Хеширане

Хеширане е метод за директен достъп до елементите в таблица чрез аритметично преобразуване на ключове в адреси от таблицата.

Хеш таблица е множество от елементи, характеризиращи се чрез ключ, които са организирани въз основа на хеш кода на ключа.

Логическо описание

Съединение на елементи от тип T, образуващи тяло на таблицата.

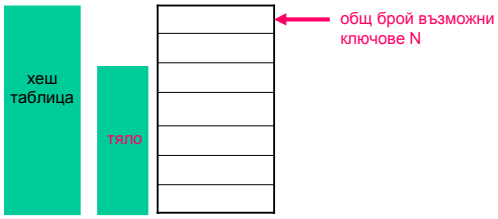
данни: тип T

тип ХЕШ_ТАБЛИЦА$_T$ =
(ПРАЗНО | НЕПРАЗНА_ХЕШ_ТАБЛИЦА$_T$)

тип НЕПРАЗНА_ХЕШ_ТАБЛИЦА$_T$ = (тяло: ХЕШ_ТАБЛИЦА$_T$)

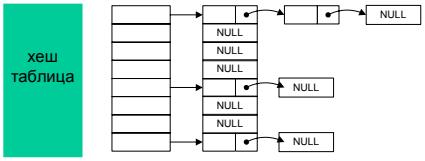
Физическо представяне

1. Непрекъснато представяне



тяло – **фиксирана област от паметта (масив)**
брой – **тип** ЦЯЛО

2. Верижно представяне



Непрекъснато представяне

Дефиниране

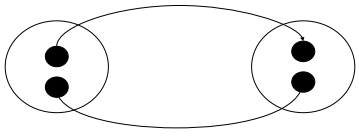
```
#define N <константа>          // брой ключове
typedef <тип> DATA;            // тип на данни
typedef DATA HASHTABLE[N];     // хеш таблица
```

Хеш функция

Хеш функция h – преобразува ключа k в адрес от таблицата i, 0≤i≤N-1, различен за различните ключове.

В идеалния случай запазваме елемента с ключ i в позиция i на таблицата.

Времето за търсене остава постоянно (O(1)).



Идеалната схема не може да се реализира:

1. **Ограничение на пространството с адреси – N е много голямо.**
2. **Колизия – два или повече ключа се хешират в един и същ адрес от таблицата.**

$$k_1 \neq k_2$$
$$h(k_1) = h(k_2)$$

Ако няма ограничение на паметта, ключът се използва като адрес на паметта.

Ако няма ограничение на времето, използва се минимално количество памет чрез използване на методи за последователно търсене.

Предимства на хеширането

1. **Ефективно използване на наличната памет**
2. **Бърз достъп до паметта**

Хеш функции $h(k)$

Хеш функцията h преобразува ключа k в число; трябва да има стойност между 0 и N-1.

Примери:

1. **Хеш функция за малки ключове – двоично представяне на ключовете като числа.**
2. **Хеш функция за дълги ключове:**

$$h(k) = k \bmod N$$

където N е просто число.

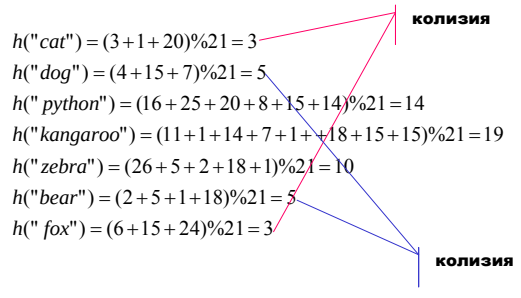
Пример: Хеш функция за низ (k е сума от стойностите на числовите кодове на символите) за ключовете:

cat dog python kangaroo zebra bear fox.

Нека N=21.

```
// Хеширане на низ
#define N 21                          // брой ключове
typedef char *DATA;                   // данни: ключ - низ
typedef DATA HASHTABLE[N];            // хеш таблица
```

Приемаме, че данните в хеш таблицата съдържат само ключ.



Разрешаване на колизиите

1. **Затворено хеширане – използва едно основно място за съхранение на данните.**
   - **Линейно пробване**
   - **Двойно хеширане**
2. **Отворено хеширане – използва допълнителна памет за разрешаване на колизиите.**
   - **Допълнителна памет за колизии**
   - **Списък на препълванията**

Линейно пробване

При наличие на свободно място в хеш таблицата (броят не елементите в таблицата n<N) и при наличие на колизия пробваме следващата позиция в таблицата.

1. Инициализиране на хеш таблица

Алгоритъм

за i от 0 до N-1
повтаряй
  таблица$_i$ ← **ПРАЗНО**

ХЕШ ТАБЛИЦА

```
void initialize (HASHTABLE t)
{
  int i;
  for (i=0; i<N; i++)
    t[i] = NULL;              // празно място в таблицата
}
```

2. Хеш функция

Алгоритъм

h ← **0**
докато не е достигнат последния символ на ключа
повтаряй
  h ← h **+ числов код на текущия символ**
h ← h **mod** N

ключ

```
int hash (DATA k)
{
  int h = 0;
  while (*k != '\0')
  {  h+ = *k - 'a' + 1;
     k++;
  }
  return h % N;
}
```

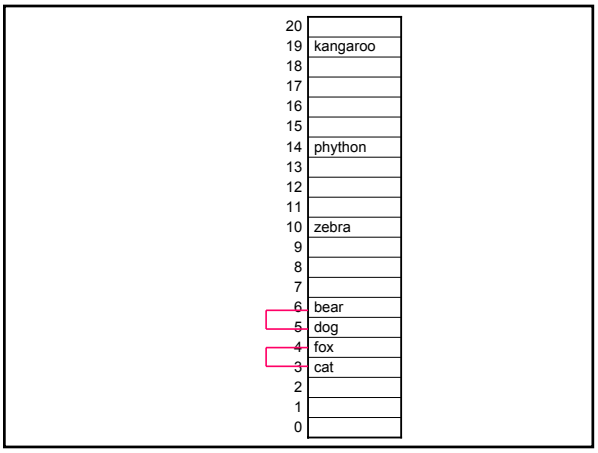
3. Включване на елемент в хеш таблица

Алгоритъм

i ← хеширан адрес в зависимост от ключа на елемента
брой_опити ← 0
докато таблица$_i$ **≠ ПРАЗНО**
повтаряй
  брой_опити ← брой опити + 1
  i ← (i + 1) mod N
ако брой_опити < N-1
  таблица$_i$ **← елемент**
в противен случай
  няма свободна позиция в таблицата

елемент ХЕШ ТАБЛИЦА

```
void insert (DATA x, HASHTABLE t)
{
  int i, number;
  i = hash(x);                // хеширан адрес
  number = 0;                 // брой опити
  while (t[i] != NULL)
  {
    number++;
    i = (i+1) % N;            // проба в следващата позиция
  }
  if (number < N - 1)
    t[i] = strdup (x);
  else
    printf("Грешка: Няма свободна позиция в таблицата!\n");
}
```



4. Търсене на елемент в хеш таблица

Алгоритъм

i ← хеширан адрес в зависимост от ключа на елемента
докато таблица$_i$ ≠ **ПРАЗНО и** таблица$_i$ ≠ **елемент**
повтаряй
  i ← (i + 1) mod N
ако таблица$_i$ = **ПРАЗНО**
  неуспешна операция
в противен случай
  намерен е адресът i

елемент — ХЕШ ТАБЛИЦА

```
int search (DATA x, HASHTABLE t)
{
  int i;
  i = hash (x);                 // хеширан адрес
  while (t[i] != NULL && strcmp (t[i], x) != 0)
    i = (i+1) % N;              // проба в следващата позиция
  if (!(t[i]))
    return -1;                  // неуспешна операция
  else
    return i;
}
```

Двойно хеширане

Стратегията на двойното хеширане при колизия е същата както при линейното пробване, но вместо да изпробваме следващата позиция в таблицата, използваме втора хеш функция, която изчислява нарастването.

$$h_1(k) = k \bmod N$$
$$h_2(k) = N - 2 - k \bmod (N-2)$$
$$h_2(k) = 8 - (k \bmod 8)$$
$$h_2(k) = k \bmod (N-2)$$

ключ

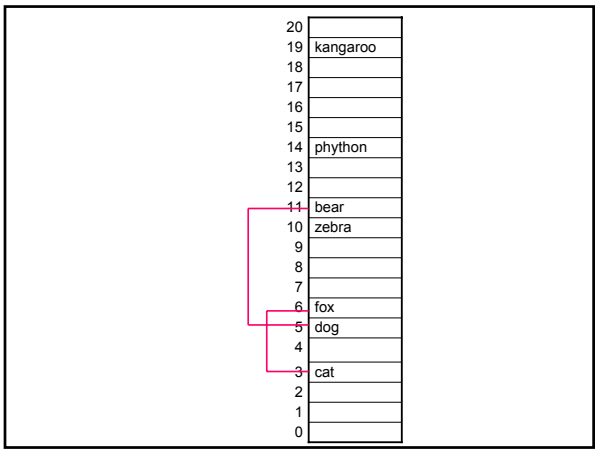
```
int hash (DATA k)           // преобразува ключ в число
{
  int h = 0;
  while (*k != '\0')
  {
    h+ = *k - 'a' + 1;
    k++;
  }
  return h;
}
```

елемент — ХЕШ ТАБЛИЦА

```
void insert (DATA x, HASHTABLE t)
{
  int i, number, u;
  i = hash (x) % N;                     // първа хеш функция
  u = 8 - (hash (x) % 8);               // втора хеш функция
  number = 0;                           // брой опити
  while (t[i] != NULL)
  {
    number++;
    i = (i+ u) % N;
  }
  if (number < N - 1)
    t[i] = strdup (x);
  else
    printf("Грешка: Няма свободна позиция в таблицата!\n");
}
```

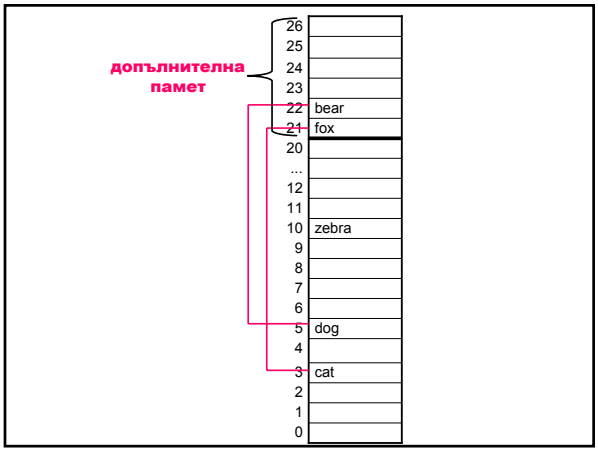
елемент — ХЕШ ТАБЛИЦА

```
int search (DATA x, HASHTABLE t)
{
  int i, number, u;
  i = hash (x) % N;                     // първа хеш функция
  u = 8-(hash(x) % 8);                  // втора хеш функция
  number = i;                           // начална позиция
  while (t[i] != NULL && strcmp (t[i], x) != 0)
  {
    i = (i+u) % N;
    if ( i== number)                    // ако търсенето започне от
      return -1;                        // началната позиция
  }
  if (!(t[i]))
    return -1;
  else
    return i;
}
```



Допълнителна памет за колизии

За обработка на колизиите се отделя допълнителна памет – всеки елемент, попаднал в колизия, се разполага в първото свободно място в допълнителната памет. При търсене първо се проверява хешираният адрес и ако търсеният елемент не е там, претърсва се цялата допълнителна памет.

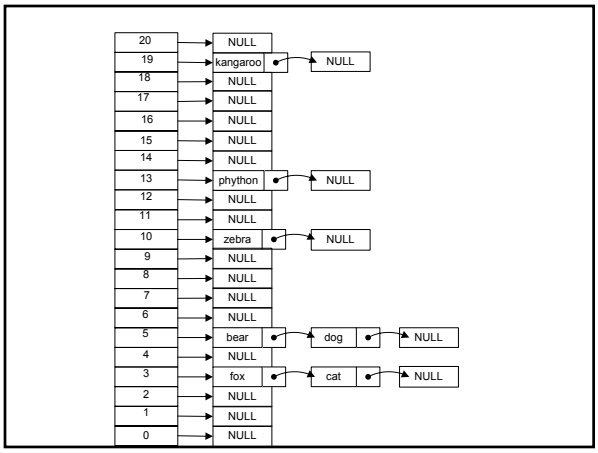


Списък на препълванията

За обработка на колизиите се отделя външна динамична памет – всеки елемент на хеш таблицата е указател към динамичен свързан списък, съдържащ елементите с еднакви хеш адреси.

Елемент се включва в началото на съответния списък – отпада необходимостта от разрешаване на колизии.

При търсене се проверява списъкът, определен от хеш функцията.




```
#define N 21                          // брой ключове
typedef char *DATA;                   // данни - низ
struct node                           // възел
{
  DATA data;                          // данни
  struct node *next;                  // връзка
};
typedef struct node NODE;
typedef NODE *LINK;
struct list                           // свързан списък
{
  LINK head;                          // начало
};
typedef struct list LIST;
typedef LIST HASHTABLE[N];            // хеш таблица
```

```
int hash (DATA k)
{                          ключ
  int h = 0;
  while (*k != '\0')
  {
    h += *k - 'a' + 1;
    k++;
  }
  return h % N;
}                                  ХЕШ ТАБЛИЦА
void initialize (HASHTABLE t)
{
  int i;
  for (i=0 ;i<N; i++)
    t[i].head = NULL;
}
```

```
void insert (DATA x, HASHTABLE t)
{                          елемент     ХЕШ ТАБЛИЦА
  int i;
  LINK p;
  i = hash (x);
  p = (LINK) malloc (sizeof (NODE));
  if (p == NULL)
  {
    printf("Няма памет.\n");
    return;
  }
  else
  {
    p->data = strdup (x);
    p->next = t[i].head;          // включване в началото
    t[i].head = p;                // на списъка
  }
}
```

```
    елемент                          ХЕШ ТАБЛИЦА
int search (DATA x, HASHTABLE t)
{
  int i;
  LINK p;
  i = hash (x);
  p = t[i].head;
  while (p != NULL && strcmp (p->data, x) != 0)
    p = p->next;
  if (!p)
    return -1;
  else
    return i;
}
```

Задача: **Абстрактен тип данни ХЕШ ТАБЛИЦА**

1. **Реализирайте хеш таблица чрез масив с елементи, които съдържат само ключ от тип символен низ. За разрешаване на колизиите използвайте двойно хеширане с хеш функции:**

$$h_1(k) = k \bmod N$$

$$h_2(k) = k \bmod (N-2)$$

2. **Напишете функция** print(), **която отпечатва хеш таблицата.**
3. **Отпечатайте хеш таблицата, като използвате функцията** print().

4. **Реализирайте хеш таблица чрез масив с елементи, които съдържат списъци на препълванията.**
5. **Напишете функция** print(), **която отпечатва хеш таблицата.**
6. **Отпечатайте хеш таблицата, като използвате функцията** print().