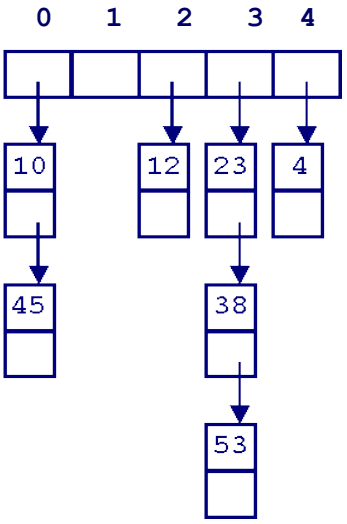
6.2 Χειρισμός συγκρούσεων

ΑΛΥΣΙΔΩΤΗ ΣΥΝΔΕΣΗ (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.

Αν για παράδειγμα 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:



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.

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=N/k και η σχέση υπολογισμού της τιμής κατακερματισμού είναι:

$$\text{HashValue} = \text{KeyValue} \% b$$

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

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.

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 (αυτές που έχουν άρτιο αριθμό).

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.

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.

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.

Οι απαραίτητες δηλώσεις είναι:

```
#define HMax 5          /*όριο μεγέθους του πίνακα HashTable*/
#define VMax 30         /*όριο μεγέθους της λίστας*/
#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 struct  {
                  int RecKey;
                  ListElementType Data;
                  /*τα υπόλοιπα πεδία*/
                  int Link;
} ListElm;

typedef struct    {
                        int HashTable[HMax]; /*πίνακας δεικτών προς τις υπολίστες
                                               συνωνύμων*/
                        int Size;             /*πλήθος εγγραφών της λίστας List*/
                        int  SubListPtr;      /*δείκτης σε μια υπολίστα*/
                        int StackPtr; /*δείκτης προς την πρώτη ελεύθερη θέση της
                                        λίστας List*/
                        ListElm List[VMax];
} HashListType;
```

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.

Για την κατασκευή μιας κενής δομής όπως η παραπάνω χρησιμοποιείται ο αλγόριθμος που παρουσιάζεται στη συνέχεια:

ΑΛΓΟΡΙΘΜΟΣ ΔΗΜΙΟΥΡΓΙΑΣ ΚΕΝΗΣ ΔΟΜΗΣ ΓΙΑ ΤΗ ΜΕΘΟΔΟ ΤΗΣ ΑΛΥΣΙΔΩΤΗΣ ΣΥΝΔΕΣΗΣ

/*Λειτουργία: Δημιουργεί μια δομή *HList*, η οποία περιλαμβάνει:

έναν πίνακα *HashTable* με δείκτες προς τις υπολίστες συνωνύμων

μια λίστα εγγραφών *List*

το πλήθος *Size* των εγγραφών της λίστας *List*

έναν δείκτη *SubListPtr* σε μια υπολίστα συνωνύμων, στην οποία - κατά περίπτωση - θα αναζητηθεί, θα εισαχθεί, θα διαγραφεί μια εγγραφή

έναν δείκτη *StackPtr* προς την πρώτη ελεύθερη θέση της λίστας *List*

Επιστρέφει: Την δομή *HList*.*/

EndOfList ← -1

/*σημαία που σηματοδοτεί το τέλος της λίστας και της κάθε υπολίστας συνωνύμων*/

Size ← 0

/*θέσε στη μεταβλητή *Size*, στην οποία αποθηκεύεται το πλήθος των εγγραφών της λίστας *List*, την τιμή 0*/

StackPtr ← 0

/*ενημέρωσε τον δείκτη *StackPtr*, ο οποίος δείχνει στην πρώτη ελεύθερη θέση της λίστας *List*, ώστε να δείχνει στην 1η θέση της λίστας αυτής*/

/*Δημιουργία της στοίβας των ελεύθερων θέσεων στη λίστα List*/

Index ← 0

/*θέσε στη μεταβλητή *Index* την τιμή 0 ώστε να δείχνει στην 1η θέση της λίστας *List**/

Όσο *Index* < *VMax*-1 **επανάλαβε**

/*όσο η τιμή της μεταβλητής *Index* είναι μικρότερη από την τιμή της *VMax*-1, δηλαδή όσο δεν έχουμε ενημερώσει το τμήμα δεσμού της προτελευταίας εγγραφής που βρίσκεται στη θέση *VMax*-1 της λίστας *List**/

List[*Index*].*Link* ← *Index* + 1

/*ενημέρωσε το τμήμα δεσμού *Link* της εγγραφής που βρίσκεται στη θέση *Index* της λίστας *List* ώστε να δείχνει στην εγγραφή που βρίσκεται στη θέση *Index* + 1 της λίστας *List**/

Index ← *Index* + 1

/*αύξησε τη μεταβλητή *Index* κατά 1 ώστε να δείχνει στην αμέσως επόμενη εγγραφή της λίστας *List**/

Τέλος_ επανάληψης

List[*Index*].*Link* ← *EndOfList*

/*θέσε στο τμήμα δεσμού *Link* της εγγραφής που βρίσκεται στη θέση *Index* της λίστας *List* την τιμή της *EndOfList*, δηλαδή το τμήμα δεσμού της τελευταίας εγγραφής που βρίσκεται στη θέση *VMax*-1 της λίστας *List* παίρνει την τιμή σημαία -1 που δηλώνει το τέλος της λίστας*/

/*Αρχικοποίηση του πίνακα *HashTable**/

Index ← 0

/*θέσε στη μεταβλητή *Index* την τιμή 0 ώστε να δείχνει στην 1η θέση του πίνακα *HashTable**/

Όσο *Index* < *HMax* **επανάλαβε**

/*όσο η τιμή της μεταβλητής *Index* είναι μικρότερη από την τιμή της *HMax,* δηλαδή όσο δεν έχουμε αρχικοποιήσει και τον τελευταίο δείκτη που βρίσκεται στην *HMax*-1 θέση *του* πίνακα *HashTable**/

HashTable[*Index*] ← *EndOfList*

/*θέσε στον δείκτη που βρίσκεται στην *Index* θέση του πίνακα *HashTable* την τιμή σημαία της *EndOfList,* αφού η αντίστοιχη υπολίστα συνωνύμων είναι κενή*/

Index ← *Index* + 1

/*αύξησε τη μεταβλητή *Index* κατά 1 ώστε να δείχνει στην αμέσως επόμενη θέση του πίνακα *HashTable**/

Τέλος_ επανάληψης

Η διαδικασία CreateHashList υλοποιεί τον παραπάνω αλγόριθμο:

```
void CreateHashList(HashListType *HList)
/*Λειτουργία:            Δημιουργεί μια δομή HList.
  Επιστρέφει:            Την δομή HList.*/
{
    int index;

    HList->Size = 0;
    HList->StackPtr = 0;
    /*Δημιουργία της στοίβας των ελεύθερων θέσεων στη λίστα List*/
    index = 0;
    while(index < VMax-1)
    {
          HList->List[index].Link = index+1;
          HList->List[index].Data = 0;
          index = index+1;
    }
    HList->List[index].Link = EndOfList;
    /*Αρχικοποίηση του πίνακα HashTable*/
    index = 0;
    while (index < HMax)
    {
          HList->HashTable[index] = EndOfList;
          index = index+1;
    }

}
```

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);
}
```

Εφόσον η συνδεδεμένη λίστα List δεν είναι γεμάτη, εξετάζουμε αν υπάρχει ήδη κάποια εγγραφή με κλειδί ίδιο με αυτό της εγγραφής που επιθυμούμε να εισάγουμε. Ο έλεγχος αυτός γίνεται με τη βοήθεια των δύο αλγορίθμων που παρουσιάζονται στη συνέχεια:

ΑΛΓΟΡΙΘΜΟΣ ΑΝΑΖΗΤΗΣΗΣ ΜΙΑΣ ΤΙΜΗΣ ΣΤΗ ΔΟΜΉ *HList*

/*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.*/

Υπολόγισε τη τιμή κατακερματισμού *HVal* για το κλειδί *KeyArg*

Αν *HashTable*[*HVal*] = *EndOfList* **τότε**

/*αν ο δείκτης που βρίσκεται στη θέση *HVal* του πίνακα *HashTable* έχει την τιμή της μεταβλητής-σημαίας *EndOfList,* δηλαδή αν η υπολίστα συνωνύμων που αντιστοιχεί στην τιμή κατακερματισμού *HVal* είναι κενή*/

/*θέσε στις μεταβλητές *Loc* και *PredPtr* την τιμή -1, εφόσον δεν υπάρχει εγγραφή με κλειδί *KeyArg* στη λίστα*/

Pred ← -1

Loc ← -1

Αλλιώς

SubListPtr ← *HashTable*[*HVal*]

/*θέσε στον δείκτη *SubListPtr* την τιμή του δείκτη που βρίσκεται στην θέση *HVal* του πίνακα *HashTable,* δηλαδή ο δείκτης *SubListPtr* δείχνει στην υπολίστα των συνωνύμων όπου θα αναζητηθεί το κλειδί *KeyArg**/

Εφάρμοσε τον αλγόριθμο αναζήτησης κλειδιού σε μια υπολίστα συνωνύμων /*θα πραγματοποιηθεί αναζήτηση του κλειδιού *KeyArg* στην υπολίστα συνωνύμων που δείχνει ο δείκτης *SubListPtr.* Ο αλγόριθμος θα επιστρέψει τη θέση *Loc* της εγγραφής με κλειδί *Keyarg* και τη θέση *Pred* της αμέσως προηγούμενης εγγραφής. Αν δεν υπάρχει εγγραφή με κλειδί *Keyarg* οι μεταβλητές *Loc* και *Pred* θα έχουν την τιμή -1*/

Τέλος _αν

Ο αλγόριθμος που ακολουθεί καλείται από τον προηγούμενο αλγόριθμο για να εντοπιστεί η εγγραφή (αν υπάρχει) μέσα στην υπολίστα συνωνύμων:

ΑΛΓΟΡΙΘΜΟΣ ΑΝΑΖΗΤΗΣΗΣ ΤΙΜΗΣ ΣΕ ΜΙΑ ΥΠΟΛΙΣΤΑ ΣΥΝΩΝΥΜΩΝ

/*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.*/

Next ← *SubListPtr* /*θέσε στον δείκτη *Next* την τιμή του δείκτη *SubListPtr*, δηλαδή ο *Next* δείχνει στην 1η εγγραφή της υπολίστας συνωνύμων όπου θα αναζητηθεί το κλειδί *KeyArg**/

Loc ← -1 /*θέσε στη μεταβλητή *Loc* την τιμή -1. Αν στη συνέχεια βρεθεί εγγραφή με κλειδί *KeyArg*, η μεταβλητή *Loc* θα πάρει ως τιμή την θέση αυτής της εγγραφής στην υπολίστα συνωνύμων*/

Pred ← -1 /*θέσε στη μεταβλητή *PredPtr* την τιμή -1. Αν στη συνέχεια βρεθεί εγγραφή με κλειδί *KeyArg*, η μεταβλητή *PredPtr* θα πάρει ως τιμή την θέση της προηγούμενης εγγραφής στην υπολίστα συνωνύμων*/

Όσο *Next* != *EndOfList* **επανάλαβε**

/*όσο η τιμή της μεταβλητής *Next* είναι διάφορη της τιμής της μεταβλητής-σημαίας *EndOfList,* δηλαδή όσο δεν έχει βρεθεί εγγραφή με κλειδί *KeyArg* και δεν έχουν ελεγχθεί όλες οι εγγραφές της υπολίστας των συνωνύμων*/

Αν *List*[*Next*].*RecKey* = *KeyArg* **τότε**

/*αν η τιμή του κλειδιού *RecKey* της εγγραφής που βρίσκεται στη θέση *Next* της λίστας *List* ισούται με την τιμή *KeyArg**/

Loc ← *Next*

/*θέσε στη μεταβλητή *Loc* την τιμή της *Next,* δηλαδή η *Loc* παίρνει ως τιμή τη θέση της εγγραφής της υπολίστας συνωνύμων που έχει κλειδί *KeyArg**/

Next ← *EndOfList*

/*θέσε στη μεταβλητή *Next* την τιμή της μεταβλητής-σημαίας *EndOfList*, έτσι ώστε στην επόμενη επανάληψη να τερματίσει ο βρόχος*/

Αλλιώς

Pred ← *Next*

/*θέσε στη μεταβλητή *Pred* την τιμή της μεταβλητής *Next*, δηλαδή η μεταβλητή *Pred* δείχνει στην τρέχουσα εγγραφή της υπολίστας συνωνύμων, της οποίας η τιμή του κλειδιού ελέγχθηκε και βρέθηκε ότι είναι διάφορη της *KeyArg**/

Next ← *List*[*Next*].*Link*

/*θέσε στη μεταβλητή *Next* την τιμή του πεδίου *Link* της εγγραφής που βρίσκεται στη θέση *Next* της λίστας *List*, δηλαδή η *Next* δείχνει στην αμέσως επόμενη εγγραφή της υπολίστας των συνωνύμων που εξετάζεται*/

Τέλος_ αν

Τέλος_ επανάληψης

Εφόσον δεν βρεθεί άλλη εγγραφή με το ίδιο κλειδί, ο αλγόριθμος που ακολουθεί εισάγει την εγγραφή που επιθυμούμε στο τέλος της κατάλληλης υπολίστας συνωνύμων:

ΑΛΓΟΡΙΘΜΟΣ ΕΙΣΑΓΩΓΗΣ ΕΓΓΡΑΦΗΣ ΣΕ ΜΙΑ ΥΠΟΛΙΣΤΑ ΣΥΝΩΝΥΜΩΝ

/*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.*/

Αν η λίστα *List* δεν είναι γεμάτη **τότε**

Loc ← -1

/*θέσε στη μεταβλητή *Loc* την τιμή -1. Αν στη συνέχεια βρεθεί εγγραφή με κλειδί *KeyArg*, η μεταβλητή *Loc* θα πάρει ως τιμή την θέση αυτής της εγγραφής στην υπολίστα συνωνύμων*/

Pred ← -1

/*θέσε στη μεταβλητή *PredPtr* την τιμή -1. Αν στη συνέχεια βρεθεί εγγραφή με κλειδί *KeyArg*, η μεταβλητή *PredPtr* θα πάρει ως τιμή την θέση της προηγούμενης εγγραφής στην υπολίστα συνωνύμων*/

Εφάρμοσε τον αλγόριθμο αναζήτησης τιμής στη δομή *HList*

/*ο αλγόριθμος, ο οποίος παρουσιάστηκε παραπάνω,

δέχεται τη δομή *HList* και την τιμή του κλειδιού *RecKey* της εγγραφής *InRec* και επιστρέφει τη θέση *Loc* της εγγραφής που έχει ως κλειδί την προαναφερθείσα τιμή και τη θέση *Pred* της προηγούμενης εγγραφής. Αν δεν βρεθεί εγγραφή με αυτή την τιμή ως κλειδί τότε οι *Loc* και *Pred* έχουν την τιμή -1*/

Αν *Loc* = -1 **τότε** /*αν η μεταβλητή *Loc* έχει τιμή -1, δηλαδή αν δεν βρέθηκε εγγραφή με το ίδιο κλειδί*/

Size ← *Size* + 1 /*αύξησε το πλήθος *Size* των εγγραφών της λίστας *List* κατά 1*/

New ← *StackPtr* /*θέσε στον δείκτη *New* την τιμή του δείκτη *StackPtr* ώστε να δείχνει στην 1η ελεύθερη θέση της λίστας *List**/

StackPtr ← *List*[*New*].*Link*

/*θέσε στον δείκτη *StackPtr* την τιμή του πεδίου *Link* της εγγραφής που βρίσκεται στη θέση *New* της λίστας *List,* δηλαδή ο δείκτης *StackPtr* ενημερώνεται ώστε να εξακολουθήσει να δείχνει στην 1η ελεύθερη θέση της λίστας *List**/

List[*New*] ← *InRec* /*η εγγραφή *InRec* εισάγεται στη θέση *New* της λίστας *List**/

Αν *Pred* = -1 **τότε** /*αν η μεταβλητή *Pred* έχει τιμή -1, δηλαδή αν δεν υπάρχει καμία εγγραφή στην υπολίστα συνωνύμων*/

/*η εγγραφή *InRec* θα εισαχθεί στην αρχή της υπολίστας συνωνύμων, αφού δεν υπάρχει άλλη εγγραφή σε αυτή*/

Υπολόγισε τη τιμή κατακερματισμού *HVal* για το κλειδί *RecKey* της εγγραφής *InRec*

HashTable[*HVal*] ← *New*

/*ενημέρωσε τον δείκτη που βρίσκεται στην *HVal* θέση του πίνακα *HashTable* ώστε να δείχνει στην 1η εγγραφή της νέας υπολίστας συνωνύμων, η οποία αντιστοιχεί στη θέση *New* της λίστας *List**/

List[*New*].*Link* ← *EndOfList*

/*θέσε στο πεδίο δείκτη *Link* της εγγραφής που εισήχθη στη θέση *New* της λίστας *List* την τιμή της μεταβλητής-σημαίας *EndOfList,* αφού η συγκεκριμένη εγγραφή είναι η τελευταία της υπολίστας συνωνύμων*/

Αλλιώς /*υπάρχει υπολίστα συνωνύμων*/

/*η εγγραφή *InRec* θα εισαχθεί στο τέλος της υπολίστας των συνωνύμων που ανήκει*/

List[*New*].*Link* ← *List*[*Pred*].*Link*

/*θέσε στο πεδίο *Link* της εγγραφής που εισήχθη στη θέση *New* της λίστας *List* την τιμή του πεδίου *Link* της εγγραφής που βρίσκεται στη θέση *Pred,* δηλαδή ο δείκτης της νέας εγγραφής που είναι η τελευταία της υπολίστας συνωνύμων θα πάρει την τιμή του δείκτη της εγγραφής που ήταν μέχρι τώρα τελευταία της υπολίστας συνωνύμων*/

List[*Pred*].*Link* ← *New*

/*θέσε στο πεδίο *Link* της εγγραφής που εισήχθη στη θέση *Pred* της λίστας *List* την τιμή του δείκτη *New,* δηλαδή η εγγραφή που ήταν πριν την εισαγωγή της νέας εγγραφής τελευταία στην υπολίστα συνωνύμων θα έχει ως επόμενη τη νέα εγγραφή*/

Τέλος_ αν

Αλλιώς

Γράψε 'Υπάρχει ήδη εγγραφή με το ίδιο κλειδί.'

Τέλος_ αν

Αλλιώς

Γράψε 'Η λίστα είναι γεμάτη'

Τέλος_ αν

Στη συνέχεια παρουσιάζονται οι διαδικασίες SearchHashList, SearchSynonymList και AddRec που υλοποιούν τους παραπάνω αλγόριθμους:

```
void SearchHashList(HashListType HList, int 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)
      {
        *Pred = -1;
        *Loc = -1;
      }
    else
     {
       HList.SubListPtr=HList.HashTable[HVal];
       SearchSynonymList(HList,KeyArg,Loc,Pred);
     }
}
```

Η διαδικασία SearchSynonymList καλείται από την SearchHashList για να εντοπιστεί η εγγραφή (αν υπάρχει) μέσα στην υπολίστα συνωνύμων:

```
void SearchSynonymList(HashListType HList,int 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;
    Next=HList.SubListPtr;
    *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;
          }
      }
}
```

Η διαδικασία AddRec που εισάγει την εγγραφή που επιθυμούμε στο τέλος της κατάλληλης υπολίστας συνωνύμων είναι η εξής:

```
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);
         if (Loc == -1)
         /*Δε βρέθηκε εγγραφή με το ίδιο κλειδί*/
            {
               HList->Size = HList->Size+1;
               New = HList->StackPtr;
               HList->StackPtr = HList->List[New].Link;
               HList->List[New] = InRec;
               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("Υπάρχει ήδη εγγραφή με το ίδιο κλειδί.\n");
      }
   else
     printf("Η λίστα είναι γεμάτη\n");
}
```

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

HValue=Key % HMax

Ο πίνακας HashTable θα έχει HMax=5 θέσεις, σε καθεμιά από τις οποίες καταχωρούμε αρχικά την τιμή -1, που λειτουργεί ως μηδενικός δείκτης. Το μέγεθος του πίνακα List θεωρούμε ότι είναι 15, και ο πίνακας List θα είναι ως εξής:

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

Δηλαδή, η εγγραφή που θα αποθηκευτεί στην θέση μηδέν του πίνακα List θα έχει ως επόμενη αυτήν που θα αποθηκευτεί στην θέση ένα (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 στην θέση μηδέν του πίνακα List, το δεύτερο στοιχείο θα αποθηκευτεί στην θέση ένα του πίνακα List, κ.ο.κ. και η αρχή της συνδεδεμένης λίστας αποθηκεύεται στη μεταβλητή StackPtr. Η συνδεδεμένη αυτή λίστα λειτουργεί ως στοίβα.

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 στην 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:

HashTable

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

List

	key	Data	Link
0	23		1 -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

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 υπολίστα νούμερο 0 και στην θέση HValue=0 του πίνακ

HashTable

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

List

	key	Data	Link
0	23		1 -1
1	40		2 -1
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

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 στην θέση δύο της λίστας List:

HashTable

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

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

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 στην θέση τρία 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. Η σύνδεση αυτή γίνεται αλλάζοντας την τιμή του πεδίου Link της 3ης εγγραφής (στην θέση 2 του πίνακα) σε 3 ώστε να δείχνει στην τέταρτη εγγραφή (στην θέση τέσσερα):

HashTable

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

List

	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

Όπως φαίνεται από το παραπάνω σχήμα, η τιμή HashTable[3] δεν αλλάζει, γιατί δείχνει στην αρχή της υπολίστας 3, που είναι η 3η εγγραφή (στην θέση δύο του πίνακα List).

Εν συνεχεία, έστω ότι θέλουμε να εισαγάγουμε μια εγγραφή με τιμή κλειδιού 12. Επειδή HValue=12 % 5 = 2 και η εγγραφή θα τοποθετηθεί στην τέταρτη θέση της λίστας List, στην θέση 2 του πίνακα καταχωρούμε την τιμή 4, όπως δείχνει και το παρακάτω σχήμα:

HashTable

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

List

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

Έστω ότι ακολουθεί νέα εισαγωγή με τιμή κλειδιού 45. Η εγγραφή αυτή θα τοποθετηθεί στην πέμπτη θέση της λίστας List και θα ανήκει στην 0-κη 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

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

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 έκτη θέση της λίστας List:

HashTable

0	-1 1
1	-1 2
2	-1 4
3	-1 0
4	-1 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	39		7 -1
7			8
8			9
9			10
10			11
11			12
12			13
13			14
14			-1

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 έβδομη θέση 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
8			9
9			10
10			11
11			12
12			13
13			14
14			-1

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 όγδοη θέση της λίστας List (40, 45, 30 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 8
6	39		7 -1
7	68		8 -1
8	30		9 -1
9			10
10			11
11			12
12			13
13			14
14			-1

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

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

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
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

Εισαγωγή εγγραφής με κλειδί 3

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

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 πρώτη περίπτωση χρειάζεται να αλλάξουμε την τιμή του πεδίου Link της προηγούμενης εγγραφής ώστε να δείχνει στην επόμενη αυτής που θα διαγραφεί, ενώ στη δεύτερη περίπτωση βρίσκουμε την τιμή κατακερματισμού HValue και αλλάζουμε την

καταχώρηση HashTable[HValue] ώστε να δείχνει στην θέση της δεύτερης εγγραφής της αντίστοιχης υπολίστας. 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. Ο αλγόριθμος της διαγραφής είναι ο εξής:

ΑΛΓΟΡΙΘΜΟΣ ΔΙΑΓΡΑΦΗΣ ΕΓΓΡΑΦΗΣ

/*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.*/

Εφάρμοσε τον αλγόριθμο αναζήτησης τιμής στη δομή *HList* /*ο αλγόριθμος, ο οποίος παρουσιάστηκε παραπάνω, δέχεται τη δομή *HList* και την τιμή του κλειδιού *DelKey* της εγγραφής που πρόκειται να διαγραφεί και επιστρέφει τη θέση *Loc* της εγγραφής που έχει ως κλειδί την προαναφερθείσα τιμή και τη θέση *Pred* της προηγούμενης εγγραφής. Αν δεν βρεθεί εγγραφή με αυτή την τιμή ως κλειδί τότε οι *Loc* και *Pred* έχουν την τιμή -1*/

Αν *Loc* != -1 **τότε** /*αν η μεταβλητή *Loc* έχει τιμή διάφορη του -1, δηλαδή αν υπάρχει στη λίστα εγγραφή με κλειδί *DelKey*/

Αν *Pred* != -1 **τότε** /*αν η μεταβλητή *Pred* έχει τιμή διάφορη του -1, δηλαδή αν η εγγραφή που θα διαγραφεί έχει προηγούμενη*/

List[*Pred*].*Link* ← *List*[*Loc*].*Link*

/*θέσε στο πεδίο *Link* της εγγραφής που βρίσκεται στη θέση *Pred* της λίστας *List* την τιμή του πεδίου *Link* της εγγραφής που βρίσκεται στη θέση *Loc* της λίστας *List*, δηλαδή η εγγραφή που είναι προηγούμενη της εγγραφής που διαγράφεται θα δείχνει στην επόμενη της εγγραφής που διαγράφεται*/

Αλλιώς /*η εγγραφή που θα διαγραφεί δεν έχει προηγούμενη ή αλλιώς είναι η 1η της υπολίστας των συνωνύμων που ανήκει*/

Υπολόγισε τη τιμή κατακερματισμού *HVal* για το κλειδί *DelKey*

HashTable[*HVal*] ← *List*[*Loc*].*Link*

/*θέσε στον δείκτη που βρίσκεται στη θέση *HVal* του πίνακα *HashTable* την τιμή του πεδίου *Link* της εγγραφής που βρίσκεται στη θέση *Loc* της λίστας *List*, δηλαδή ο δείκτης του πίνακα που δείχνει στην υπολίστα συνωνύμων της οποίας διαγράφηκε η 1η εγγραφή δείχνει στην εγγραφή που μέχρι

τώρα ήταν 2η*/

Τέλος_ αν

List[*Loc*].*Link* ← *StackPtr*

/*θέσε στο πεδίο *Link* της εγγραφής που βρίσκεται στη θέση *Loc* της λίστας *List* την τιμή του δείκτη *StackPtr*, δηλαδή η εγγραφή που διαγράφηκε δείχνει στην - μέχρι τώρα - 1η ελεύθερη θέση της λίστας *List**/

StackPtr ← *Loc*

/*θέσε στο δείκτη *StackPtr* την τιμή της *Loc*, δηλαδή ο δείκτης *StackPtr* δείχνει στη θέση της διαγραμμένης εγγραφής που είναι πλέον η 1η διαθέσιμη θέση της λίστας *List**/

Size ← *Size* - 1

/*μείωσε το πλήθος *Size* των εγγραφών της λίστας *List* κατά 1*/

Αλλιώς

Γράψε 'Δεν υπάρχει εγγραφή με κλειδί', *DelKey*

Τέλος_ αν

Σε μορφή κώδικα η λειτουργία της διαγραφής δίνεται από τη διαδικασία DeleteRec:

```
void DeleteRec(HashListType *HList, int 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.*/
{
    int Loc, Pred, 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 = HList->Size-1;
     }
   else
      printf("Δεν υπάρχει εγγραφή με κλειδί %d\n",DelKey);
}
```

Έστω, για παράδειγμα, 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, θέτουμε το πεδίο Link της εγγραφής με κλειδί 23, δηλαδή της προηγούμενης εγγραφής του 68 στη λίστα συνωνύμων, ίσο με τη θέση της επόμενης εγγραφής του 68 στη λίστα συνωνύμων, δηλαδή 10. Ο δείκτης StackPtr έχει τιμή 12, επομένως στο πεδίο Link της εγγραφής με κλειδί 68 θέτουμε την τιμή 12, ενώ ο δείκτης StackPtr παίρνει τώρα τιμή 7:

HashTable

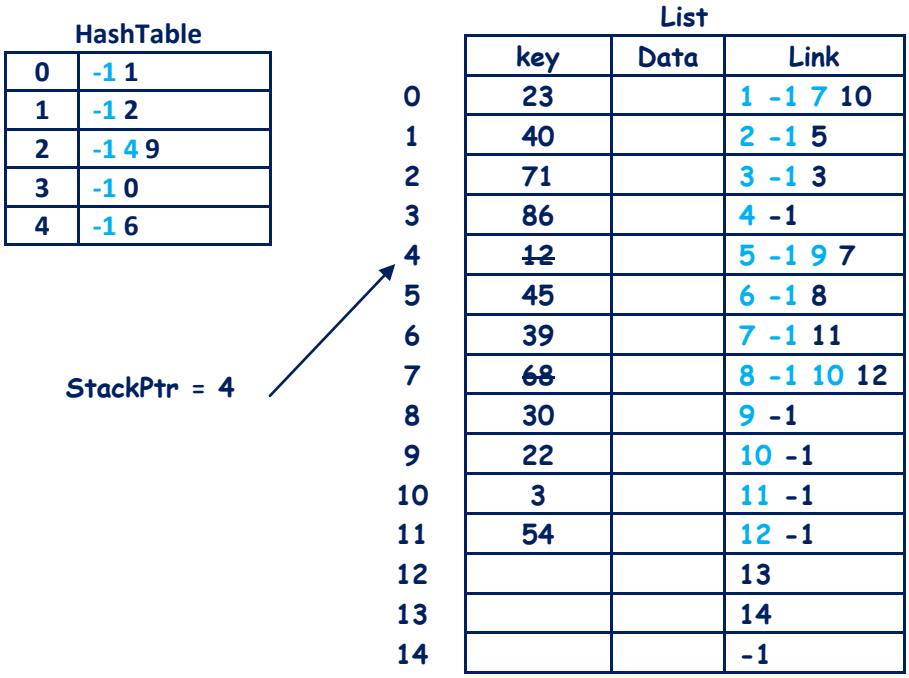
0	-1 1
1	-1 2
2	-1 4
3	-1 0
4	-1 6

StackPtr = 7 →

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
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

Αν επιθυμούμε να διαγράψουμε την εγγραφή με τιμή κλειδιού 12 (12 % 5 = 2), δηλαδή την πρώτη εγγραφή της 2ης υπολίστας συνωνύμων, τότε χρειάζεται να θέσουμε την τιμή 9 στην θέση 3 του πίνακα HashTable, γιατί η πρώτη εγγραφή της τρίτης υπολίστας θα είναι τώρα η εγγραφή με τιμή κλειδιού 22, που βρίσκεται στη θέση 9 της λίστας List. Στο πεδίο Link της διαγραμμένης εγγραφής θέτουμε την τιμή StackPtr=7 και ο δείκτης StackPtr παίρνει τιμή 4:



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.

Η διασύνδεση HashList.h και η υλοποίηση της HashList.c υλοποιούν την τεχνική του κατακερματισμού με την αλυσιδωτή σύνδεση, που περιγράψαμε μέχρι τώρα, μαζί με τις βασικές λειτουργίες, που συνδέονται με αυτήν, δίνεται παρακάτω. Η εντολή

#include "HashList.h";

μπορεί να ενσωματωθεί σε ένα πρόγραμμα C για να χρησιμοποιηθεί η δομή του κατακερματισμού.

```
//filename HashList.h

 #define HMax 5                    /*όριο μεγέθους του πίνακα HashTable*/
```

```
 #define VMax 30                    /*όριο μεγέθους της λίστας*/
 #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 struct   {
                    int RecKey;
                    ListElementType Data;
                    /*τα υπόλοιπα πεδία*/
                    int Link;
                  } ListElm;
 typedef struct   {
                     int HashTable[HMax]; /*πίνακας δεικτών προς τις υπολίστες
                     συνωνύμων*/
                     int Size; /*πλήθος εγγραφών της λίστας List*/
                     int  SubListPtr; /*δείκτης σε μια υπολίστα*/
                     int StackPtr; /*δείκτης προς την πρώτη ελεύθερη θέση της
                     λίστας List*/
                     ListElm List[VMax];
                  } HashListType;
   typedef enum {
                FALSE, TRUE
   } boolean;
   void CreateHashList(HashListType *HList);
   int HashKey(int Key);
   boolean FullHashList(HashListType HList);
   void SearchSynonymList(HashListType HList, int KeyArg, int *Loc, int *Pred);
   void SearchHashList(HashListType HList, int KeyArg,int *Loc, int *Pred);
   void AddRec(HashListType *HList,ListElm InRec);
   void DeleteRec(HashListType *HList, int DelKey);

//filename HashList.c

#include <stdio.h>

#include "HashList.h"

int  HashKey(int Key)
/*Δέχεται:          Την τιμή Key ενός κλειδιού.
  Λειτουργία:       Βρίσκει την τιμή κατακερματισμού HValue για το κλειδί Key.
  Επιστρέφει:       Την τιμή κατακερματισμού HValue.*/
{
        return Key % HMax;
}

void CreateHashList(HashListType *HList)
/*Λειτουργία:             Δημιουργεί μια δομή HList.
  Επιστρέφει:             Την δομή HList.*/
{
    int index;
```

```
    HList->Size = 0;
    HList->StackPtr = 0;

    /*Δημιουργία της στοίβας των ελεύθερων θέσεων στη λίστα List*/
    index = 0;
    while(index < VMax - 1)
    {
        HList->List[index].Link = index+1;
        HList->List[index].Data = 0;
        index = index+1;
    }
    HList->List[index].Link = EndOfList;

    /*Αρχικοποίηση του πίνακα HashTable*/
    index = 0;
    while (index < HMax)
    {
        HList->HashTable[index] = EndOfList;
        index = index+1;
    }

}

 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);
}

void SearchSynonymList(HashListType HList,int 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;
    Next=HList.SubListPtr;
    *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;
            }
        }
}

void SearchHashList(HashListType HList, int 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)
     {
       *Pred = -1;
       *Loc = -1;
     }
    else
     {
       HList.SubListPtr=HList.HashTable[HVal];
       SearchSynonymList(HList,KeyArg,Loc,Pred);
     }
}

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, &Loc, &Pred);
         if (Loc == -1)
         /*Δε βρέθηκε εγγραφή με το ίδιο κλειδί*/
            {
```

```
                HList->Size = HList->Size+1;
                New = HList->StackPtr;
                HList->StackPtr = HList->List[New].Link;
                HList->List[New] = InRec;
                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("Υπάρχει ήδη εγγραφή με το ίδιο κλειδί.\n");
       }
    else
      printf("Η λίστα είναι γεμάτη\n");
}

void DeleteRec(HashListType *HList, int 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.*/
{
    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 = HList->Size-1;
  }
 else
    printf("Δεν υπάρχει εγγραφή με κλειδί %d\n",DelKey);
}
```

Το πρόγραμμα-πελάτης Chaining.c χρησιμοποιεί τη διασύνδεση HashList.h και την αντίστοιχη υλοποίηση της HashList.c και δείχνει τη χρήση όλων των λειτουργιών του ΑΔΤ κατακερματισμού με αλυσιδωτή σύνδεση για τη διαχείριση των συνωνύμων.

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.

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, 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			

ΑΝΟΙΧΤΗ ΔΙΕΥΘΥΝΣΙΟΔΟΤΗΣΗ (OPEN ADDRESSING)

Στην τεχνική της ανοιχτής διευθυνσιοδότησης, 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.

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.

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.

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.