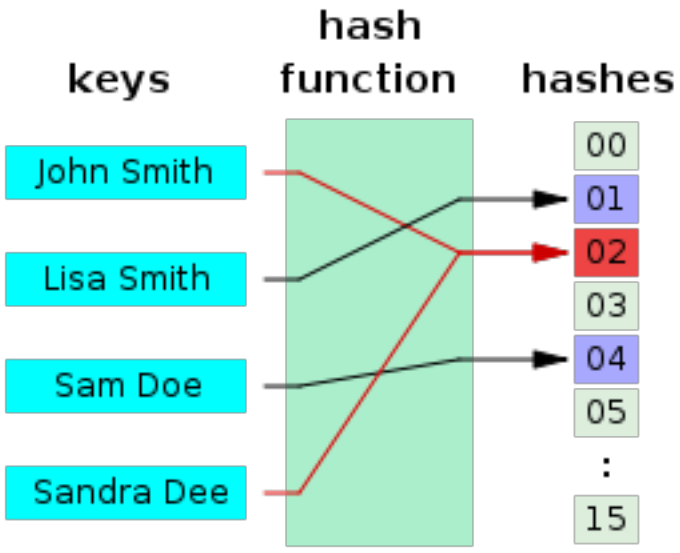
# Δομές Δεδομένων και Αλγόριθμοι - Εργαστήριο 7
# Κατακερματισμός

Τ.Ε.Ι. Ηπείρου, Τμήμα Μηχανικών Πληροφορικής Τ.Ε.
Χρήστος Γκόγκος - Αναπληρωτής Καθηγητής

## 1 Εισαγωγή

## 2 Τι είναι ο κατακερματισμός;

Ο κατακερματισμός (hashing) είναι μια μέθοδος που επιτυγχάνει ταχύτατη αποθήκευση και αναζήτηση δεδομένων. Σε ένα σύστημα κατακερματισμού τα δεδομένα αποθηκεύονται σε έναν πίνακα που ονομάζεται πίνακας κατακερματισμού (hash table). Εφαρμόζοντας στο κλειδί κάθε εγγραφής που πρόκειται να αποθηκευτεί ή να αναζητηθεί τη συνάρτηση κατακερματισμού (hash function) προσδιορίζεται μονοσήμαντα η θέση του πίνακα στην οποία τοποθετούνται τα δεδομένα της εγγραφής. Μια καλή συνάρτηση κατακερματισμού θα πρέπει να κατανέμει τα κλειδιά στα κελιά του πίνακα κατακερματισμού όσο πιο ομοιόμορφα γίνεται και να είναι εύκολο να υπολογιστεί.




Σχήμα 1:

Είναι επιθυμητό το παραγόμενο αποτέλεσμα από τη συνάρτηση κατακερματισμού να εξαρτάται από το κλειδί στο σύνολό του.

Οι πίνακες κατακερματισμού είναι ιδιαίτερα κατάλληλοι για εφαρμογές στις οποίες πραγματοποιούνται συχνές αναζητήσεις εγγραφών με δεδομένες τιμές κλειδιών. Ωστόσο, οι πίνακες κατακερματισμού έχουν και μειονεκτήματα καθώς είναι δύσκολο να επεκταθούν από τη στιγμή που έχουν δημιουργηθεί και μετά. Επίσης, η απόδοσή των πινάκων κατακερματισμού υποβαθμίζεται καθώς οι θέσεις τους γεμίζουν με στοιχεία. Συνεπώς, εφόσον ο προγραμματιστής προχωρήσει στη δική του υλοποίηση ενός πίνακα κατακερματισμού είτε θα πρέπει

να γνωρίζει εκ των προτέρων το πλήθος των στοιχείων που πρόκειται να αποθηκευτούν είτε όταν αυτό απαιτηθεί να υπάρχει πρόβλεψη έτσι ώστε τα δεδομένα να μεταφέρονται σε μεγαλύτερο πίνακα κατακερματισμού.

Στις περισσότερες εφαρμογές υπάρχουν πολύ περισσότερα πιθανά κλειδιά εγγραφών από ότι θέσεις στο πίνακα κατακερματισμού. Αν για δύο ή περισσότερα κλειδιά η εφαρμογή της συνάρτησης κατακερματισμού δίνει το ίδιο αποτέλεσμα τότε λέμε ότι συμβαίνει σύγκρουση (collision) η οποία θα πρέπει να διευθετηθεί με κάποιο τρόπο. Ειδικότερα, η εύρεση μιας εγγραφής με κλειδί k είναι μια διαδικασία δύο βημάτων:

- Εφαρμογή της συνάρτησης κατακερματισμού στο κλειδί της εγγραφής.
- Ξεκινώντας από την θέση που υποδεικνύει η συνάρτηση κατακερματισμού στον πίνακα κατακερματισμού, εντοπισμός της εγγραφής που περιέχει το ζητούμενο κλειδί (ενδεχόμενα θα χρειαστεί να εφαρμοστεί κάποιος μηχανισμός διευθέτησης συγκρούσεων).

### 2.1 Ανοικτή διευθυνσιοδότηση

### 2.2 Κατακερματισμός με αλυσίδες

## 3 Κατακερματισμός με την STL

## 4 Παραδείγματα

### 4.1 Παράδειγμα 1

Έστω μια επιχείρηση η οποία επιθυμεί να αποθηκεύσει τα στοιχεία των υπαλλήλων της (όνομα, διεύθυνση) σε μια δομή έτσι ώστε με βάση το όνομα του υπαλλήλου να επιτυγχάνει τη γρήγορη ανάκληση των υπόλοιπων στοιχείων των υπαλλήλων. Στη συνέχεια παρουσιάζεται η υλοποίηση ενός πίνακα κατακερματισμού στον οποίο κλειδί θεωρείται το όνομα του υπαλλήλου και η επίλυση των συγκρούσεων πραγματοποιείται με ανοικτή διευθυνσιοδότηση (open addressing) και γραμμική αναζήτηση (linear probing). Ο πίνακας κατακερματισμού μπορεί να δεχθεί το πολύ 10.000 εγγραφές υπαλλήλων. Στο παράδειγμα χρονομετρείται η εκτέλεση για 2.000, 3.000 και 8.000 υπαλλήλους. Παρατηρείται ότι λόγω των συγκρούσεων καθώς ο συντελεστής φόρτωσης του πίνακα κατακερματισμού αυξάνεται η απόδοση της δομής υποβαθμίζεται.

```
#include <random>
using namespace std;

mt19937 mt(1821);
uniform_int_distribution<int> uni(0, 25);

struct employee {
   string name;
   string address;
};

string generate_random_string(int k) {
   string s{};
   const string letters_en = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
   for (int i = 0; i < k; i++)
      s += letters_en[uni(mt)];
   return s;
}
```

Κώδικας 1: Ορισμός δομής employee και δημιουργία τυχαίων λεκτικών (employees.cpp)

```
#include "employees.cpp"
#include <chrono>
#include <iomanip>
#include <iostream>
```

```
#include <string>
#include <vector>

using namespace std::chrono;

const int N = 10000; // HashTable size

int simple_string_hash(string &s) {
  int h = 0;
  for (char c : s)
    h += c;
  return h % N;
}

void insert(employee hash_table[], employee &ypa) {
  int pos = simple_string_hash(ypa.name);
  while (hash_table[pos].name != "") {
    pos++;
    pos %= N;
  }
  hash_table[pos] = ypa;
}

bool search(employee hash_table[], string &name, employee &ypa) {
  int pos = simple_string_hash(name);
  int c = 0;
  while (hash_table[pos].name != name) {
    if (hash_table[pos].name == "")
      return false;
    pos++;
    pos %= N;
    c++;
    if (c > N)
      return false;
  }
  ypa = hash_table[pos];
  return true;
}

int main() {
  vector<int> SIZES{2000, 3000, 8000};
  for (int x : SIZES) {
    struct employee hash_table[N];
    // generate x random employees, insert them at the hashtable
    vector<string> names;
    for (int i = 0; i < x; i++) {
      employee ypa;
      ypa.name = generate_random_string(3);
      ypa.address = generate_random_string(20);
      insert(hash_table, ypa);
      names.push_back(ypa.name);
    }
    // generate x more names
    for (int i = 0; i < x; i++)
      names.push_back(generate_random_string(3));
    // time execution of 2*x searches in the HashTable
    auto t1 = high_resolution_clock::now();
    employee ypa;
    int c = 0;
    for (string name : names)
      if (search(hash_table, name, ypa)) {
```

```
66            // cout << "Employee " << ypa.name << " " << ypa.address << endl;
67            c++;
68          }
69        auto t2 = high_resolution_clock::now();
70    std:
71        chrono::duration<double, std::micro> duration = t2 - t1;
72        cout << "Load factor: " << setprecision(2) << (double)x / (double)N
73             << " employees found: " << c << " employees not found: " << 2 * x - c
74             << " time elapsed: " << std::fixed << duration.count()
75             << " microseconds" << endl;
76    }
77 }
```

Κώδικας 2: Υλοποίηση πίνακα κατακερματισμού για γρήγορη αποθήκευση και αναζήτηση εγγραφών (lab07_ex1.cpp)

```
1 Load factor: 0.2 employees found: 2000 employees not found: 2000 time elapsed: 141452.50 microseconds
2 Load factor: 0.30 employees found: 3000 employees not found: 3000 time elapsed: 316067.20 microseconds
3 Load factor: 0.80 employees found: 8000 employees not found: 8000 time elapsed: 2270363.00 microseconds
```

## 4.2 Παράδειγμα 2

```
1  #include "employees.cpp"
2  #include <chrono>
3  #include <iomanip>
4  #include <iostream>
5  #include <string>
6  #include <unordered_map>
7  #include <vector>
8
9  using namespace std::chrono;
10
11 int main() {
12   vector<int> SIZES{2000, 3000, 8000};
13   for (int x : SIZES) {
14     unordered_map<string, employee> umap;
15     // generate x random employees, insert them at the hashtable
16     vector<string> names;
17     for (int i = 0; i < x; i++) {
18       employee ypa;
19       ypa.name = generate_random_string(3);
20       ypa.address = generate_random_string(3);
21       umap[ypa.name] = ypa;
22       names.push_back(ypa.name);
23     }
24     // generate x more names
25     for (int i = 0; i < x; i++)
26       names.push_back(generate_random_string(10));
27
28     // time execution of 2*E searches in the HashTable
29     auto t1 = high_resolution_clock::now();
30     int c = 0;
31     for (string name : names)
32       if (umap.find(name) != umap.end()) {
33         // cout << "Employee " << name << " " << umap[name].address << endl;
34         c++;
35       }
36     auto t2 = high_resolution_clock::now();
37   std:
38     chrono::duration<double, std::micro> duration = t2 - t1;
```

```
39      cout << "Load factor: " << setprecision(2) << umap.load_factor()
40           << " employees found: " << c << " employees not found: " << 2 * x - c
41           << " time elapsed: " << std::fixed << duration.count()
42           << " microseconds" << endl;
43    }
44 }
```

Κώδικας 3: Γρήγορη αποθήκευση και αναζήτηση εγγραφών με τη χρήση της std::unordered_map (lab07_ex2.cpp)

```
1 Load factor: 0.53 employees found: 2000 employees not found: 2000 time elapsed: 0.00 microseconds
2 Load factor: 0.80 employees found: 3000 employees not found: 3000 time elapsed: 0.00 microseconds
3 Load factor: 0.53 employees found: 8000 employees not found: 8000 time elapsed: 0.00 microseconds
```

## 4.3 Παράδειγμα 3

Στο παράδειγμα αυτό εξετάζονται πέντε διαφορετικοί τρόποι με τους οποίους ελέγχεται για ένα μεγάλο πλήθος τιμών (5.000.000) πόσες από αυτές δεν περιέχονται σε ένα δεδομένο σύνολο 1.000 τιμών. Οι τιμές είναι ακέραιες και επιλέγονται με τυχαίο τρόπο στο δάστημα [0,100.000]. Ο χρόνος που απαιτεί η κάθε προσέγγιση χρονομετρείται. Η πρώτη προσέγγιση

```
1  #include <algorithm>
2  #include <bitset>
3  #include <chrono>
4  #include <iostream>
5  #include <random>
6  #include <set>
7  #include <unordered_set>
8  #include <vector>
9
10 using namespace std;
11 using namespace std::chrono;
12
13 // number of items in the set
14 constexpr int N = 1000;
15 // number of values checked whether they exist in the set
16 constexpr int M = 5E6;
17 // number of bits of the bitset
18 constexpr int BS_SIZE = 10001;
19 uniform_int_distribution<uint32_t> dist(0, 1E5);
20
21 void scenario1(vector<uint32_t> &avector) {
22   long seed = 1940;
23   mt19937 mt(seed);
24   int c = 0;
25   for (int i = 0; i < M; i++)
26     if (find(avector.begin(), avector.end(), dist(mt)) == avector.end())
27       c++;
28   cout << "Values not in the set (using unsorted vector): " << c << " ";
29 }
30
31 void scenario2(vector<uint32_t> &avector) {
32   sort(avector.begin(), avector.end());
33   long seed = 1940;
34   mt19937 mt(seed);
35   int c = 0;
36   for (int i = 0; i < M; i++)
37     if (!binary_search(avector.begin(), avector.end(), dist(mt)))
38       c++;
39   cout << "Values not in the set (using sorted vector): " << c << " ";
```

```
}

void scenario3(set<uint32_t> &aset) {
  long seed = 1940;
  mt19937 mt(seed);
  int c = 0;
  for (int i = 0; i < M; i++)
    if (aset.find(dist(mt)) == aset.end())
      c++;
  cout << "Values not in the set (using std::set): " << c << " ";
}

void scenario4(unordered_set<uint32_t> &auset) {
  long seed = 1940;
  mt19937 mt(seed);
  int c = 0;
  for (int i = 0; i < M; i++)
    if (auset.find(dist(mt)) == auset.end())
      c++;
  cout << "Values not in the set (using std::unordered_set): " << c << " ";
}

inline uint32_t hash1(uint32_t x) { return x; }
inline uint32_t hash2(uint32_t x) { return 3 * x; }
inline uint32_t hash3(uint32_t x) { return 5 * x + 7; }

void scenario5(bitset<BS_SIZE> &bs, unordered_set<uint32_t> &aset) {
  long seed = 1940;
  mt19937 mt(seed);
  int c = 0;
  for (int i = 0; i < M; i++) {
    uint32_t x = dist(mt), h1 = hash1(x), h2 = hash2(x), h3 = hash3(x);
    if (!bs.test(h1 % BS_SIZE) || !bs.test(h2 % BS_SIZE) ||
        !bs.test(h3 % BS_SIZE) || aset.find(x) == aset.end())
      c++;
  }
  cout << "Values not in the set (using bloom filter + std::unordered_set): "
       << c << " ";
}

int main() {
  long seed = 1821;
  mt19937 mt(seed);
  high_resolution_clock::time_point t1, t2;
  duration<double, std::micro> duration_micro;
  vector<uint32_t> avector(N);
  // fill vector with random values using std::generate and lambda function
  std::generate(avector.begin(), avector.end(), [&mt]() { return dist(mt); });

  t1 = high_resolution_clock::now();
  scenario1(avector);
  t2 = high_resolution_clock::now();
  duration_micro = t2 - t1;
  cout << "elapsed time: " << duration_micro.count() << " microseconds, "
       << duration_micro.count() / 1E6 << " seconds" << endl;

  t1 = high_resolution_clock::now();
  scenario2(avector);
  t2 = high_resolution_clock::now();
  duration_micro = t2 - t1;
  cout << "elapsed time: " << duration_micro.count() << " microseconds, "
```

```
101         << duration_micro.count() / 1E6 << " seconds" << endl;
102
103   set<uint32_t> aset(avector.begin(), avector.end());
104   t1 = high_resolution_clock::now();
105   scenario3(aset);
106   t2 = high_resolution_clock::now();
107   duration_micro = t2 - t1;
108   cout << "elapsed time: " << duration_micro.count() << " microseconds, "
109         << duration_micro.count() / 1E6 << " seconds" << endl;
110
111   unordered_set<uint32_t> auset(avector.begin(), avector.end());
112   t1 = high_resolution_clock::now();
113   scenario4(auset);
114   t2 = high_resolution_clock::now();
115   duration_micro = t2 - t1;
116   cout << "elapsed time: " << duration_micro.count() << " microseconds, "
117         << duration_micro.count() / 1E6 << " seconds" << endl;
118
119   bitset<BS_SIZE> bs;
120   for (int x : avector) {
121     bs.set(hash1(x) % BS_SIZE, true);
122     bs.set(hash2(x) % BS_SIZE, true);
123     bs.set(hash3(x) % BS_SIZE, true);
124   }
125   t1 = high_resolution_clock::now();
126   scenario5(bs, auset);
127   t2 = high_resolution_clock::now();
128   duration_micro = t2 - t1;
129   cout << "elapsed time: " << duration_micro.count() << " microseconds, "
130         << duration_micro.count() / 1E6 << " seconds" << endl;
131 }
```

Κώδικας 4: Έλεγχος ύπαρξης τιμών σε ένα σύνολο τιμών (lab07_ex3.cpp)

```
1 Values not in the set (using unsorted vector): 4994953 elapsed time: 1.9185e+007 microseconds, 19.185 seconds
2 Values not in the set (using sorted vector): 4994953 elapsed time: 1.02876e+006 microseconds, 1.02876 seconds
3 Values not in the set (using std::set): 4994953 elapsed time: 1.09391e+006 microseconds, 1.09391 seconds
4 Values not in the set (using std::unordered_set): 4994953 elapsed time: 598562 microseconds, 0.598562 seconds
5 Values not in the set (using bloom filter + std::unordered_set): 4994953 elapsed time: 297832 microseconds, 0.297832 seconds
```

# 5 Ασκήσεις

1. α
2. β

# Αναφορές