραφή υπάρχει στη λίστα*/
    if (Pred != -1)         /*Η εγγραφή έχει προηγούμενη*/
        HList->List[Pred].Link = HList->List[Loc].Link;
    else {                  /*Η εγγραφή δεν έχει προηγούμενη*/
        HVal  = HashKey(DelKey);
        HList ->HashTable[HVal] = HList->List[Loc]. Link;   (1)
    }
    HList ->List[Loc].Link = HList ->StackPtr;              (2)
    HList ->StackPtr = Loc;                                 (3)
    HList ->Size--;
```

Διαδικασία διαγραφής εγγραφής: *παράδειγμα*

Από τα αριθμητικά παραδείγματα φαίνεται ότι **στη δομή List αποθηκεύονται οι υπολίστες συνωνύμων**, ως συνδεδεμένες λίστες, αλλά και οι εγγραφές που είναι διαθέσιμες για αποθήκευση στοιχείων.

Για τις εγγραφές αυτές διατηρούμε μια στοίβα, **της οποίας η κορυφή αποθηκεύεται στη μεταβλητή StackPtr** και **η διεύθυνση του επόμενου στοιχείου της στοίβας αποθηκεύεται στο πεδίο Link**.

Εναλλακτική υλοποίηση

Υπάρχει και μια εναλλακτική υλοποίηση κατακερματισμού με αλυσιδωτή σύνδεση χωρίς να χρειάζεται ο πίνακας HashTable:

- Αν υπάρχουν k πιθανές τιμές κατακερματισμού, τότε **κρατούμε τις k πρώτες θέσεις του πίνακα List για τις πρώτες εγγραφές των k υπολιστών** αντίστοιχα.
- Όταν πρόκειται να εισαγάγουμε μια νέα εγγραφή στη λίστα, κατακερματίζουμε το κλειδί της για να βρούμε τη θέση μέσα στη λίστα List, όπου πρέπει να τοποθετήσουμε την εγγραφή αυτή.
- Αν η θέση δεν είναι κατειλημμένη, τότε η εγγραφή τοποθετείται εκεί και αποτελεί την αρχή μιας υπολίστας συνωνύμων.
- Αν η θέση είναι κατειλημμένη από άλλη εγγραφή, τότε τοποθετούμε τη νέα εγγραφή στην πρώτη διαθέσιμη θέση της λίστας List μετά από τις πρώτες k θέσεις και τη συνδέουμε με την αντίστοιχη υπολίστα συνωνύμων.

Εναλλακτική υλοποίηση

Αν έχουμε τις τιμές κατακερματισμού

0, 1, 2, 3, 4

όπως παραπάνω, και εισαγάγουμε τις ίδιες εγγραφές με την ίδια σειρά, όπως στο προηγούμενο παράδειγμα, δηλαδή οι τιμές των κλειδιών είναι με τη σειρά

23, 40, 71, 23, 86, 12, 45, 39, 68, 30, 22, 3 και 54,

τότε οι καταχωρήσεις στον πίνακα List είναι ως εξής:

List

	key	Data	Link
0	40		6
1	71		5
2	12		9
3	23		7
4	39		11
5	86		
6	45		8
7	68		10
8	30		
9	22		
10	3		
11	54		
12			
13			
14			

Ανοιχτή διευθυνσιοδότηση

Στην τεχνική της **ανοιχτής διευθυνσιοδότησης** (**open addressing**), όταν συμβαίνει κάποια σύγκρουση, γίνεται εξέταση ή ανίχνευση της περιοχής των εγγραφών σύμφωνα με κάποιο προκαθορισμένο σχήμα για να εντοπιστεί μια κενή σχισμή για τη νέα εγγραφή.

Αργότερα, όταν αναζητείται αυτή η εγγραφή, χρησιμοποιείται το ίδιο σχήμα ανίχνευσης για τον εντοπισμό της.

Ανοιχτή διευθυνσιοδότηση

Η πιο απλή μορφή εξέτασης που χρησιμοποιείται στην μέθοδο της ανοιχτής διευθυνσιοδότησης είναι η **γραμμική εξέταση** (**linear probing**).

Στην γραμμική εξέταση, όταν θέλουμε να εισαγάγουμε μια εγγραφή και η σχισμή στην οποία πρέπει να τοποθετηθεί είναι κατειλημμένη, ελέγχουμε αν η επόμενη σειριακά σχισμή (ή κάδος) είναι ελεύθερη:

- Έτσι, δηλαδή, αν η σχισμή στη θέση k της λίστας είναι κατειλημμένη, τότε η επόμενη θέση που εξετάζεται είναι η k+1.
- Για την ακρίβεια, η θέση που εξετάζεται είναι η (k+1)%(VMax), πράγμα που σημαίνει ότι η εξέταση γίνεται κατά κυκλικό τρόπο, δηλαδή, αν φτάσει στην τελευταία θέση του πίνακα, η διαδικασία συνεχίζεται στην πρώτη θέση.
- Η αναζήτηση, λοιπόν, προχωρά γραμμικά στις διαδοχικές θέσεις του πίνακα μέχρις ότου βρεθεί κενός χώρος ή προσπελαστεί πάλι η αρχική θέση (που σημαίνει ότι ο πίνακας είναι γεμάτος).

Αργότερα, αφού έχει δημιουργηθεί η δομή της λίστας με κατακερματισμό, χρησιμοποιείται η ίδια σειρά εξέτασης για την αναζήτηση μιας εγγραφής.

Ανοιχτή διευθυνσιοδότηση

Ένα μεγάλο μειονέκτημα της γραμμικής εξέτασης σε σχέση με άλλες τεχνικές εξέτασης είναι το πρόβλημα της **συγκέντρωσης** (**clustering**).

Συγκέντρωση είναι η τάση που έχουν οι εγγραφές να συγκεντρώνονται γύρω από μια περιοχή της λίστας, όπου έχουν συμβεί μία ή παραπάνω συγκρούσεις.

Η συγκέντρωση εμφανίζεται στην περίπτωση της γραμμικής εξέτασης, γιατί μια σύγκρουση προκαλεί την επόμενη σειριακά διαθέσιμη θέση να χρησιμοποιηθεί για την επίλυση της σύγκρουσης.

Αυτό έχει ως αποτέλεσμα να υπάρχει μεγαλύτερη πιθανότητα για επόμενες συγκρούσεις σ' αυτήν την γειτονιά, οπότε προκύπτουν περισσότερες συγκρούσεις και αυξάνεται η συγκέντρωση.

Ανοιχτή διευθυνσιοδότηση

Όταν οι εγγραφές συγκεντρώνονται ως αποτέλεσμα της γραμμικής εξέτασης, μια αναζήτηση για μια εγγραφή που έπρεπε να τοποθετηθεί σε άλλη θέση λόγω σύγκρουσης μπορεί να εκφυλιστεί σε μια υπερβολικά μεγάλη σειριακή αναζήτηση.

Ωστόσο, αν το σχήμα εξέτασης που χρησιμοποιείται, κατανέμει καλύτερα τις εγγραφές μέσα στη λίστα, όταν γίνονται συγκρούσεις, τότε οι περισσότερες αναζητήσεις, που απαιτούν εξέταση, μπορούν να παραμείνουν σχετικά μικρές.