Κατακερματισμός

Καθηγητής Π. Λουρίδας

Τμήμα Διοικητικής Επιστήμης και Τεχνολογίας,
Οικονομικό Πανεπιστήμιο Αθηνών
louridas@aueb.gr

Επισκόπηση

Γκαρνταρόμπα

- Όταν παραδίδουμε ένα αντικείμενο σε μία γκαρνταρόμπα, παίρνουμε μια απόδειξη που αντιστοιχεί στο αντικείμενο που παραδώσαμε.
- Όταν θέλουμε να πάρουμε πίσω το αντικείμενο, δίνουμε στον υπεύθυνο την απόδειξη και ο υπεύθυνος μας επιστρέφει το αντικείμενό μας.
- Αν το σκεφτούμε, αυτό είναι μια *μέθοδος εύρεσης χωρίς αναζήτηση*.

Εύρεση Χωρίς Αναζήτηση

- Όταν θέλουμε να αποθηκεύσουμε ένα αντικείμενο, απλώς υπολογίζουμε τη διεύθυνση στην οποία θα το αποθηκεύσουμε.
- Όταν θέλουμε να αναζητήσουμε ένα αντικείμενο, υπολογίζουμε πάλι τη διεύθυνσή του και το ανακτούμε κατ' ευθείαν από εκεί.

Υλοποίηση Εύρεσης Χωρίς Αναζήτηση

- Θέλουμε να αποθηκεύσουμε και να εντοπίζουμε κάποια συγκεκριμένη εγγραφή.
- Η αποθήκευση και ο εντοπισμός θα γίνονται με βάση το κλειδί της.
- Θέλουμε, έχοντας το κλειδί της εγγραφής, να υπολογίζουμε τη διεύθυνση στην οποία θα αποθηκεύεται την εγγραφή.

Υλοποίηση Εύρεσης Χωρίς Αναζήτηση

- Η διεύθυνση της εγγραφής θα είναι μια διεύθυνση στον υπολογιστή.
- Η διεύθυνση θα είναι ένας αριθμός.
- Χρειαζόμαστε μια συνάρτηση που μετατρέπει κλειδιά σε τέτοιες διευθύνσεις.

Υλοποίηση Εύρεσης Χωρίς Αναζήτηση

- Χρειαζόμαστε μια συνάρτηση, έστω $f(K)$, η οποία παίρνει το κλειδί K μιας εγγραφής R και επιστρέφει μια τιμή a.
- Η τιμή a θα είναι η διεύθυνση που θα αποθηκευτεί η R.
- Κάθε φορά που θέλουμε να ανακτήσουμε την εγγραφή R με βάση το κλειδί της, υπολογίζουμε πάλι την $f(K)$, η οποία θα μας δώσει την ίδια διεύθυνση a, όπου θα βρούμε την R.

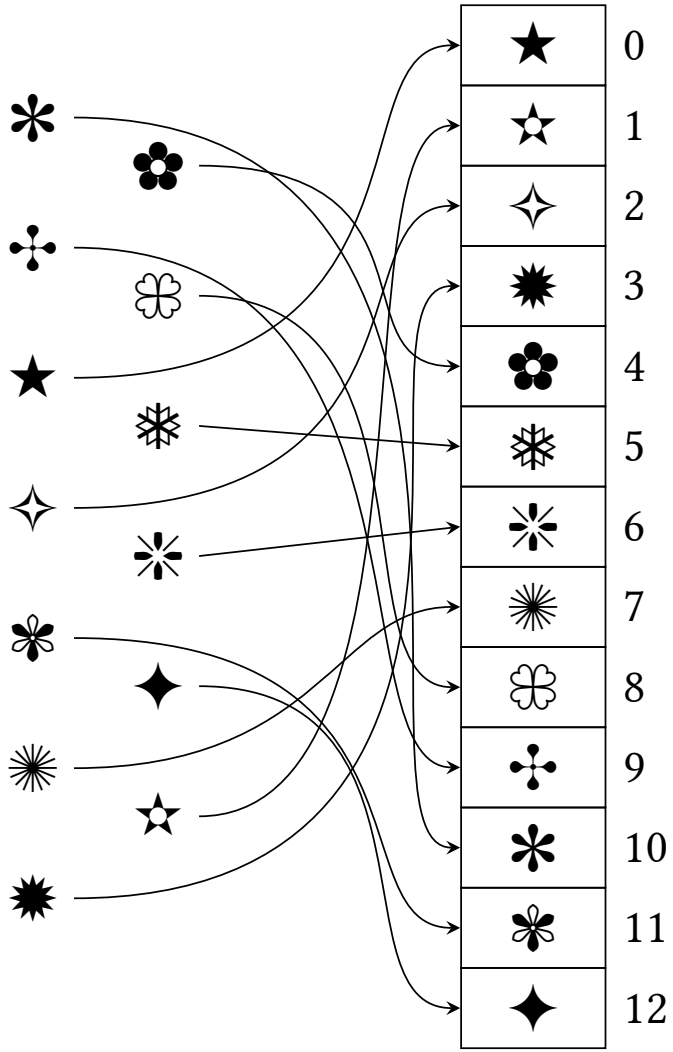
Προϋπόθεση

- Για να λειτουργήσει η συγκεκριμένη ιδέα θα πρέπει η συνάρτηση $f(K)$ να είναι γρήγορη.
- Αν είναι γρήγορη, τότε η συνάρτηση θα αναλαμβάνει τον εντοπισμό.
- Αν είναι αργή, δεν αξίζει τον κόπο.

Λειτουργία Συνάρτησης

- Έστω ότι έχουμε n διαφορετικές εγγραφές, τις οποίες γνωρίζουμε εκ των προτέρων.
- Τότε το πρόβλημα ανάγεται στην εύρεση μιας συνάρτησης $f(K)$ η οποία να παράγει μια διαφορετική τιμή από το 0 μέχρι το $n-1$ για κάθε διαφορετικό κλειδί.
- Τότε μπορούμε να αποθηκεύσουμε κάθε εγγραφή R σε έναν πίνακα T μεγέθους n, έτσι ώστε αν $f(K) = a$, όπου K το κλειδί του R, τότε $T[a] = R$.

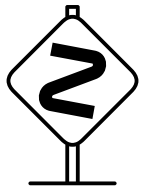
Παράδειγμα



Δημιουργία Συνάρτησης $f(K)$

- Η δημιουργία μιας συνάρτησης $f(K)$ για ένα σύνολο εγγραφών δεν είναι εύκολη υπόθεση.
- Μπορούμε να φτιάξουμε μια συνάρτηση f «με το χέρι», αλλά είναι περίπλοκο και κουραστικό.
- Συγκεκριμένα, μπορούμε να φτιάξουμε, με επιμονή, μια συνάρτηση η οποία αντιστοιχεί κάθε εγγραφή σε μία διαφορετική διεύθυνση.
- Μια τέτοια συνάρτηση είναι μια *τέλεια απεικόνιση* (perfect mapping), αφού αντιστοιχεί κάθε είσοδο σε μία από τις διαθέσιμες διαφορετικές τιμές.

Προσοχή



Η χειροποίητη συνάρτηση που ακολουθεί είναι απλώς παράδειγμα, όχι αντικείμενο μελέτης.

Παράδειγμα Τέλειας Απεικόνισης

Algorithm: A perfect mapping of 31 of most common words in English to numbers.

`PerfectMapping`$(s) \to r$

Input: s, a string among a predefined list of 31 of the most common words in English

Output: r, an address in the range $-10, 1, \ldots, 29$

```
1  r ← −Code(s[0])
2  s ← Code(s[1])
3  r ← r − 8 + s
4  if r ≤ 0 then
5      r ← r + 16 + s
6  s ←Code(s[2])
7  if s = 0 then
8      return r
9  r ← r − 28 + s
10 if r > 0 then
11     return r
12 r ← r + 11 + s
13 t ← Code(s[3])
14 if t = 0 then
15     return r
16 r ← r − (s − 5)
17 if r < 0 then
18     return r
19 r ← r + 10
20 return r
```

Επεξήγηση

- Η συνάρτηση Code παίρνει έναν χαρακτήρα και επιστρέφει μια αριθμητική τιμή. Ο κενός χαρακτήρας παίρνει την τιμή μηδέν, το «Α» παίρνει την τιμή ένα, το «Β» την τιμή δύο, κ.λπ.
- Σε αυτό προστίθεται ένα αν η τιμή του χαρακτήρα είναι μεγαλύτερη από το εννέα και άλλα δύο αν η τιμή που προκύπτει είναι μεγαλύτερη από το 19.
- Στη συνέχεια γίνονται διάφορες πράξεις στους τέσσερεις πρώτους χαρακτήρες της λέξης (αν έχει λιγότερους χαρακτήρες, θεωρούμε ότι στο τέλος έχει κενά).

Αποτέλεσμα Απεικόνισης

A	7	FOR	23	IN	29	THE	-6
AND	-3	FROM	19	IS	5	THIS	-2
ARE	3	HAD	-7	IT	6	TO	17
AS	13	HAVE	25	NOT	20	WAS	11
AT	14	HE	10	OF	4	WHICH	-5
BE	16	HER	1	ON	22	WITH	21
BUT	9	HIS	12	OR	30	YOU	8
BY	18	I	-1	THAT	-10		

Προβλήματα Απεικόνισης

- Η συνάρτηση `PerfectMapping` δουλεύει και είναι πολύ γρήγορη. Αρκεί στο αποτέλεσμα να προσθέσουμε 10 για να πάρουμε μια θέση σε έναν πίνακα από το $0, 1, \ldots, 39$.
- Είναι όμως εξαιρετικά δυσνόητη.
- Δεν μπορούμε να ψάχνουμε τέτοιες συναρτήσεις για κάθε σύνολο εγγραφών που έχουμε να αποθηκεύσουμε.
- Ακόμα, αν αλλάξει έστω και ένα κλειδί είναι πολύ πιθανό η συνάρτηση να μη δουλεύει πλέον γιατί μπορεί δύο κλειδιά να αντιστοιχούν στην ίδια αριθμητική τιμή.
- Τέλος, στο διάστημα $-10, -9, \ldots, 29$ υπάρχουν εννέα θέσεις που δεν χρησιμοποιούνται—άρα ξοδεύουμε χώρο.

Προς μια άλλη Συνάρτηση

- Μια άλλη ιδέα είναι να πάρουμε τον πρώτο χαρακτήρα κάθε κλειδιού, τον τελευταίο χαρακτήρα κάθε κλειδιού, και τον αριθμό των χαρακτήρων κάθε κλειδιού.
- Στη συνέχεια θα μπορούσαμε να υπολογίσουμε μια αριθμητική τιμή ως εξής:

$$h = \mathsf{Code}(b) + \mathsf{Code}(e) + |K|$$

όπου K είναι το κλειδί, $|K|$ είναι το μήκος του κλειδιού, b είναι ο χαρακτήρας στην αρχή του κλειδιού και e είναι ο χαρακτήρας στο τέλος του κλειδιού.

- Η τιμή h θα είναι η διεύθυνση που θα χρησιμοποιήσουμε στον πίνακα για να αποθηκεύσουμε την εγγραφή.
- Το πρόβλημα τότε ανάγεται στην εύρεση των σωστών κωδικών, της σωστής συνάρτησης Code.

Αλγόριθμος Ελάχιστης Τέλειας Απεικόνισης

Algorithm: A minimal perfect mapping.

`MinimalPerfectMapping`$(s) \longrightarrow r$

Input: s, a string among a predefined list of 31 of the most common words in English

Output: r, an address in the range $1, \ldots, 32$

Data: C, an array of 26 integers

```
1  C ← [
2      3,  23,  -1,  17,   7,  11,  -1,   5,   0,
3     -1,  -1,  -1,  16,  17,   9,  -1,  -1,  13,
4      4,   0,  23,  -1,   8,  -1,   4,  -1
5  ]
6  l ← |s|
7  b ←Ordinal(s[0])
8  e ←Ordinal(s[l − 1])
9  r ← l + C[b] + C[e]
10 return r
```

Επεξήγηση

- Η συνάρτηση `Code` είναι στην ουσία ο πίνακας C στον αλγόριθμο.
- Κάθε θέση του πίνακα C αντιστοιχεί στον αριθμό που θα προσδώσουμε στον αντίστοιχο χαρακτήρα του αλφαβήτου.
- Η θέση μηδέν του πίνακα περιέχει τον αριθμό για το «Α», η θέση ένα περιέχει τον αριθμό για το «Β», κ.λπ.
- Η συνάρτηση `Ordinal(`ch`)` επιστρέφει τη θέση του χαρακτήρα ch στο αλφάβητο (μηδέν για το «Α», ένα για το «Β», κ.λπ.).
- Ο πίνακας C έχει κάποιες θέσεις ίσες με -1. Αυτές αντιστοιχούν στα γράμματα του αλφαβήτου που δεν εμφανίζονται ούτε ως πρώτος ούτε ως τελευταίος χαρακτήρας σε κανένα από τα κλειδιά μας (π.χ. ο χαρακτήρας «C»).

Αποτέλεσμα Απεικόνισης

A	7	FOR	27	IN	19	THE	10
AND	23	FROM	31	IS	6	THIS	8
ARE	13	HAD	25	IT	2	TO	11
AS	9	HAVE	16	NOT	20	WAS	15
AT	5	HE	14	OF	22	WHICH	18
BE	32	HER	21	ON	28	WITH	17
BUT	26	HIS	12	OR	24	YOU	30
BY	29	I	1	THAT	4		

Χαρακτηριστικά Αλγορίθμου

- Είναι μια καλύτερη λύση από την προηγούμενη. Ο αλγόριθμος είναι απλούστερος και δεν σπαταλάμε χώρο.
- Είναι ένα παράδειγμα *ελάχιστης τέλειας απεικόνισης* (minimal perfect mapping), αφού αντιστοιχεί όλες τις λέξεις σε διαφορετικές τιμές, και αντιστοιχεί n λέξεις σε n τιμές, τον ελάχιστο αριθμό.
- Είναι όμως μια γενική λύση στο πρόβλημά μας;
- Όχι—απαιτεί γνώση όλων των κλειδιών εκ των προτέρων. Δεν είναι βέβαιο ότι δουλεύει για όλα τα πιθανά κλειδιά (π.χ. τα κλειδιά «ERA» και «ARE» θα είχαν την ίδια απεικόνιση). Η δημιουργία ενός πίνακα όπως ο C είναι από μόνη της ένα πρόβλημα που απαιτεί εξειδικευμένους αλγορίθμους.

Επισκόπηση

Το Πρόβλημα

- Θέλουμε μια συνάρτηση γενικής χρήσης η οποία να αντιστοιχεί κλειδιά σε τιμές μέσα σε ένα συγκεκριμένο εύρος.
- Η συνάρτηση θα πρέπει να το κάνει αυτό χωρίς να γνωρίζει εκ των προτέρων τα δυνατά κλειδιά.
- Η συνάρτηση θα πρέπει να αποφεύγει «όσο είναι δυνατόν» την απεικόνιση διαφορετικών κλειδιών στην ίδια αριθμητική τιμή.
- Λέμε «όσο είναι δυνατόν» γιατί αν έχουμε έναν πίνακα μεγέθους n και $2n$ δυνατά κλειδιά, τότε τουλάχιστον n κλειδιά θα πάρουν τιμές που θα συμπέσουν με τις τιμές άλλων κλειδιών.

Κατακερματισμός

- Η τεχνική αυτή ονομάζεται *κατακερματισμός* (hashing). Επίσης ονομάζεται *αποθήκευση διασποράς* (scatter storage).
- Η συνάρτηση που χρησιμοποιούμε για την απεικόνιση ονομάζεται *συνάρτηση κατακερματισμού* (hash function).
- Ο πίνακας στον οποίο αντιστοιχούμε τα κλειδιά ονομάζεται *πίνακας κατακερματισμού* (hash table).
- Τα κελιά του πίνακα ονομάζονται *κάδοι* (buckets) ή *θέσεις* (slots).

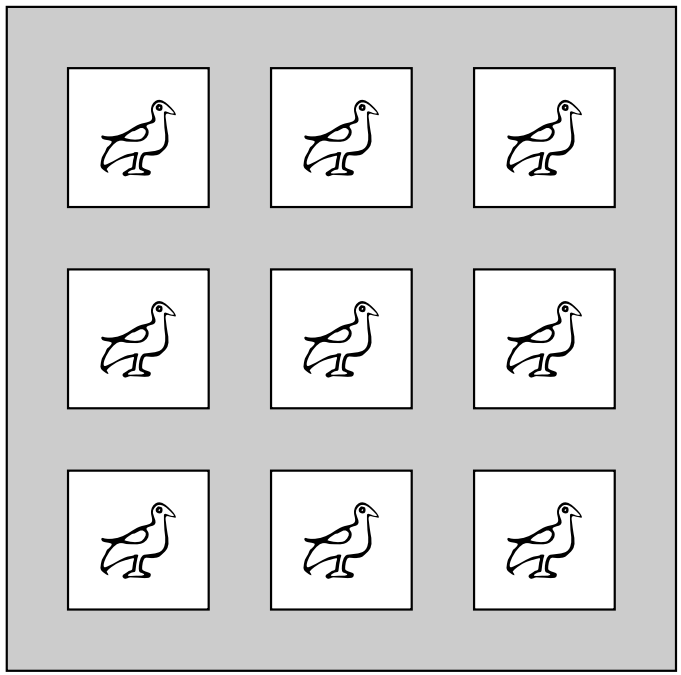
Είδη Συναρτήσεων Κατακερματισμού

- Αν η συνάρτηση καταφέρνει να αντιστοιχεί όλα τα κλειδιά σε διακριτές τιμές, τότε ονομάζεται *τέλεια συνάρτηση κατακερματισμού* (perfect hash function).
- Αν η συνάρτηση καταφέρνει να αντιστοιχεί όλα τα κλειδιά σε διακριτές τιμές και το εύρος τιμών είναι ακριβώς ίσο με το μέγεθος του πίνακα κατακερματισμού, ώστε να μην ξοδεύεται περιττός χώρος, τότε ονομάζεται *ελάχιστη τέλεια συνάρτηση κατακερματισμού* (miminal perfect hash function).
- Αν αυτό δεν είναι δυνατόν, τότε λέμε ότι έχουμε *σύγκρουση* (collision).
- Αν το μέγεθος του πίνακα είναι μικρότερο από το πλήθος των δυνατών κλειδιών, τότε οι συγκρούσεις είναι αναπόφευκτες, οπότε το ζητούμενο είναι να τις αποφεύγουμε όσο είναι δυνατόν.

Η Αρχή του Περιστερεώνα

- Οι συγκρούσεις είναι απόρροια της *αρχής του περιστερεώνα* (pigeonhole principle).
- Αν έχουμε n αντικείμενα και m δυνατές θέσεις, με $n > m$ και θέλουμε να τοποθετήσουμε τα αντικείμενα στις θέσεις, τουλάχιστον μία θέση θα βρεθεί να περιέχει περισσότερα από ένα αντικείμενο.

Η Αρχή του Περιστερεώνα



?

Σύμπτωση Αριθμού Τριχών της Κεφαλής

- Σε κάθε μεγάλη πόλη θα υπάρχουν τουλάχιστον δύο άνθρωποι με ακριβώς τον ίδιο αριθμό τριχών στις κεφαλές τους.
- Ο μέσος αριθμών τριχών στην κεφαλή είναι περίπου 150.000.
- Φυσικά ο αριθμός τριχών στην κεφαλή του καθενός διαφέρει, αλλά μπορούμε να είμαστε σίγουροι ότι δεν υπάρχουν άνθρωποι με περισσότερες από 1.000.000 τρίχες στο κεφάλι τους.
- Άρα σε κάθε πόλη με περισσότερους από 1.000.000 κατοίκους θα υπάρχουν τουλάχιστον δύο με τη συνθήκη που θέλουμε.

Το Παράδοξο των Γενεθλίων (1)

- Ποιος είναι ο ελάχιστος αριθμός ατόμων που πρέπει να είναι μέσα σε ένα δωμάτιο, ώστε δύο από αυτά να έχουν γενέθλια την ίδια ημέρα;
- Έστω η πιθανότητα να συμβεί αυτό είναι $P(B)$.
- Θέλουμε να βρούμε τον ελάχιστο αριθμό ατόμων ώστε να έχουμε $P(B) > 0{,}5$.

Το Παράδοξο των Γενεθλίων (2)

- Είναι πιο εύκολο να υπολογίσουμε την πιθανότητα $P(\overline{B})$ να μην υπάρχει κανένας που να έχει κοινά γενέθλια με άλλον.
- Από τους βασικούς νόμους των πιθανοτήτων έχουμε $P(\overline{B}) = 1 - P(B)$.

Το Παράδοξο των Γενεθλίων (3)

- Ας πάρουμε τα άτομα ένα-ένα.
- Το πρώτο μπορεί να έχει γενέθλια οποιαδήποτε ημέρα του χρόνου.
- Αυτό έχει πιθανότητα 365/365.

Το Παράδοξο των Γενεθλίων (4)

- Τα γενέθλια του δεύτερου ατόμου μπορούν να πέσουν οποιαδήποτε ημέρα εκτός από την ημέρα των γενεθλίων του πρώτου.
- Η πιθανότητα να συμβεί αυτό είναι $364/365$.
- Άρα συνολικά η πιθανότητα για δύο άτομα είναι $365/365 \times 364/365$.

Το Παράδοξο των Γενεθλίων (5)

- Αν συνεχίσουμε έτσι για n άτομα θα έχουμε: $365/365 \times 364/365 \times \cdots \times (365 - n + 1)/365$.
- Αν κάνουμε τους υπολογισμούς βρίσκουμε ότι για $n = 23$ έχουμε $P(\overline{B}) = 365/365 \times 364/365 \times \cdots \times 343/365 \approx 0.49$.
- Άρα $P(B) \approx 0.51$.
- Συνεπώς αν έχουμε 23 άτομα σε ένα δωμάτιο είναι πιο πιθανό δύο από αυτά να έχουν γενέθλια την ίδια ημέρα από ό,τι να μην έχουν.
- Φυσικά όσο περισσότερα άτομα έχουμε στο δωμάτιο, τόσο περισσότερο αυξάνει η πιθανότητα.

Επισκόπηση

Αναζήτηση Συναρτήσεων Κατακερματισμού

- Έστω ότι τα κλειδιά μας είναι διευθύνσεις (δρόμος και αριθμός) στα αγγλικά.
- Έστω ότι χρησιμοποιούμε 25 χαρακτήρες.
- Τότε τα δυνατά κλειδιά είναι $37^{25} = 1{,}6 \times 10^{39}$: κάθε χαρακτήρας μπορεί να είναι ένα γράμμα, ένα ψηφίο, ή το κενό.
- Προφανώς τα πραγματικά κλειδιά που θα χειριστούμε θα είναι πολύ λιγότερα. Έστω ότι περιμένουμε 100.000 κλειδιά.
- Χρειαζόμαστε τότε μια συνάρτηση κατακερματισμού που να μπορεί να αντιστοιχίσεις από ένα εύρος $1{,}6 \times 10^{39}$ σε 100.000 διευθύνσεις του πίνακα κατακερματισμού.
- Φυσικά μπορεί να υπάρχουν συγκρούσεις, αλλά θέλουμε να τις ελαχιστοποιήσουμε.

Συνάρτηση Κατακερματισμού Ακεραίων

- Αν τα κλειδιά μας είναι ακέραιοι αριθμοί, μια συνάρτηση κατακερματισμού που δουλεύει καλά στην πράξη είναι η:

$$h(K) = K \bmod m$$

όπου K είναι το κλειδί και m το μέγεθος του πίνακα κατακερματισμού.

- Η συνάρτηση δουλεύει και για κλειδιά που είναι αρνητικοί αριθμοί.
- Για οποιοδήποτε K έχουμε $K \bmod m = r$ όπου $K = qm + r$ με q το $\lfloor K/m \rfloor$. Συνεπώς $r = K - m\lfloor K/m \rfloor$.
- Για παράδειγμα, $-6 \bmod 10 = 4$ επειδή $\lfloor -6/10 \rfloor = -1$ και $r = -6 - 10(-1) = -6 + 10 = 4$.

Αλγόριθμος Κατακερματισμού Ακεραίων

Algorithm: An integer hash function.

`IntegerHash`$(k, m) \rightarrow h$

Input: k, an integer number
m, the size of the hash table

Output: h, the hash value of k

1 $h \leftarrow k \bmod m$

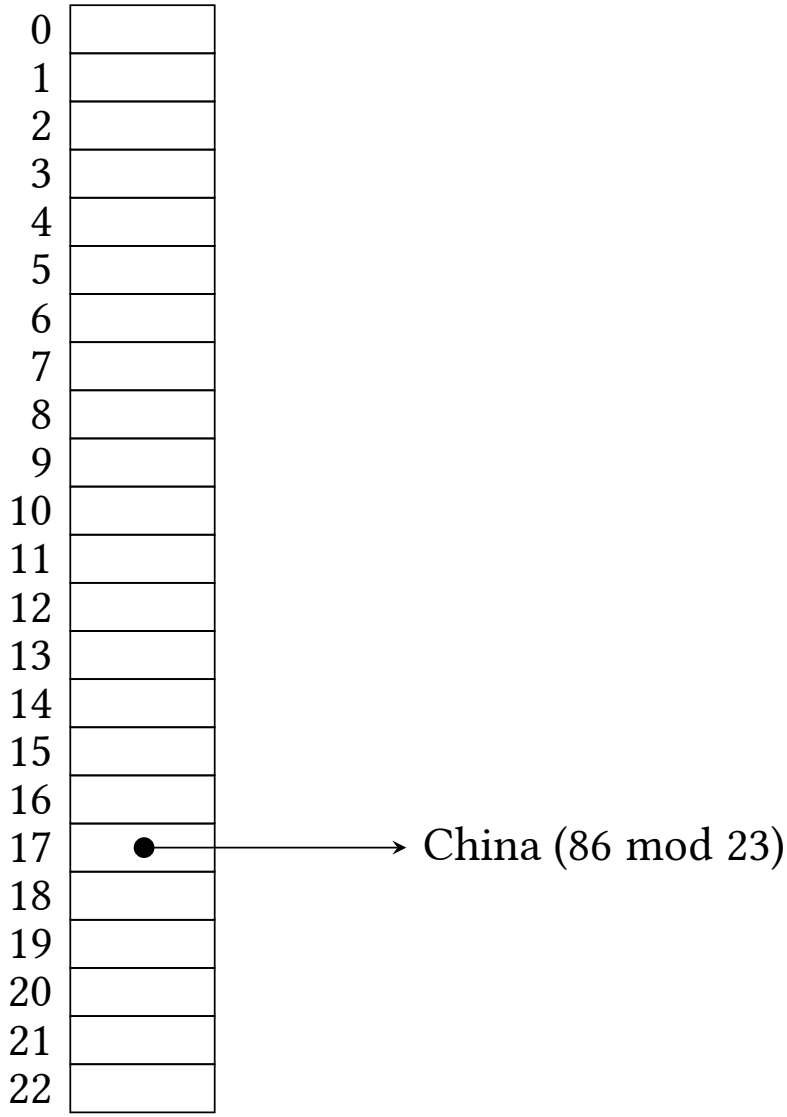
2 **return** h

Παράδειγμα Κατακερματισμού Ακεραίων (1)

China	86	Japan	81
India	91	Mexico	52
United States	1	Philippines	63
Indonesia	62	Vietnam	84
Brazil	55	Ethiopia	251
Pakistan	92	Egypt	20
Nigeria	234	Germany	49
Bangladesh	880	Iran	98
Russia	7		

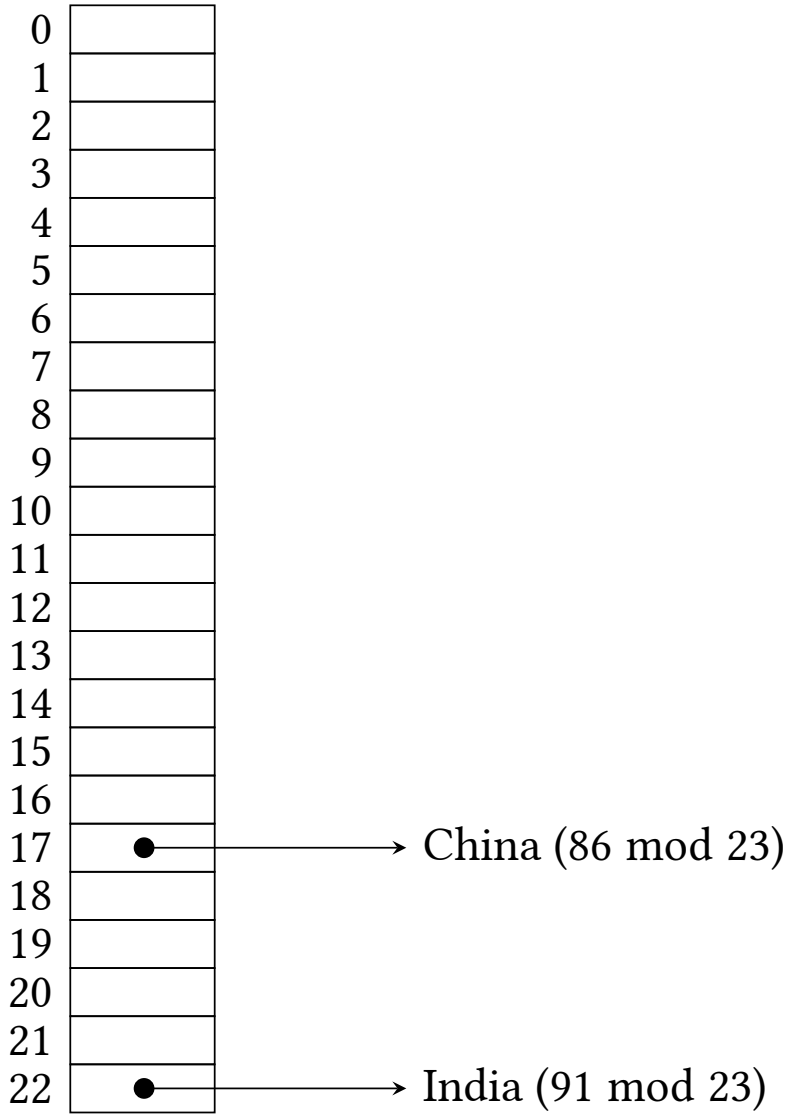
Πρώτες δεκαεπτά χώρες ως προς τον πληθυσμό με τους τηλεφωνικούς κωδικούς κλήσης τους (2015).

Παράδειγμα Κατακερματισμού Ακεραίων (2)



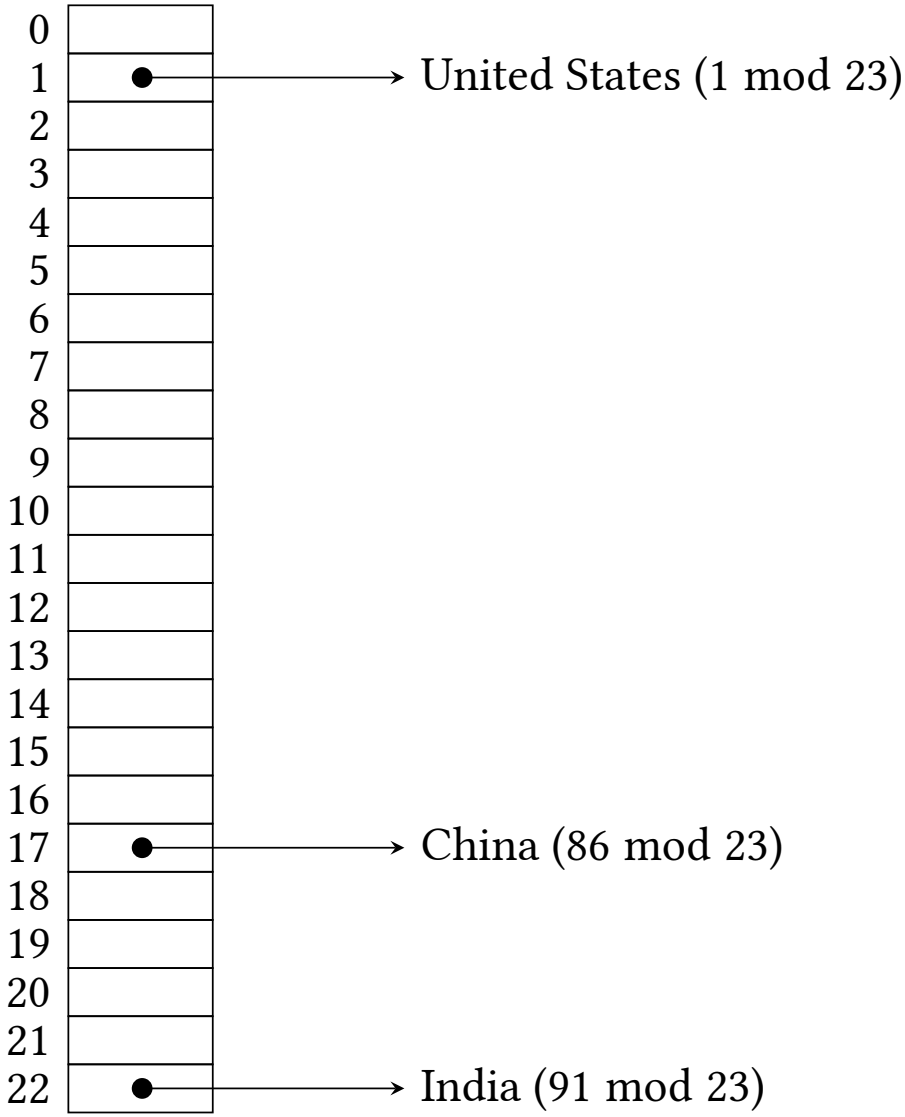
Εισαγωγή Κίνας.

Παράδειγμα Κατακερματισμού Ακεραίων (2)



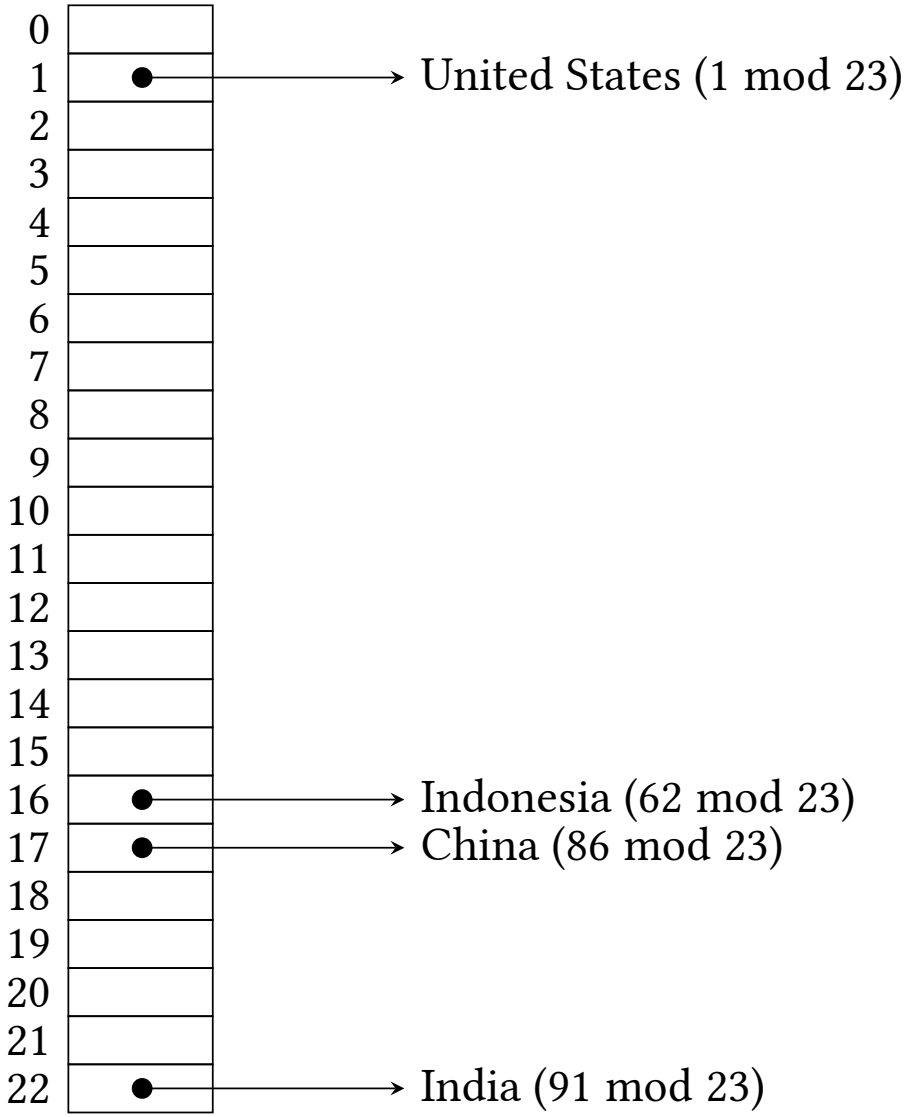
Εισαγωγή Ινδίας.

Παράδειγμα Κατακερματισμού Ακεραίων (3)



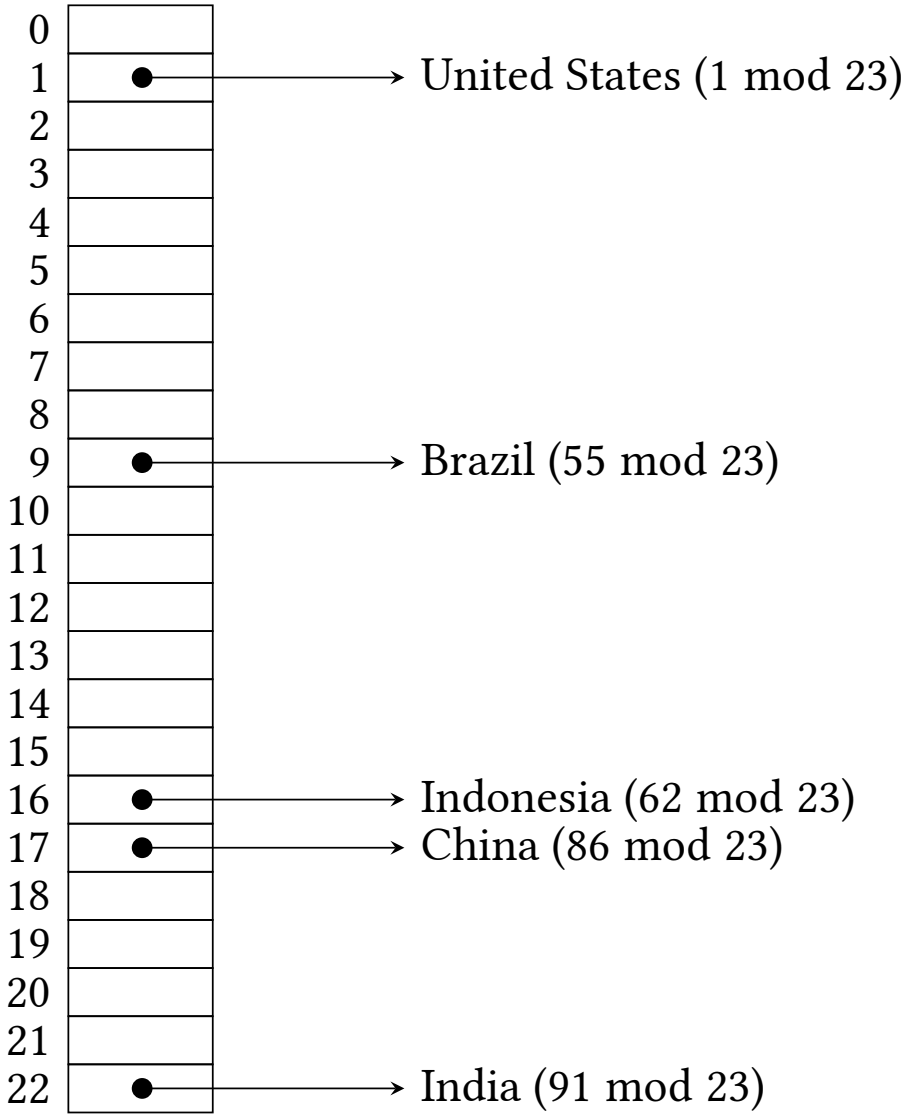
Εισαγωγή Η.Π.Α.

Παράδειγμα Κατακερματισμού Ακεραίων (4)



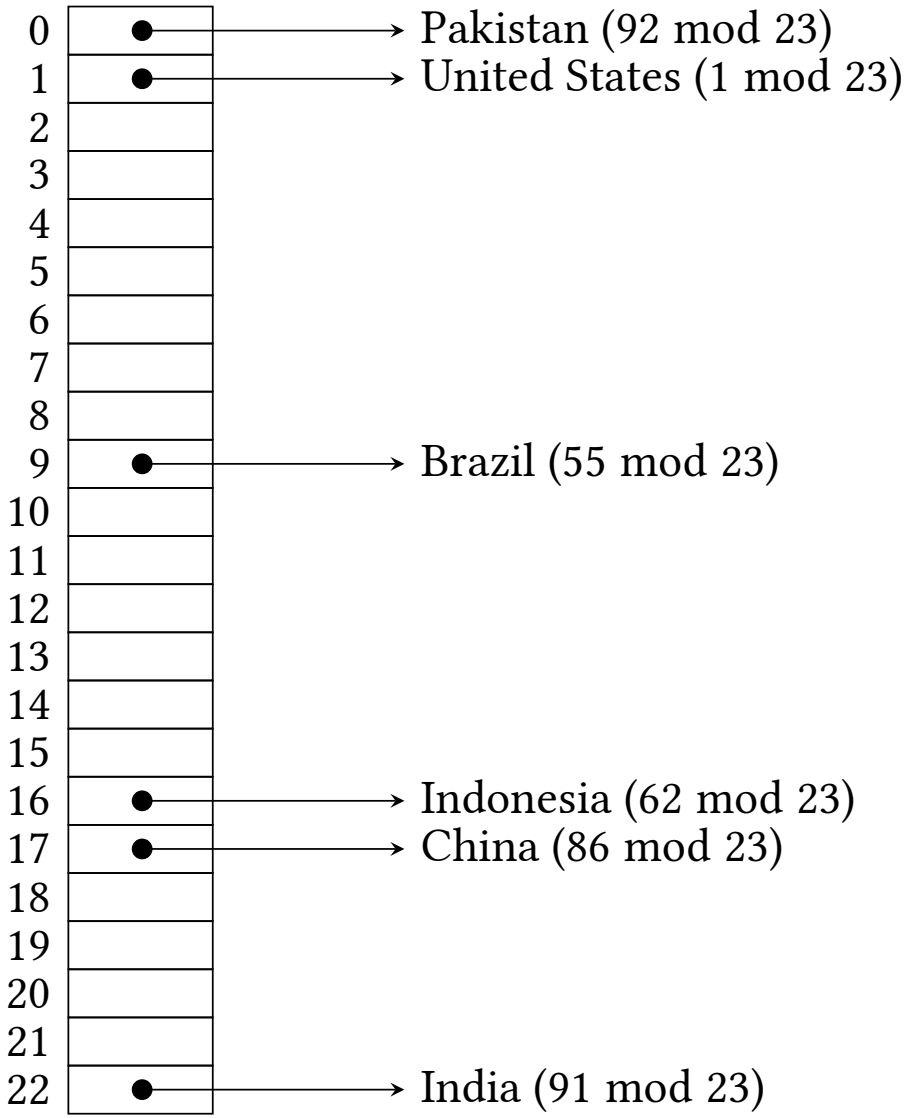
Εισαγωγή Ινδονησίας.

Παράδειγμα Κατακερματισμού Ακεραίων (5)



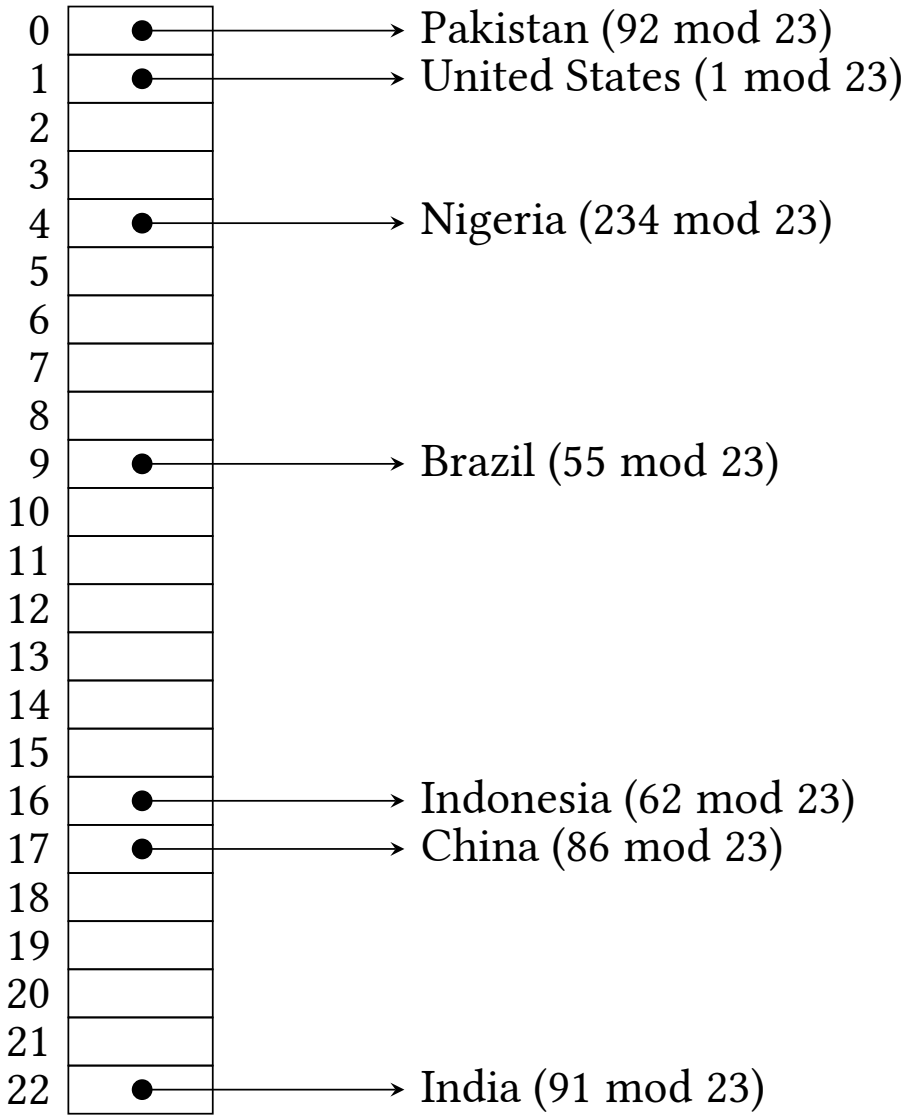
Εισαγωγή Βραζιλίας.

Παράδειγμα Κατακερματισμού Ακεραίων (6)



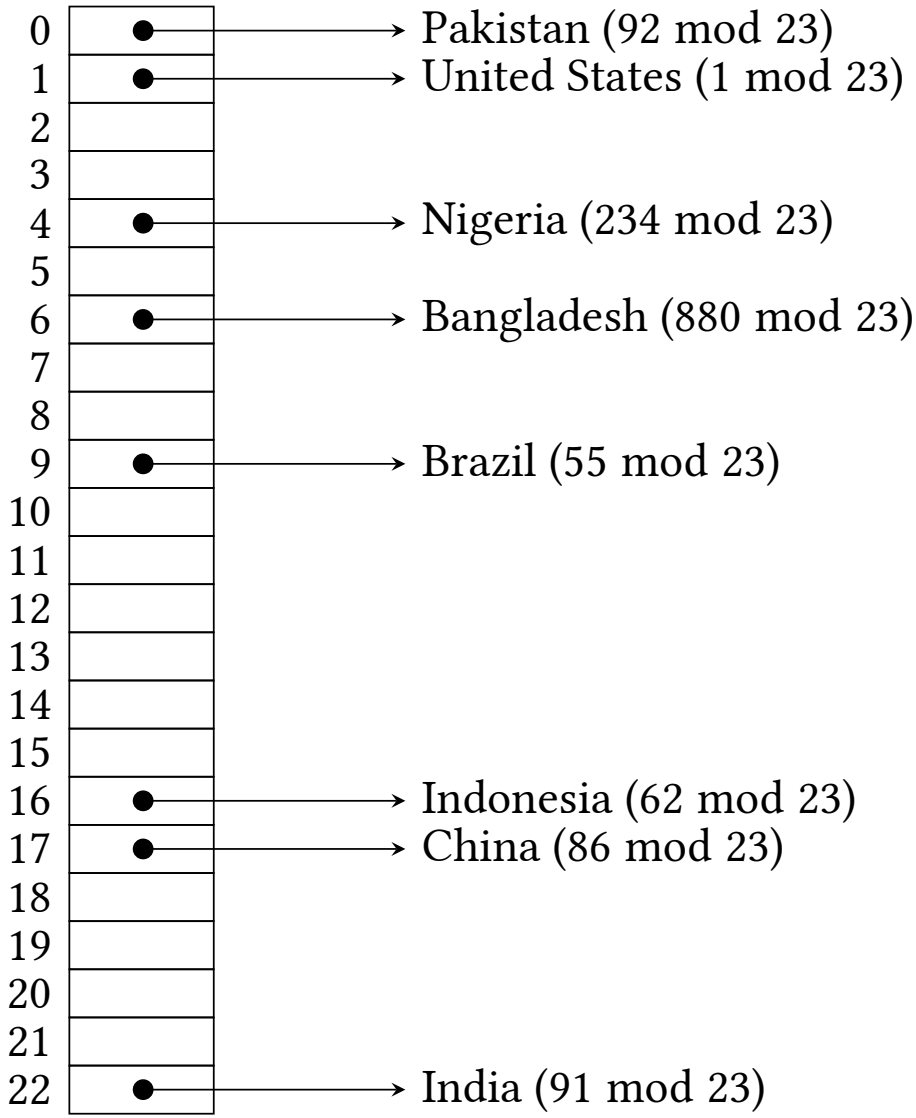
Εισαγωγή Πακιστάν.

Παράδειγμα Κατακερματισμού Ακεραίων (7)



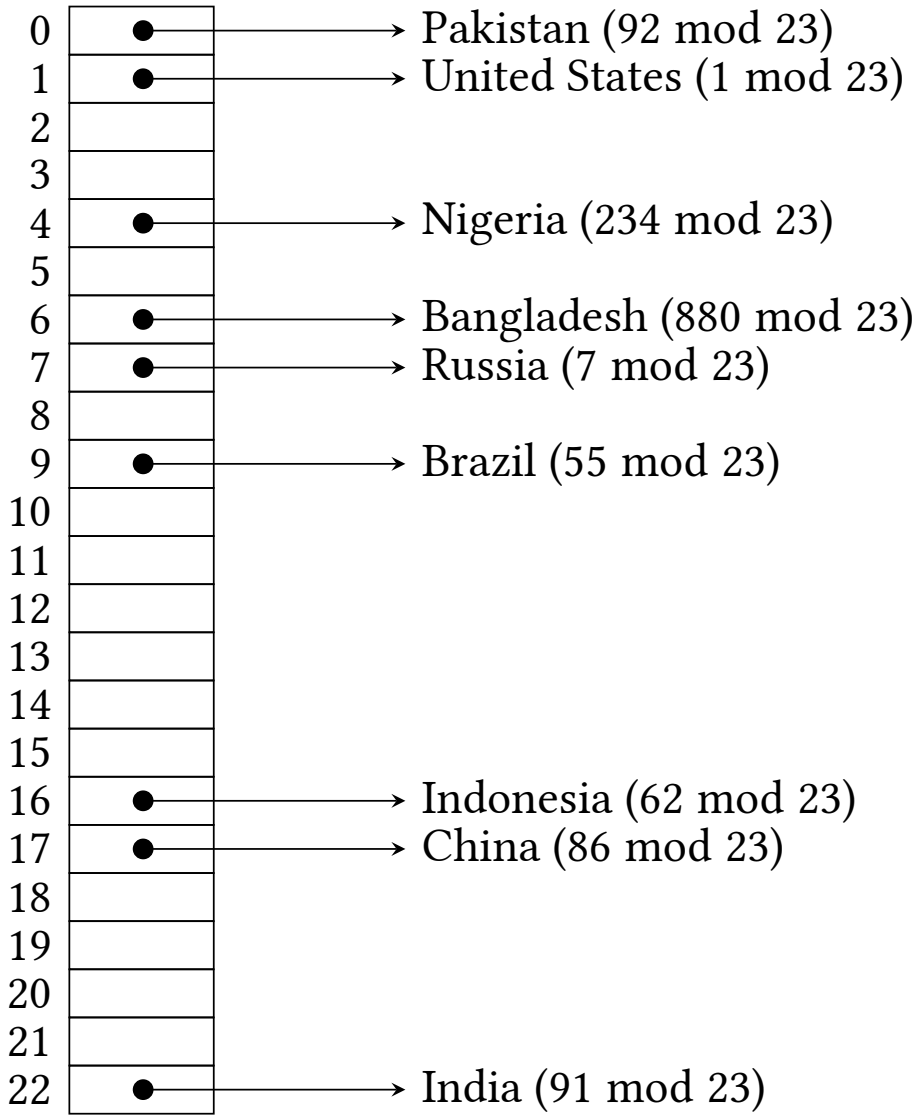
Εισαγωγή Νιγηρίας.

Παράδειγμα Κατακερματισμού Ακεραίων (8)



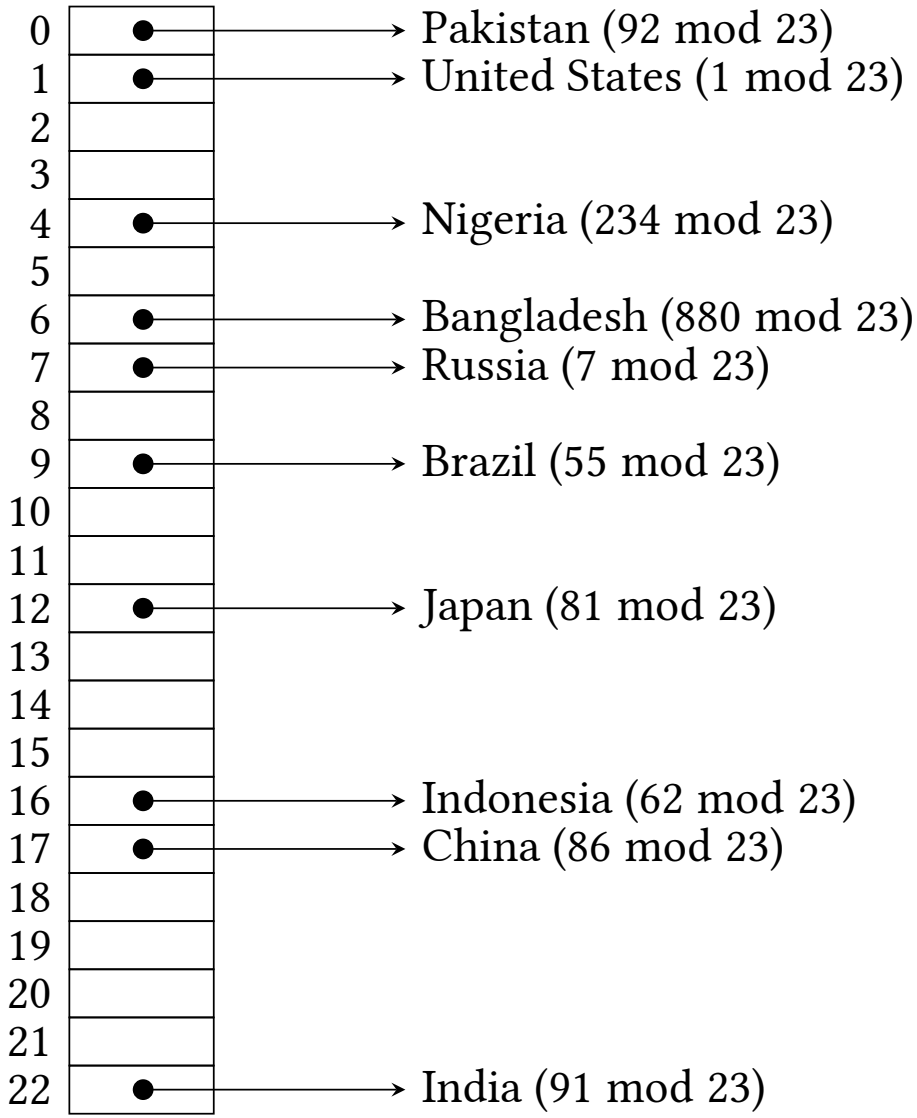
Εισαγωγή Μπανγκλαντές.

Παράδειγμα Κατακερματισμού Ακεραίων (9)



Εισαγωγή Ρωσίας.

Παράδειγμα Κατακερματισμού Ακεραίων (10)



Εισαγωγή Ιαπωνίας.

Απαιτήσεις Πίνακα

- Η μέθοδος αυτή δουλεύει καλά, με την έννοια ότι δεν προκαλεί πολλές συγκρούσεις.
- Αυτό, όμως, εξαρτάται από το μέγεθος του πίνακα που χρησιμοποιείται.
- Για παράδειγμα, αν το μέγεθος του πίνακα είναι 10^x δημιουργούνται πολλές συγκρούσεις.
- Ο λόγος είναι ότι το υπόλοιπο της διαίρεσης ενός φυσικού αριθμού με το 10^x είναι απλώς τα x τελευταία ψηφία του αριθμού.
- Άρα όλοι οι αριθμοί με τα ίδια x τελευταία ψηφία θα έχουν την ίδια τιμή στη συνάρτηση κατακερματισμού.

Πίνακας Μεγέθους 10^x

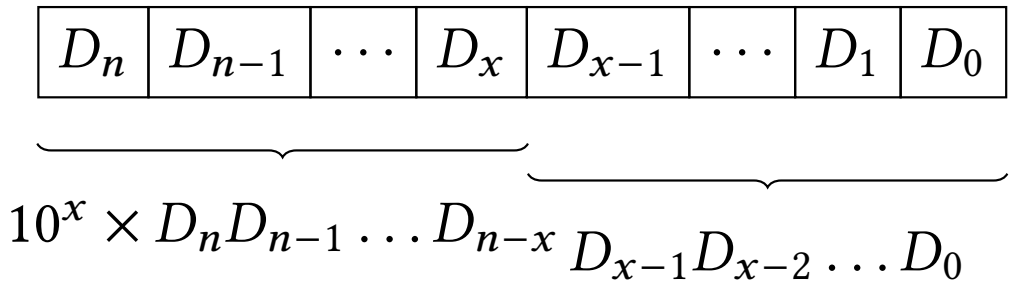
- Για παράδειγμα, $12345 \bmod 100 = 45$, $2345 \bmod 100 = 45$, κ.λπ.
- Γενικά, αφού κάθε θετικός ακέραιος με n ψηφία $D_n D_{n-1} \ldots D_1 D_0$ έχει τιμή $D_n \times 10^n + D_{n-1} \times 10^{n-1} + \cdots + D_1 \times 10^1 + D_0 \times 10^0$.
- Για κάθε δύναμη του δέκα, 10^x, έχουμε:

$$D_n D_{n-1} \ldots D_1 D_0 = 10^x \times D_n D_{n-1} \ldots D_x + D_{x-1} D_{x-2} \ldots D_1 D_0$$

- Συνεπώς έχουμε:

$$D_n D_{n-1} \ldots D_1 D_0 \bmod 10^x = D_{x-1} D_{x-2} \ldots D_1 D_0$$

Πίνακας Μεγέθους 10^x



Ανάλυση υπολοίπου αριθμού σε διαίρεση με δύναμη 10^x.

Μέγεθος Πίνακα σε άλλα Συστήματα Αρίθμησης

- Το ίδιο ακριβώς πρόβλημα εμφανίζεται αν ο αριθμός μας είναι σε άλλο σύστημα αρίθμησης, π.χ. δυαδικό.
- Αν χρησιμοποιούμε ένα σύστημα αρίθμησης με βάση το b, τότε αν το μέγεθος του πίνακα είναι b^x μόνο τα x τελευταία ψηφία θα έχουν σημασία στον υπολογισμό της συνάρτησης κατακερματισμού.

Ομοιόμορφη Κατανομή

- Ιδανικά η συνάρτηση κατακερματισμού θα πρέπει να κατανέμει τα κλειδιά σε κάθε θέση του πίνακα με ίση πιθανότητα.
- Με άλλα λόγια, η κατανομή των κλειδιών θα πρέπει να είναι *ομοιόμορφη* (uniform).
- Μια κατανομή ονομάζεται ομοιόμορφη όταν όλες οι τιμές έχουν την ίδια πιθανότητα.
- Αυτό μας αποκλείει αμέσως κάποιες επιλογές για το μέγεθος του πίνακα κατακερματισμού. Αν έχει μέγεθος άρτιο αριθμό, τότε όλα τα άρτια κλειδιά θα πηγαίνουν στις άρτιες θέσεις του πίνακα και όλα τα περιττά στις περιττές θέσεις του πίνακα.

Επιλογή Μεγέθους Πίνακα

- Στην πράξη μια καλή επιλογή είναι το μέγεθος του πίνακα να είναι ένας πρώτος αριθμός.
- Έτσι αν θέλουμε να αποθηκεύσουμε μέχρι περίπου 1000 κλειδιά, θα χρησιμοποιήσουμε έναν πίνακα μεγέθους 997, που είναι πρώτος αριθμός, αντί για έναν πίνακα μεγέθους 1000.

Κατακερματισμός συμβολοσειρών

- Αν τα κλειδιά μας είναι συμβολοσειρές (strings) τότε μπορούμε να τα χειριστούμε σαν να ήταν αριθμοί σε ένα αριθμητικό σύστημα με την κατάλληλη βάση, π.χ. 26.
- Αν s είναι μία συμβολοσειρά με n χαρακτήρες, η αριθμητική τιμή της αντιστοιχεί είναι η:

$$v = \texttt{Ordinal}(s[0])b^{n-1} + \texttt{Ordinal}(s[1])b^{n-2} + \cdots + \texttt{Ordinal}(s[n-1])b^0$$

- Η συνάρτηση `Ordinal` επιστρέφει τη θέση του γράμματος στο αλφάβητο και b είναι η βάση του αριθμητικού μας συστήματος.
- Στο τέλος υπολογίζουμε την τελική τιμή κατακερματισμού:

$$h = v \bmod m$$

Συνάρτηση Κατακερματισμού Συμβολοσειρών

Algorithm: A string hash function.

`StringHash`$(s, b, m) \rightarrow h$

Input: s, a string
b, the base of the number system
m, the size of the hash table

Output: h, the hash value of s

1 $v \leftarrow 0$
2 $n \leftarrow |s|$
3 **for** $i \leftarrow 0$ **to** n **do**
4 $\quad v \leftarrow v + \texttt{Ordinal}(s[i]) \cdot b^{n-1-i}$
5 $h \leftarrow v \bmod m$
6 **return** h

Παράδειγμα Συνάρτησης Κατακερματισμού Συμβολοσειρών

$$v_0 = \texttt{Ordinal}(\text{`H'}) \cdot 26^4 = 7 \cdot 456.976 = 3.198.832$$
$$v_1 = 3.198.832 + \texttt{Ordinal}(\text{`E'}) \cdot 26^3 = 3.198.832 + 4 \cdot 26^3 = 3.269.136$$
$$v_2 = 3.269.136 + \texttt{Ordinal}(\text{`L'}) \cdot 26^2 = 3.269.136 + 11 \cdot 26^2 = 3.276.572$$
$$v_3 = 3.276.572 + \texttt{Ordinal}(\text{`L'}) \cdot 26^1 = 3.276.572 + 11 \cdot 26 = 3.276.858$$
$$v_4 = 3.276.858 + \texttt{Ordinal}(\text{`O'}) = 3.276.858 + 14 = 3.276.872$$
$$h = 3.276.872 \bmod 31 = 17$$

Υπολογισμός συνάρτησης κατακερματισμού για το κλειδί «HELLO».

Αντιστοιχία Συνάρτησης με Πολυώνυμο

- Με τον τρόπο αυτό η συνάρτηση αντιστοιχεί σε υπολογισμό πολυωνύμου.
- Συγκεκριμένα, βρίσκουμε το υπόλοιπο της διαίρεσης με το m του πολυωνύμου:

$$p(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_0$$

- Οι συντελεστές του πολυωνύμου είναι οι χαρακτήρες της συμβολοσειράς και ο υπολογισμός γίνεται για $x = b$.
- Στην περίπτωση του κλειδιού «HELLO» το πολυώνυμο είναι το:

$$p(x) = 7x^4 + 4x^3 + 11x^2 + 11x + 14$$

το οποίο υπολογίζουμε για $x = 26$.

Πολυπλοκότητα Υπολογισμού Πολυωνύμου

- Στον παραπάνω αλγόριθμο, για να υπολογίσουμε ένα πολυώνυμο βαθμού n, $p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_0$, υπολογίζουμε όλες τις δυνάμες από τα αριστερά προς το δεξιά.
- Για να υπολογίσουμε το $a_n x^n$ χρειαζόμαστε $n-1$ πολλαπλασιασμούς (για την ύψωση σε δύναμη) συν ένα πολλαπλασιασμό με το a_n, άρα n πολλαπλασιασμούς συνολικά.
- Για τον υπολογισμό του $a_{n-1} x^{n-1}$ χρειαζόμαστε $n-1$ πολλαπλασιασμούς.
- Συνολικά θέλουμε $n + (n-1) + \cdots + 1 = n(n+1)/2$ πολλαπλασιασμούς και n προσθέσεις για να αθροίσουμε τους όρους.

Ο Κανόνας του Horner

- Ένας πιο αποτελεσματικός τρόπος για τον υπολογισμό πολυωνύμου είναι ο *κανόνας του Horner* (Horner's rule).
- Βασίζεται στην αναδιάταξη του πολυωνύμου ως εξής:

$$a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n = \big(\ldots (a_n x + a_{n-1}) x + \cdots \big) x + a_0$$

- Με αυτή τη διάταξη κάνουμε τον υπολογισμό από τα μέσα προς τα έξω.

Παράδειγμα Κανόνα του Horner

$$\Big(\ldots(a_n x + a_{n-1})x + \cdots\Big)x + a_0$$

$$r_n$$

$$\Big(\ldots(\underbrace{r_n x + a_{n-1}}_{r_{n-1}})x + \cdots\Big)x + a_0$$

$$r_{n-1}$$

$$\Big(\ldots(r_{n-1}x + a_{n-2})x + \cdots\Big)x + a_0$$

$$\vdots$$

$$\underbrace{(b_2 x + a_1)}_{r_1}x + a_0$$

$$\underbrace{r_1 x + a_0}_{r_0}$$

$$\Big(\big(\underbrace{(7x+4)}_{186x+11}x + 11\big)x + 11\Big)x + 14$$

$$\big(\underbrace{(186x + 11)}_{4.847x+11}x + 11\big)x + 14$$

$$\underbrace{(4.847x + 11)}_{126.033x+14}x + 14$$

$$126.033x + 14$$

$$3.276.872$$

Όπου $x = 26$.

Αλγόριθμος του Horner

Algorithm: Horner's rule.

`HornerRule`$(A, x) \rightarrow r$

Input: A, an array containing the coefficients of a polynomial of degree n
x, the point at which to evaluate the polynomial

Output: r, the value of the polynomial at x

```
1  r ← 0
2  foreach c in A do
3      r ← r · x + c
4  return r
```

Απόδοση Αλγορίθμου Horner

- Η γραμμή 3 του αλγορίθμου εκτελείται n φορές.
- Συνεπώς απαιτούνται n πολλαπλασιασμοί και n προσθέσεις.

Βελτίωση Κατακερματισμού Συμβολοσειρών

Στην πράξη $a \bmod b$ ισχύουν οι παρακάτω ιδιότητες:

$$(a + b) \bmod m = \big((a \bmod m) + (b \bmod m)\big) \bmod m$$

$$(ab) \bmod m = \big((a \bmod m) \cdot (b \bmod m)\big) \bmod m$$

Αυτό μας επιτρέπει να αποφύγουμε τον υπολογισμό μεγάλων δυνάμεων, βρίσκοντας το υπόλοιπο της διαίρεσης με το m καθώς προχωράμε στους υπολογισμούς μας.

Βελτιωμένος Κατακερματισμός Συμβολοσειρών

Algorithm: An optimized string hash function.

`OptimizedStringHash`$(s, b, m) \rightarrow h$

Input: s, a string
b, the base of the number system
m, the size of the hash table

Output: h, the hash value of s

```
1  h ← 0
2  foreach c in s do
3      h ← (b · h + Ordinal(c)) mod m
4  return h
```

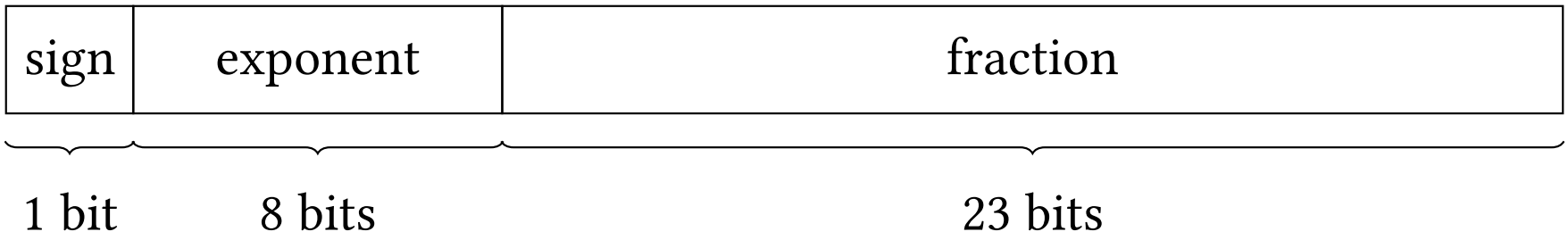
Παράδειγμα Βελτιωμένου Αλγορίθμου

$$h_0 = \texttt{Ordinal}(\text{`H'}) \bmod 31 = 7 \bmod 31 = 7$$
$$h_1 = \left(26 \cdot 7 + \texttt{Ordinal}(\text{`E'})\right) \bmod 31 = (182 + 4) \bmod 31 = 0$$
$$h_2 = \left(26 \cdot 0 + \texttt{Ordinal}(\text{`L'})\right) \bmod 31 = 11 \bmod 31 = 11$$
$$h_3 = \left(26 \cdot 11 + \texttt{Ordinal}(\text{`L'})\right) \bmod 31 = (286 + 11) \bmod 31$$
$$h_4 = \left(26 \cdot 18 + \texttt{Ordinal}(\text{`O'})\right) \bmod 31 = 482 \bmod 31 = 17$$

Κατακερματισμός Πραγματικών Αριθμών

- Εκτός από ακέραιοι και συμβολοσειρές τα κλειδιά μας μπορεί φυσικά να είναι και πραγματικοί αριθμοί (floating point numbers).
- Τι είδους συνάρτηση κατακερματισμού μπορούμε να εφαρμόσουμε σε πραγματικούς αριθμούς;
- Μια ιδέα είναι να τους μετατρέψουμε σε συμβολοσειρά και να δουλέψουμε με αυτήν. Π.χ. τον αριθμό 261,63 να τον μετατρέψουμε στη συμβολοσειρά «261,63».
- Αυτό όμως είναι γενικώς αργή διαδικασία.
- Επίσης δεν μπορούμε απλώς να αφαιρέσουμε την υποδιαστολή, ώστε από το 261,63 να πάρουμε 26163, γιατί ο υπολογιστής δεν αποθηκεύει με αυτόν τον τρόπο τους πραγματικούς αριθμούς.

Αναπαράσταση Πραγματικών Αριθμών



Οι πραγματικοί αριθμοί αποθηκεύονται στον υπολογιστή όπως παραπάνω (για 32 bits, με αντίστοιχο τρόπο αποθηκεύονται και αριθμοί 64 bits). Η τιμή ενός αποθηκευμένου αριθμού είναι:

$$(-1)^s \times 1,f \times 2^{e-127}$$

όπου s είναι το πρόσημο (sign), f είναι το δεκαδικό μέρος (fraction, mantissa) και e είναι ο εκθέτης.

Δυαδικοί Δεκαδικοί Αριθμοί

- Αν ένας δυαδικός αριθμός έχει και ακέραιο και δεκαδικό μέρος, η τιμή του είναι ίση με το άθροισμα των δύο.
- Έτσι ο αριθμός $B_1B_0, B_1B_2 \ldots B_n$ έχει τιμή $B_1 \times 2^2 + B_0 + B_1 \times 2^{-1} + B_2 \times 2^{-2} + \cdots + B_n \times 2^{-n}$.
- Για παράδειγμα, ο αριθμός 1,01 στο δυαδικό σύστημα είναι ίσος με τον αριθμό $1 + 0 \times 2^{-1} + 1 \times 2^{-2} = 1{,}25$ στο δεκαδικό σύστημα.

Παραδείγματα Πραγματικών Αριθμών

0	01110011	00000110001001001101111

$$(-1)^0 \times 2^{115-127} \times 1{,}02400004864 = 0{,}00025$$

1	10010101	10001101010101101100000

$$(-1)^1 \times 2^{149-127} \times 1{,}55210494995 = -6.510.000$$

0	10000111	00000101101000010100100

$$(-1)^0 \times 2^{135-127} \times 1{,}02199220657 = 261{,}63$$

Κατακερματισμός Πραγματικών Αριθμών

- Αν έχουμε έναν πραγματικό αριθμό στον υπολογιστή και θέλουμε να εφαρμόσουμε μια συνάρτηση κατακερματισμού σε αυτόν, ερμηνεύουμε τα bits του σαν να ήταν ακέραιος αριθμός.
- Στη συνέχεια εφαρμόζουμε τη μέθοδο κατακερματισμού ακεραίων.
- Για παράδειγμα, ο αριθμός 261,63 αναπαρίσταται ως 01000011100000101101000010100100.
- Αν τον θεωρήσουμε ακέραιο, τότε έχουμε

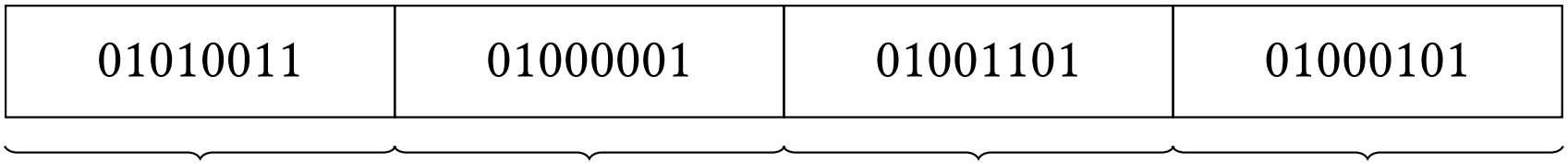
$$(01000011100000101101000010100100)_2 = (1132646564)_{10}$$

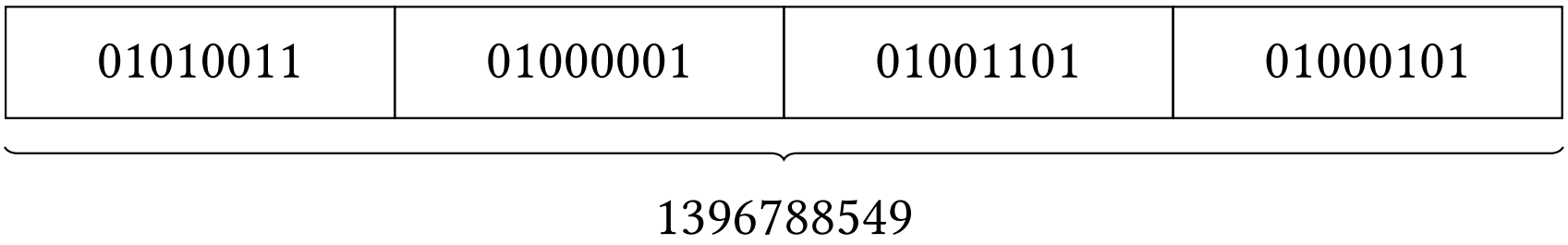
- Άρα χρησιμοποιούμε τον αριθμό 1132646564 ως είσοδο στη συνάρτηση κατακερματισμού ακεραίων.

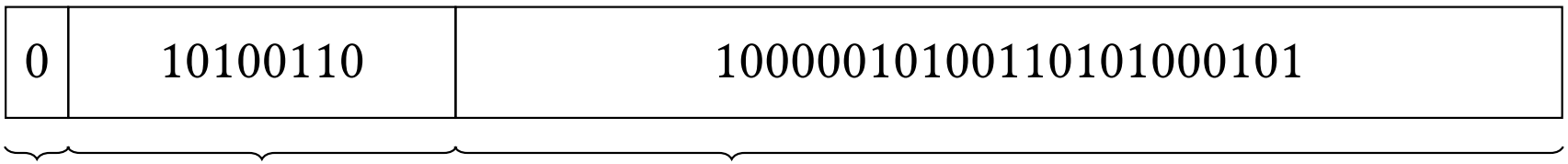
Ερμηνεία Bits

- Προσοχή! Ο ίδιος ο υπολογιστής δεν ξέρει τι σημαίνει μια ακολουθία bits.
- Τα bits μπορεί να αναπαριστάνουν συμβολοσειρά, ακέραιο αριθμό, ή πραγματικό αριθμό.
- Πρέπει στα προγράμματά μας να εξασφαλίζουμε ότι διαβάζουμε τα δεδομένα μας με τη σημασία που πραγματικά πρέπει να έχουν.

Παράδειγμα Ερμηνείας Bits





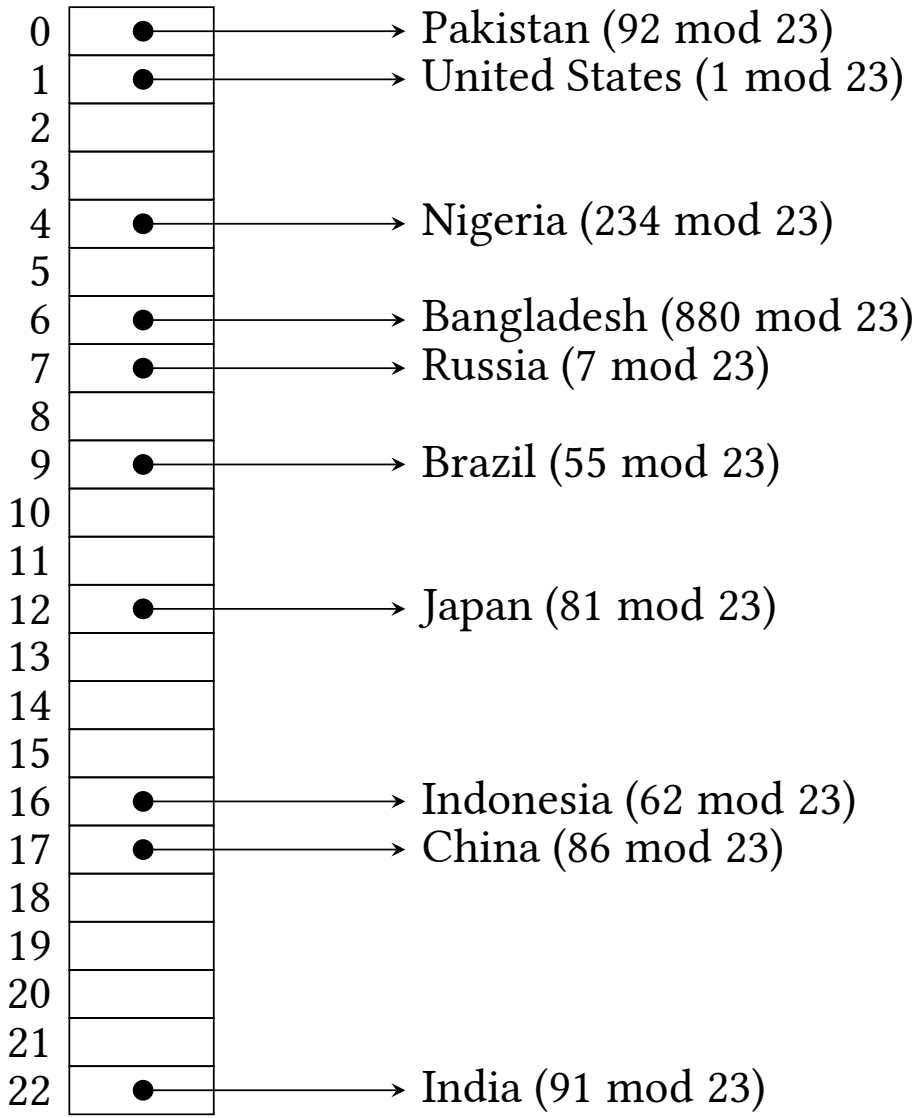


$$(-1)^0 \times 2^{166-127} \times 1{,}51017057896 = 8{,}30225055744 \times 10^{11}$$

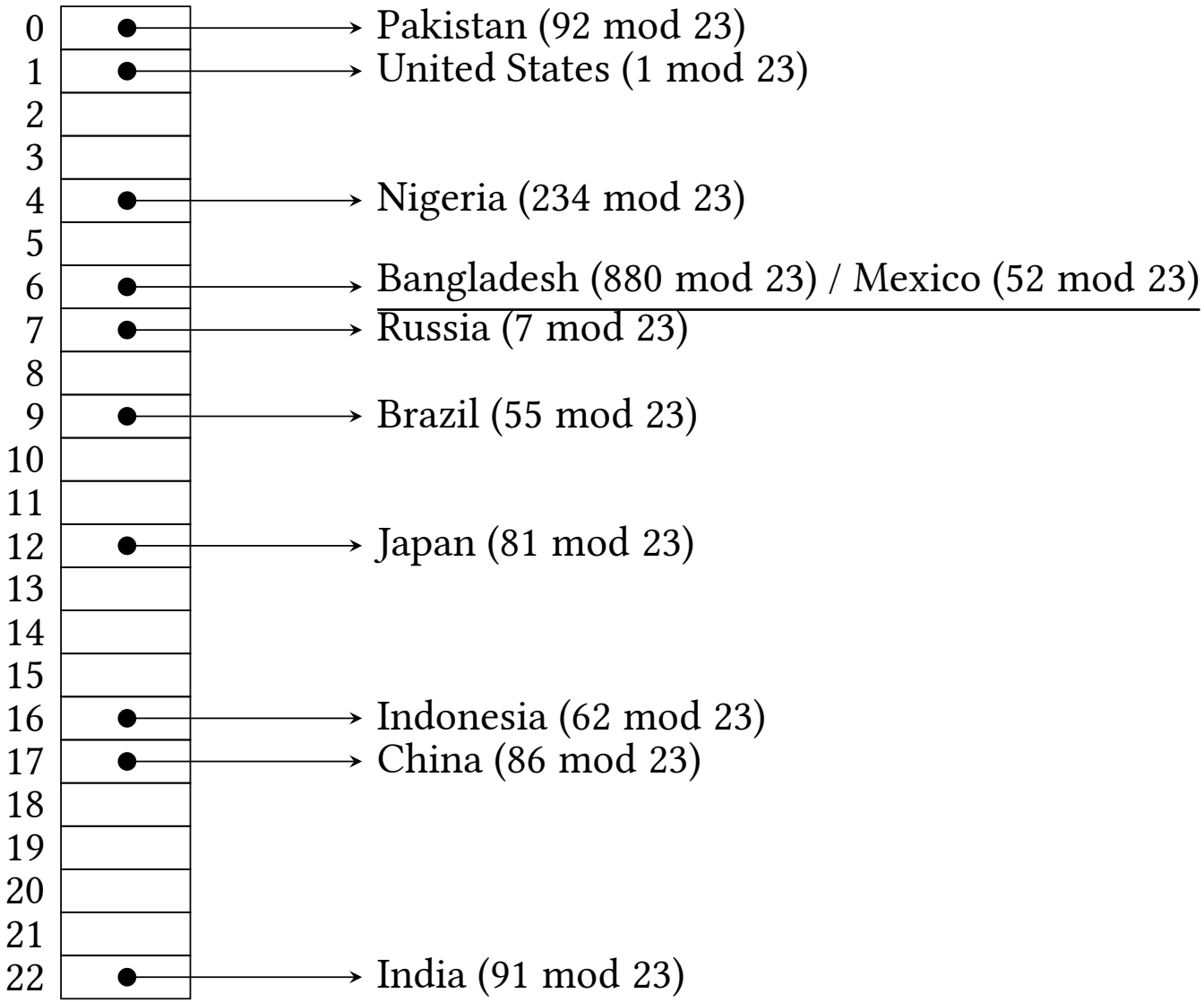
Επισκόπηση

- Λόγω της αρχής του περιστερεώνα δεν είναι δυνατόν να αποφύγουμε τις συγκρούσεις.
- Το καλύτερο που μπορούμε να πετύχουμε, με μία καλή συνάρτηση κατακερματισμού, είναι να τις περιορίσουμε όσο είναι δυνατόν.
- Θα πρέπει όμως να έχουμε έναν τρόπο να τις χειριστούμε όταν προκύπτουν.

Παράδειγμα Συγκρούσεων (1)



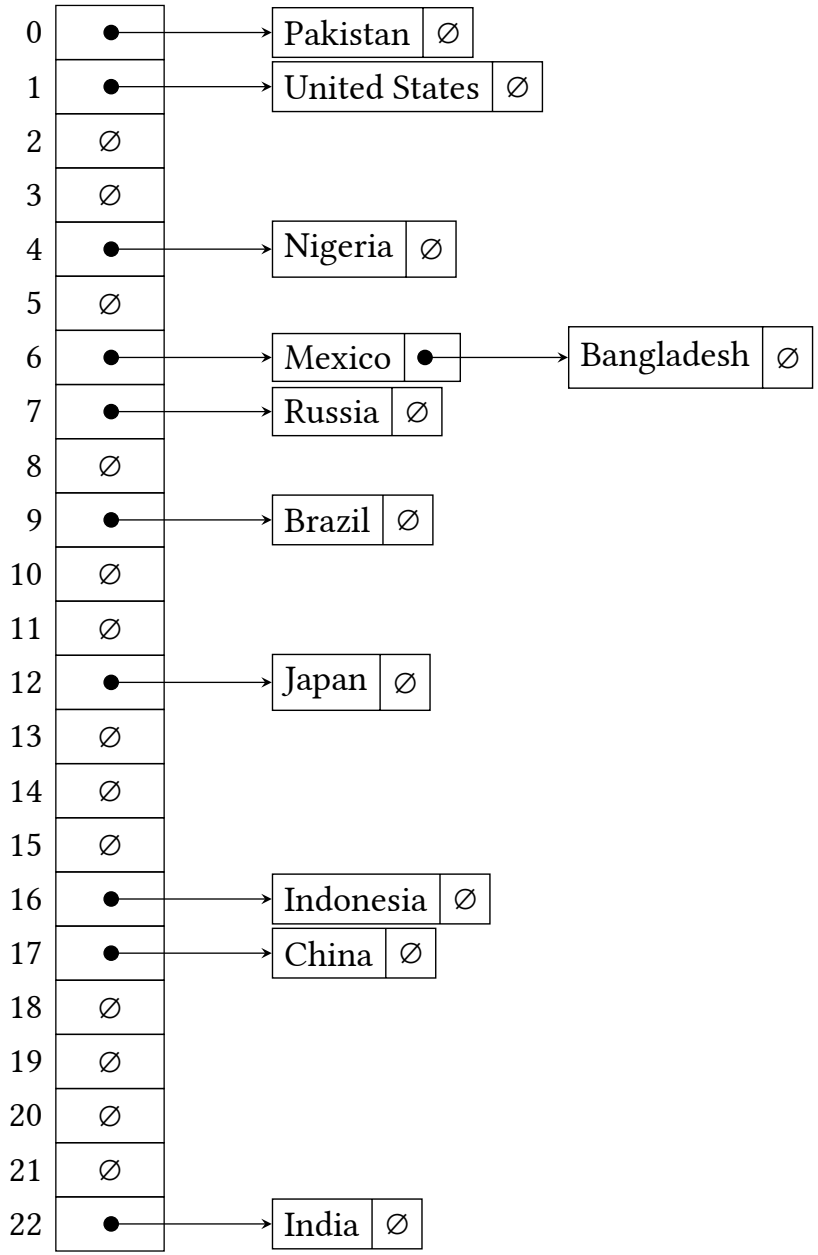
Παράδειγμα Συγκρούσεων (2)



Αλυσίδες

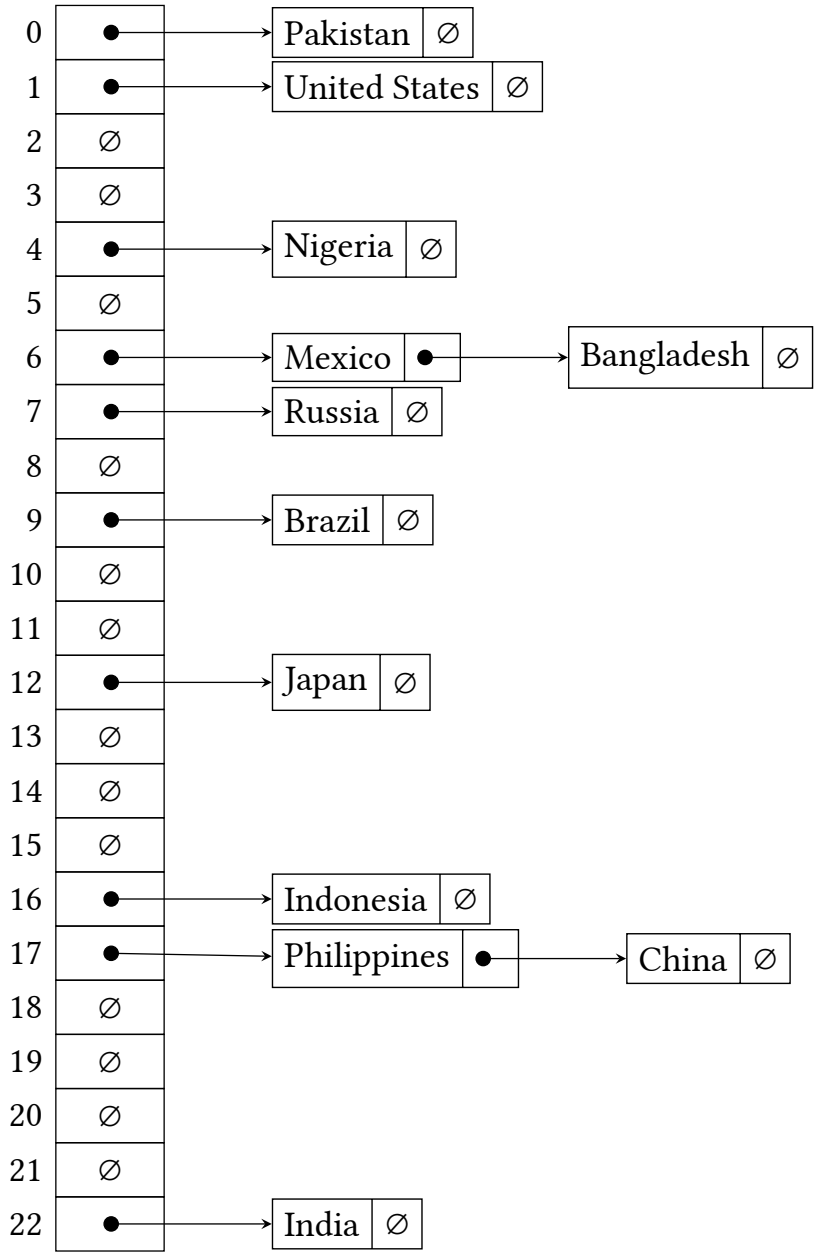
- Ο πιο δημοφιλής τρόπος να χειριστούμε τις συγκρούσεις είναι να μην αποθηκεύουμε στον πίνακα εγγραφές, αλλά λίστες εγγραφών.
- Κάθε κάδος θα δείχνει σε μία λίστα που θα περιέχει όλες τις εγγραφές που έχουν την ίδια τιμή συνάρτησης κατακερματισμού.
- Αυτό εξηγεί και τη σημασία του όρου «κάδος».
- Αν δεν υπάρχουν κλειδιά για κάποιο κάδο, αυτός κατά σύμβαση θα δείχνει στην τιμή NULL.
- Ονομάζουμε τις λίστες που δημιουργούνται *αλυσίδες* (chains).
- Η μέθοδος αυτή ονομάζεται *κατακερματισμός με ξεχωριστές αλυσίδες* (hashing with separate chaining).

Χειρισμός Συγκρούσεων (1)



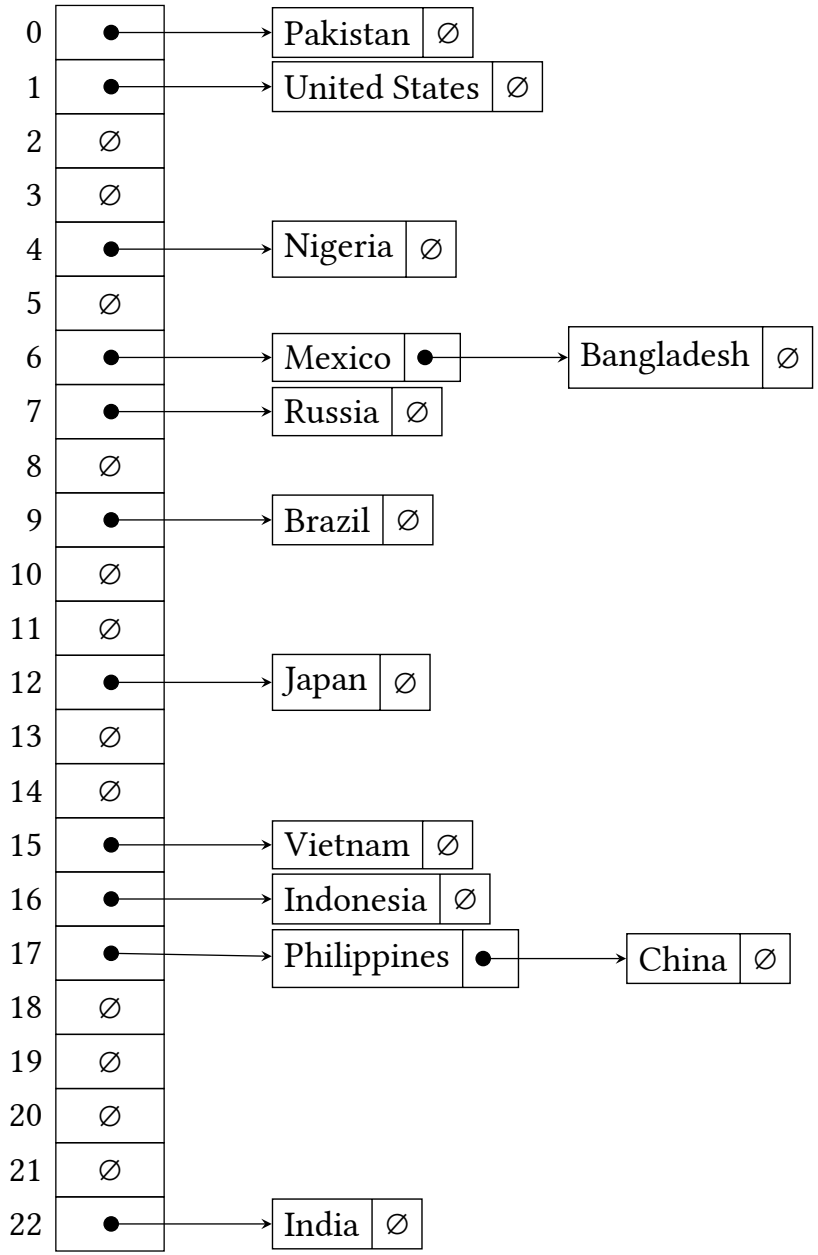
Εισαγωγή Mexico (σύγκρουση).

Χειρισμός Συγκρούσεων (2)



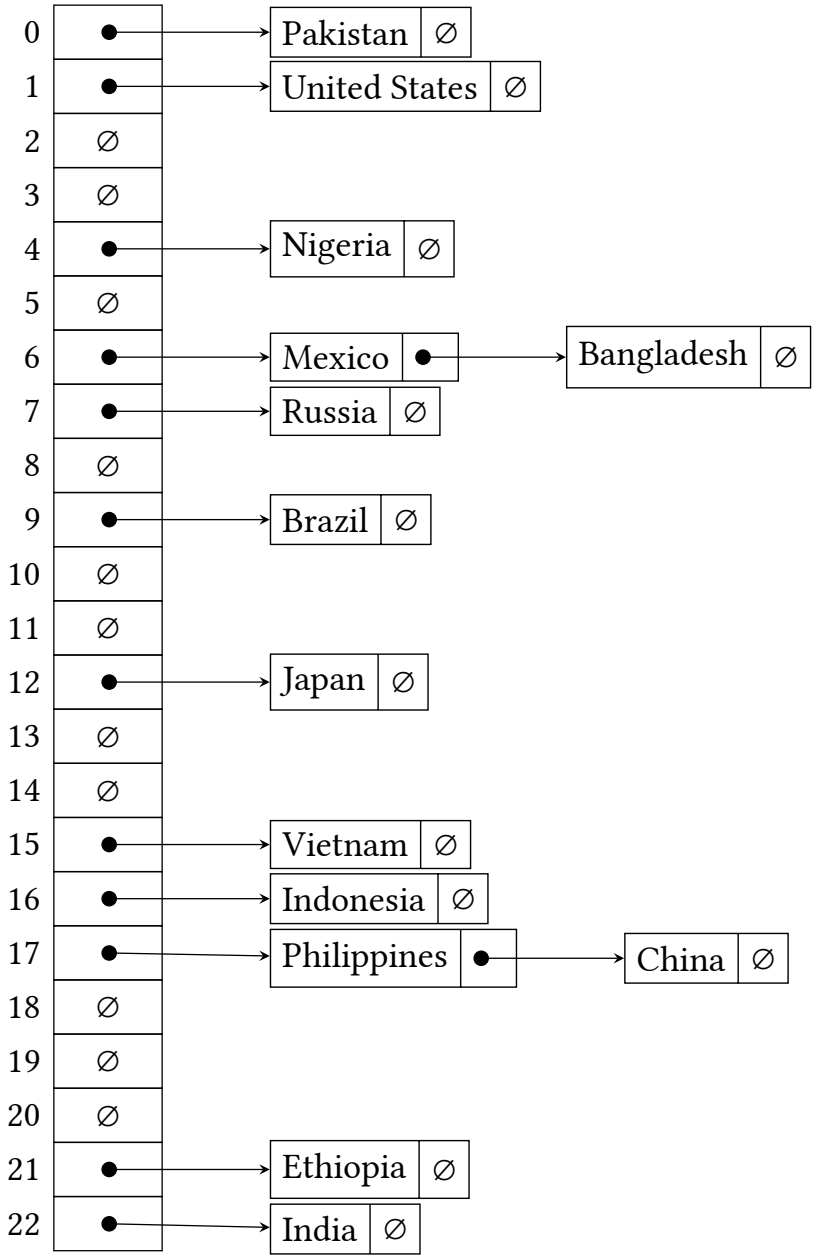
Εισαγωγή Philippines (σύγκρουση).

Χειρισμός Συγκρούσεων (3)



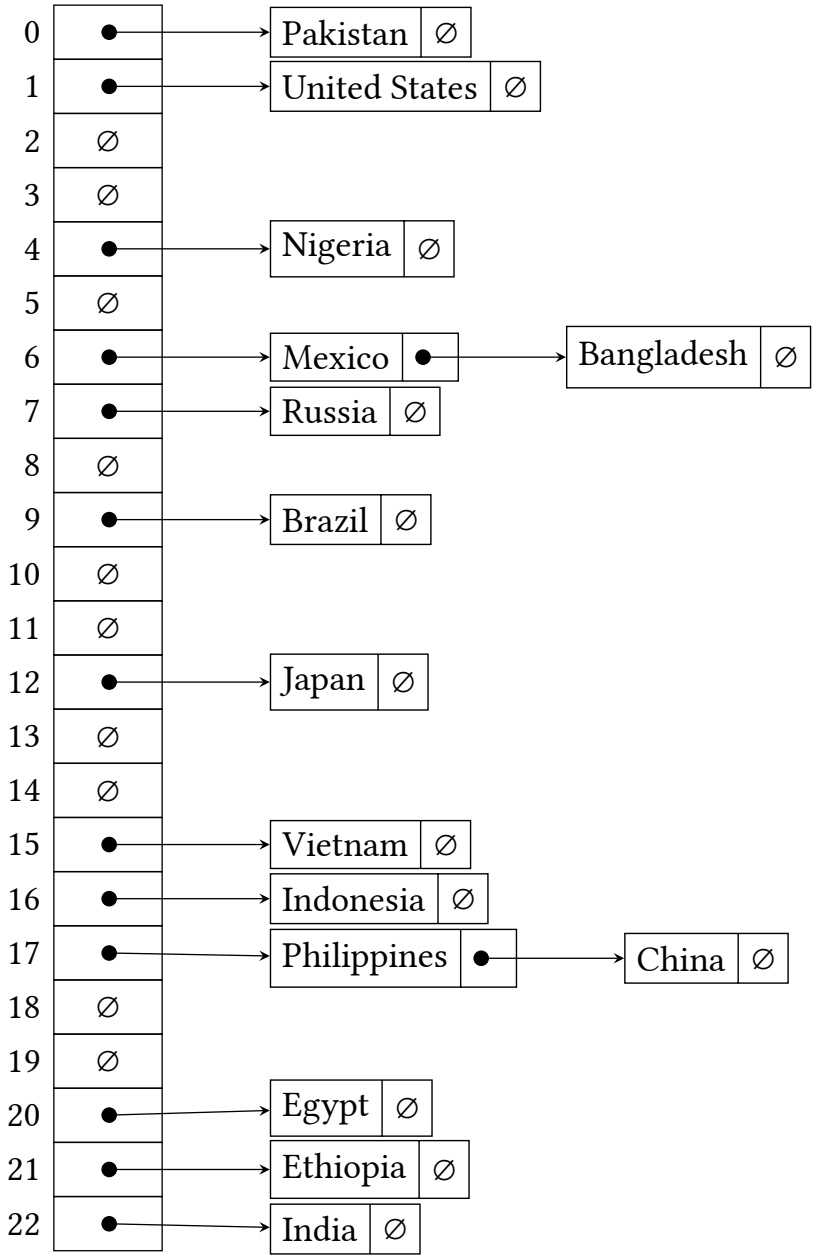
Εισαγωγή Vietnam.

Χειρισμός Συγκρούσεων (4)



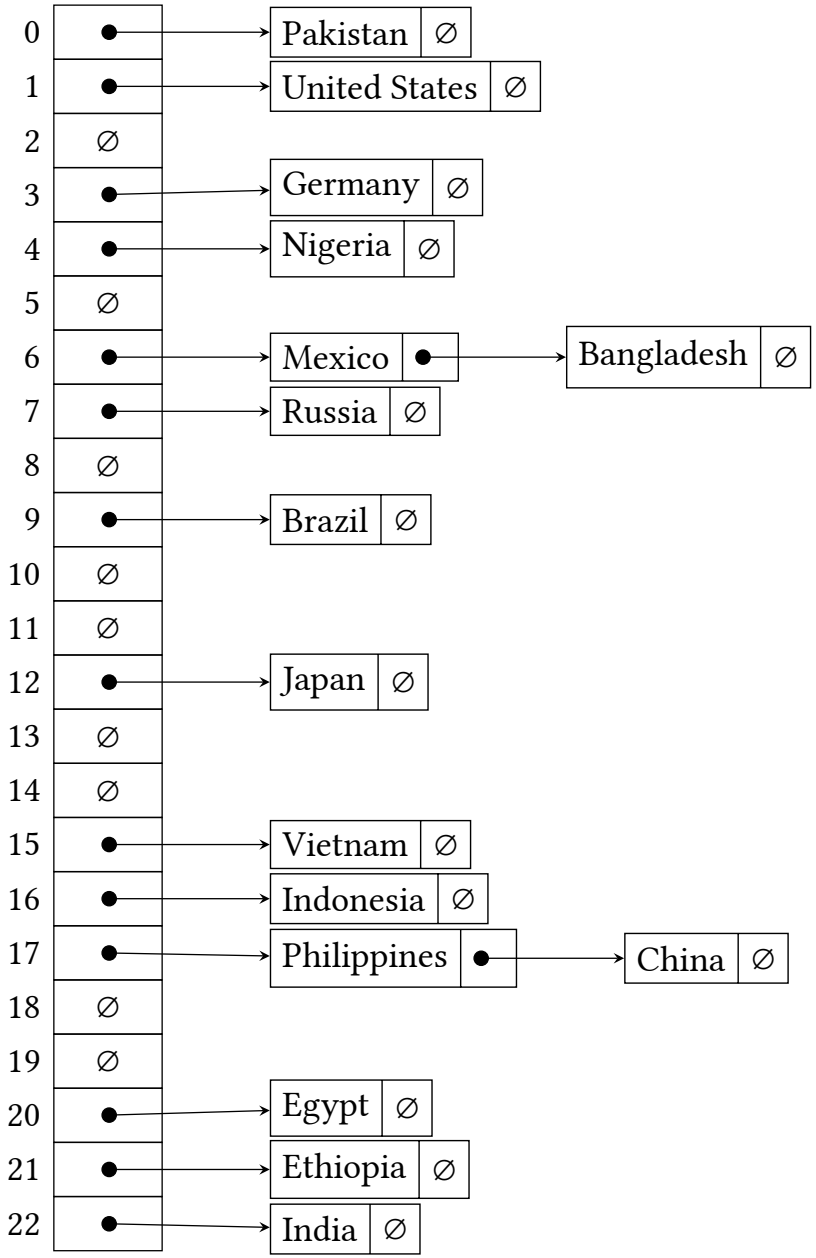
Εισαγωγή Ethiopia.

Χειρισμός Συγκρούσεων (5)



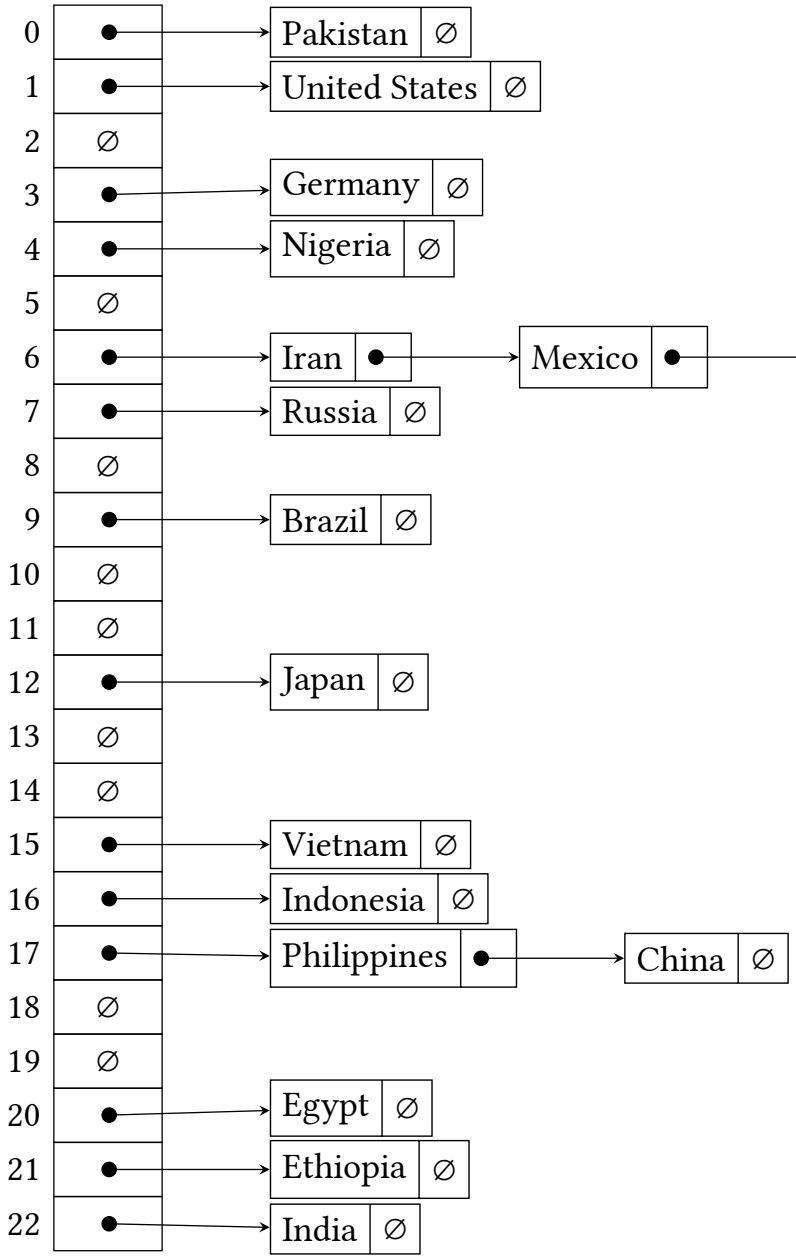
Εισαγωγή Egypt.

Χειρισμός Συγκρούσεων (6)



Εισαγωγή Germany.

Χειρισμός Συγκρούσεων (7)



Εισαγωγή Iran (σύγκρουση).

Αλγόριθμος Εισαγωγής σε Πίνακα Κατακερματισμού

Algorithm: Insertion in hash table with chained lists.

```
InsertInHash(T, x)
    Input: T, a hash table
           x, a record to insert in the hash table
    Result: x is inserted into T
1   h ← Hash(Key(x))
2   InsertInList(T[h], NULL, x)
```

Η συνάρτηση `InsertInList`(L, p, x) εισάγει έναν νέο κόμβο με δεδομένα x στη λίστα L μετά τον κόμβο p της λίστας. Αν το p είναι NULL, τον εισάγει στην αρχή της λίστας.

Αλγόριθμος Αναζήτησης σε Πίνακα Κατακερματισμού

Algorithm: Search in hash table with chained lists.

`SearchInHash`$(T, x) \longrightarrow$ TRUE or FALSE

Input: T, a hash table
x, a record to lookup in the hash table

Output: TRUE if found, FALSE otherwise

```
1  h ← Hash(Key(x))
2  if SearchInList(T[h], x) = NULL then
3      return FALSE
4  else
5      return TRUE
```

Η συνάρτηση `SearchInList`(L, x) αναζητά το στοιχείο x στη λίστα L και το επιστρέφει, αν το βρει, ή επιστρέφει NULL αν δεν το βρει.

Αλγόριθμος Διαγραφής από Πίνακα Κατακερματισμού

Algorithm: Removal from hash table with chained lists.

`RemoveFromHash`$(T, x) \longrightarrow x$ or NULL

Input: T, a hash table

x, a record to remove from the hash table

Output: x, the record x if it was removed, or NULL if x was not in the hash table

```
1  h ← Hash(Key(x))
2  return RemoveFromList(T[h], x)
```

Η συνάρτηση `RemoveFromList`(L, x) αφαιρεί το στοιχείο x από τη λίστα L και το επιστρέφει, αν το αφαίρεσε, ή επιστρέφει NULL αν δεν υπήρχε στη λίστα.

Απόδοση Αναζήτησης σε Πίνακα Κατακερματισμού

- Κατ' αρχήν πρέπει να υπολογίσουμε την τιμή κατακερματισμού.
- Αν τα κλειδιά μας είναι αριθμητικά, είναι το κόστος μιας διαίρεσης, που το θεωρούμε σταθερό, $O(1)$.
- Αν τα κλειδιά μας είναι συμβολοσειρές, το κόστος είναι $\Theta(n)$, όπου n είναι το μήκος της συμβολοσειράς. Επειδή η συνάρτηση κατακερματισμού είναι πολύ γρήγορη και εξαρτάται μόνο από το κλειδί και όχι τον αριθμό των εγγραφών, συνήθως το θεωρούμε και αυτό σταθερό.

Απόδοση Αναζήτησης σε Πίνακα Κατακερματισμού

- Σε αυτό πρέπει να προσθέσουμε το κόστος της αναζήτησης μετά τον υπολογισμό της συνάρτησης κατακερματισμού.
- Αν ο κάδος είναι άδειος, τότε είναι το κόστος μιας μόνο σύγκρισης και άρα $O(1)$.
- Αν όχι, το κόστος είναι $O(|L|)$, όπου $|L|$ είναι το μέγεθος της αλυσίδας.
- Άρα το ερώτημα είναι, πόσο μεγάλη μπορεί να είναι η αλυσίδα;

Απόδοση Αναζήτησης σε Πίνακα Κατακερματισμού

- Η απάντηση εξαρτάται από τη συνάρτηση κατακερματισμού.
- Αν είναι καλή, τότε το μήκος κάθε λίστας θα είναι περίπου n/m, όπου n είναι ο αριθμός των κλειδιών και m είναι το μέγεθος του πίνακα.
- Ο αριθμός n/m ονομάζεται *συντελεστής φόρτου* ή *παράγοντας φόρτου* (load factor) του πίνακα.
- Για μια ανεπιτυχή αναζήτηση χρειαζόμαστε τότε χρόνο $\Theta(n/m)$ για να φτάσουμε στο τέλος της λίστας.
- Για μια επιτυχημένη αναζήτηση αποδεικνύεται ότι ο χρόνος που χρειαζόμαστε είναι $\Theta(1 + (n-1)/2m)$.

Απόδοση Αναζήτησης σε Πίνακα Κατακερματισμού

- Και στις επιτυχείς και στις ανεπιτυχείς αναζητήσεις ο χρόνος που απαιτείται εξαρτάται από το n/m.
- Άρα αν ο αριθμός των κλειδιών είναι ανάλογος με το μέγεθος του πίνακα, δηλαδή $n = cm$, ο χρόνος αναζήτησης γίνεται σταθερός.
- Πράγματι, για ανεπιτυχή αναζήτηση έχουμε $\Theta(n/m) = \Theta(cm/m) = \Theta(c) = O(1)$. Η επιτυχής αναζήτηση σίγουρα χρειάζεται λιγότερο χρόνο, άρα πάλι έχουμε $O(1)$.

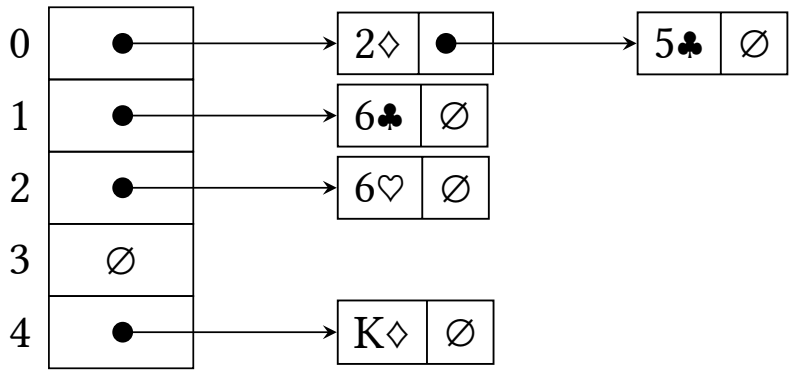
Απόδοση Αναζήτησης σε Πίνακα Κατακερματισμού

- Αφού αν $n = cm$ ο χρόνος αναζήτησης δεν μπορεί να είναι μεγαλύτερος από c, θέλουμε να εξασφαλίσουμε ότι το c παραμένει μικρό.
- Για παράδειγμα, αν $n = 2m$, οι αναζητήσεις δεν θα χρειαστούν πάνω από δύο συγκρίσεις.
- Αν λοιπόν ξέρουμε πόσα αντικείμενα θα εισάγουμε στον πίνακα, επιλέγουμε το μέγεθος του πίνακα κατάλληλα.

Αναπροσαρμογή Πίνακα Κατακερματισμού

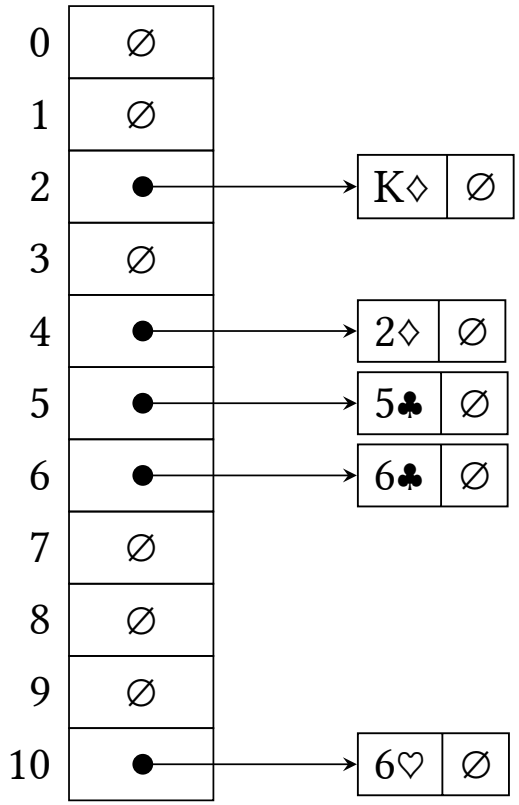
- Αν δεν ξέρουμε εκ των προτέρων πόσα αντικείμενα θα αποθηκεύσουμε, ή αν πέσουμε έξω στις προβλέψεις μας, μπορούμε να θέσουμε ένα όριο στον συντελεστή φόρτου.
- Όταν φτάσουμε το συγκεκριμένο όριο, τότε δημιουργούμε ένα νέο, μεγαλύτερο, πίνακα, για παράδειγμα διπλάσιο του προηγούμενου.
- Εισάγουμε όλα τα στοιχεία στον νέο, μεγαλύτερο πίνακα.
- Διαγράφουμε τον προηγούμενο πίνακα.
- Με τον τρόπο αυτό μπορούμε να εγγυηθούμε ότι οι αναζητήσεις κατά μέσο όρο δεν θα καθυστερούν.
- Οι εισαγωγές όμως θα αργούν στην περίπτωση που χρειάζεται η δημιουργία νέου πίνακα.

Παράδειγμα Αναπροσαρμογής (1)



Ο συντελεστής φόρτου είναι 1.

Παράδειγμα Αναπροσαρμογής (2)



Ο συντελεστής φόρτου είναι περίπου 0,45.

Χαρακτηριστικά Απόδοσης

- Η απόδοση των πινάκων κατακερματισμού είναι πιθανοθεωρητική.
- Κατά μέσο όρο περιμένουμε ότι το μέγεθος των λιστών δεν θα ξεπεράσει το m/n, αλλά αυτό μπορεί να συμβεί.
- Επιπλέον, η αναδιοργάνωση ενός πίνακα χρειάζεται χρόνο, όταν συμβαίνει (σπάνια).

Χαρακτηριστικά Απόδοσης

- Ταυτόχρονα, έχουμε τη δυνατότητα να ανταλλάσουμε (trade-off) χώρο με χρόνο.
- Αν το σημαντικότερο είναι ο χρόνος, τότε μπορούμε να χρησιμοποιήσουμε πολύ χώρο για να έχουμε χαμηλό συντελεστή φόρτου.
- Αν το σημαντικότερο είναι ο χώρος, σκεφτόμαστε αντιστρόφως και τοποθετούμε περισσότερα κλειδιά σε έναν πίνακα.

Επισκόπηση

Χρήσεις Πινάκων Κατακερματισμού

- Οι πίνακες κατακερματισμού είναι πολύ δημοφιλείς.
- Χρησιμοποιούνται σε πλήθος εφαρμογών.
- Είναι επίσης η βάση στην οποία υλοποιούνται δομές δεδομένων όπως *σύνολα* (sets) και *λεξικά* (dictionaries).

Υλοποίηση Συνόλων (Sets)

- Ένα σύνολο είναι μία δομή δεδομένων που περιέχει μοναδικά στοιχεία.
- Ένα σύνολο μπορεί να υλοποιηθεί απ΄ ευθείας με έναν πίνακα κατακερματισμού.
- Η εισαγωγή στοιχείου στο σύνολο αντιστοιχεί στην εισαγωγή στοιχείου στον πίνακα κατακερματισμού.
- Η διαγραφή στοιχείου από το σύνολο αντιστοιχεί στη διαγραφή στοιχείου από τον πίνακα κατακερματισμού.
- Ο έλεγχος αν ένα στοιχείο είναι μέλος ενός συνόλου αντιστοιχεί στην αναζήτηση ενός στοιχείου στον πίνακα κατακερματισμού.
- Τα σύνολα, όπως και στα μαθηματικά, δεν είναι ταξινομημένα.

Λεξικά

- Μια άλλη δομή δεδομένων που υλοποιείται με πίνακες κατακερματισμού είναι τα λεξικά.
- Επίσης ονομάζονται και *απεικονίσεις* (maps) ή *πίνακες αντιστοίχισης* (associative arrays).
- Ένα λεξικό περιέχει ζεύγη κλειδιών-τιμών (key-value pairs).
- Όταν αναζητούμε ένα κλειδί, μας επιστρέφει την αντίστοιχη τιμή (όπως όταν κοιτάζουμε τον ορισμό μιας λέξης σε ένα πραγματικό λεξικό).
- Μπορούμε επίσης να εισάγουμε ζεύγη, να αφαιρούμε ζεύγη, ή να αλλάζουμε την τιμή που αντιστοιχεί σε ένα
- Σε αντίθεση με τα παραδοσιακά λεξικά, τα λεξικά αυτά δεν είναι ταξινομημένα.

Αλγόριθμος Εισαγωγής σε Λεξικό

Algorithm: Insertion in a dictionary (map).

`InsertInMap`(T, k, v)

Input: T, a hash table
k, the key of the key-value pair
v, the value of the key-value pair

Result: value v is inserted into the dictionary associated with the key k

```
1  h ← Hash(k)
2  p ← SearchInListByKey(T[h], k)
3  if p = NULL then
4      p ← CreateArray(2)
5      p[0] ← k
6      p[1] ← v
7      InsertInList(T[h], NULL, p)
8  else
9      p[1] = v
```

Αλγόριθμος Αναζήτησης σε Λεξικό

Algorithm: Lookup in a dictionary (map).

Lookup$(T, k) \longrightarrow v$ or NULL

Input: T, a hash table

k, a key

Output: v, the corresponding value, if it exists, or NULL otherwise

```
1  h ← Hash(k)
2  p ← SearchInHash(T[h], k)
3  if p = NULL then
4      return NULL;
5  else
6      return p[1]
```

Αλγόριθμος Διαγραφής από Λεξικό

Algorithm: Removal from dictionary (map).

`RemoveFromMap`$(T, k) \rightarrow [k, v]$

Input: T, a hash table

k, a key to remove the corresponding key-value pair from the dictionary

Output: $[k, v]$, the key-value pair corresponding to k if it was removed, or NULL if no corresponding key-value pair was found in the dictionary

```
1  h ← Hash(k)
2  return RemoveFromListByKey(T[h], k)
```