Системне програмування І

Пустоваров В. І., НТУУ”КПІ”,
м. Київ vipustovarov@ukr.net

Лекція 1/10
Організація хеш-пошуку як узагальнення вибірки за прямою адресою, вибір хеш-функцій та розв’язання колізій

1. *Узагальнення пошуку за прямою адресою, хеш-пошук та вибір хеш-функцій*
2. *Розв’язання колізій хеш-пошуку*
3. *Багатосегментні таблиці та індекси і вимоги унікальності ключів*

Передумови виникнення методів пошуку за хеш-функціями

- гранично висока швидкість пошуку за прямою адресою;
- необхідність надмірно великого адресного простору для збереження значень ключів при великих діапазонах припустимих значень;
- слабка заповнюваність навіть при великих обсягах оброблюваних текстів, що призводить до неефективного використання простору збереження значень ключів.

Вибірка за прямою адресою

// функція вибірки за прямою адресою на мові C/C++
struct recrd* selNmb(**struct** recrd* tb, **int** nElm)
{**return** &tb[nElm];
}
На мові Асемблера основна частина кодів вибірки за прямою адресою, розміщеною в [ebx], може мати вигляд:
XLAT ; вибірка байта за прямою адресою
або
MOV eax,tb[ebx*4]; вибірка подвійного слова за адресою
На базі цієї функції будуються функції зміни таблиць
При пошуку за прямою адресою ключова частина полів стає неістотною і може бути усунута з записів.

Використання хеш-функції як узагальнення пошуку за прямою адресою

Кожний елемент таблиці знаходиться за адресою $A = A_п + h(A_i)$, де Aп – початкова адреса сегменту таблиці, $h(Ai)$ – хеш-функція i–го ключа.

Вимоги до хеш-функції:

- давати детерміновані результати для кожного значення аргументу;
- обмеження значень у відповідності з розміром таблиці або її сегментів: $A_п + h(A_i) < A_{max}$;
- формувати якомога віддалені значення для близьких значень аргументів пошуку в таблиці;
- Забезпечити якомога рівномірний розподіл значень хеш-функції в області визначення.

Методи розрахунку хеш-функцій

- **метод залишку**, за яким $h(Ai) = Ai \bmod N$, де N – велике просте число, що відповідає розміру сегмента таблиці, наприклад, 1009.
- **лінійно-мультиплікативний метод**, за яким $h(A_i) = (\Sigma_i\, w_i\, E_{Ai}) \bmod N$,

  де w_i – вагові коефіцієнти, а E_{Ai} – елементи аргументу пошуку A.
- **метод виділення бітів**, за яким $h(Ai)$ формується зіпленням потрібної кількості бітів, взятих з визначених позицій ланцюжка бітів заданої функції;
- **метод фрагментації аргументу**, за яким бітовий рядок, що відповідає аргументу A, ділиться на фрагменти, що дорівнюють за довжиною хеш-адресі.

Вставки на мові Асемблера для розрахунку хеш-функцій

```
// hash-функція за формулою                    як вставка на Асемблері
unsigned hFunc(struct keyStr*pK, unsigned sgLen)
{char*s=pK->str;
 _asm{
      cld  ; визначення позитивного напрямку обробки рядка
      mov      esi, s      ; завантаження адреси початку ключа
      mov      ecx, 0ffffh      ; завантаження максимальної довжини
      sub edx, edx ; очистка регістру накопичення суми
l0:   lodsd     ; завантаження чергових чотирьох байтів
      add edx,eax  ; додавання складових hash-функції
      test al,al ; контроль закінчення рядка
      loopnz   l0     ; продовження циклу при продовженні рядка
      mov      eax, edx ; формування результату в акумуляторі
      sub edx, edx ; очистка регістру старших розрядів
      div  sgLen     ; одержання залишку
      mov      eax, edx ; формування результату в акумуляторі
 }
}
```

Структура елементів хеш-таблиць і хеш-індексів

Сегмент таблиці, впорядкованої за хеш-функцією	
Ключові поля	Функціональні поля
Значення ключів або ознаки відсутності “\0”...	Значення функцій ...

Сегмент індексу, впорядкованого за хеш-функцією	
Покажчики на ключі	Функціональні поля
Значення вказівників або ознаки відсутності 0 чи -1...	Значення функцій ...

Колізії та їх розв'язання

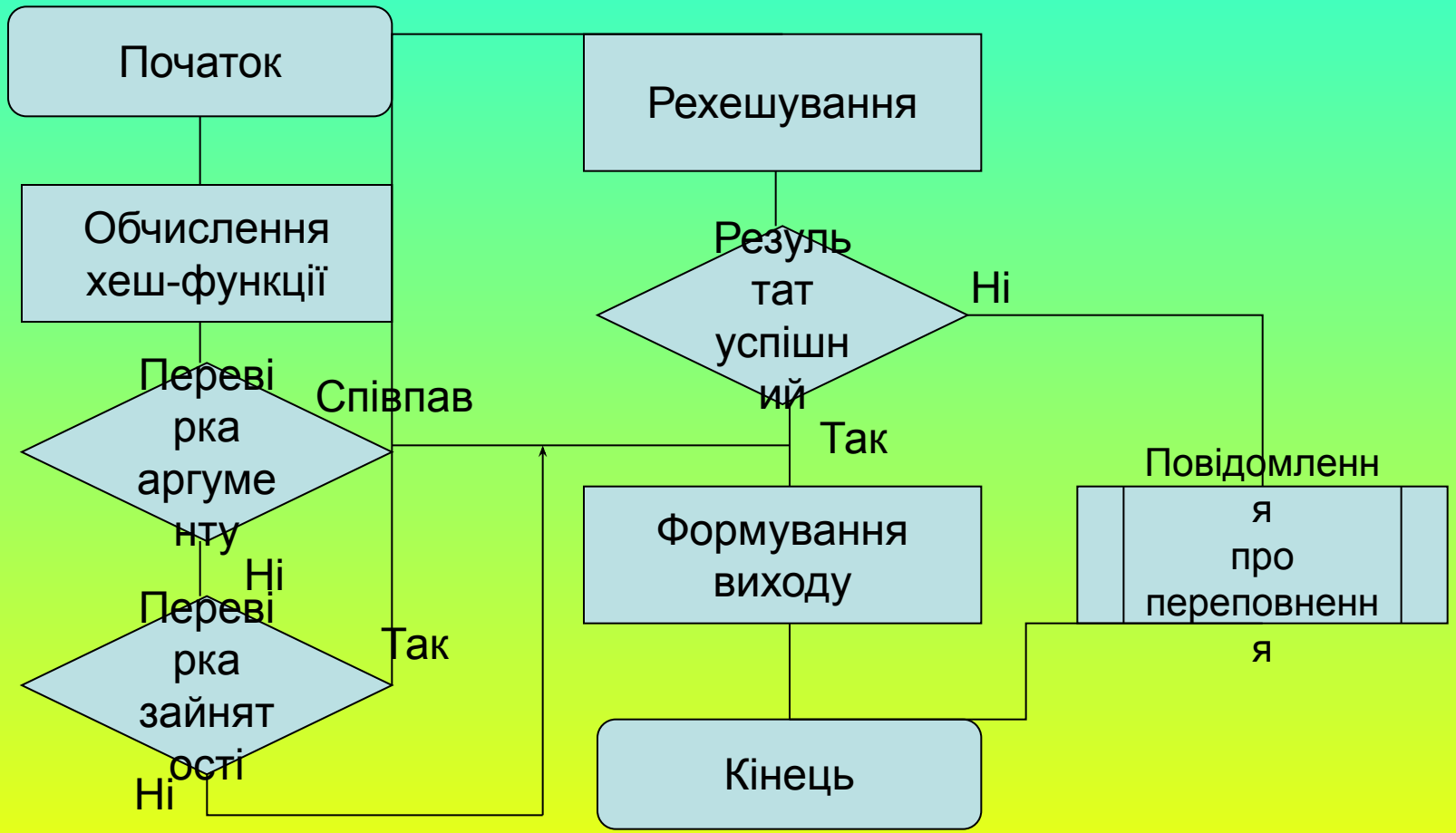
Умова виникнення колізій:

$h(A_j) = h(A_i)$, де $A_j \neq A_i$.

Базові способи розв'язання колізій:

- Рехешування лінійним пошуком вільного місця біля обчисленої хеш-адреси, найбільш відоме і найменш ефективне через можливе нагромадження колізійних ланцюжків
- функціональне рехешування за допомогою додаткової хеш-функції або функції довільного типа

Алгоритм хеш-пошуку з розв'язанням колізій



Приклад програми розв'язання колізій на мові C/C++ рехешування лінійним пошуком

```
// розв'язання колізій
unsigned hColRes
(struct recrd*pElm, // адреса елемента пошуку
 struct recrd*tb, // адреса початку сегмента таблиці
 unsigned h,      // значення первинної hash-функції
 unsigned sgLen) // довжина сегмента таблиці
{int l=2;          // кількість кроків розв'язання
 while(cmpKys(&(pElm->key), &(tb+h)->key)&&l--)
      h=h<sgLen?h+1:0; // пошук сусіднього елемента
 if(l<0)return l; // контроль досягнення граничного кроку
 return h;}
```

Визначення якості hash-функцій

Дослідження якості та ефективності hash-функцій спирається на визначенні критеріїв для:

- Статистики рівномірності розподілу значень в області припустимих значень;
- Часових інтервалів повторення значень хеш-функції при надходженні реальних даних аргументів пошуку;
- Часових інтервалів виникнення колізій;
- Часових інтервалів технічного заповнення таблиць.

БАГАТОСЕГМЕНТНІ ТАБЛИЦІ ТА ІНДЕКСИ

Багатосегментні таблиці є найбільш загальним варіантом побудови таблиць і індексів, які повинні обслуговувати вхідні дані системних програм різних обсягів. Якщо розглядати повні таблиці або індекси як автономні об'єкти, для них треба створювати управляючі структури даних, що поєднуватимуть такі автономні структури. В цьому випадку таблиці та структури даних, розглянуті в попередніх розділах можна вважати сегментами даних більш загальних багатосегментних таблиць. Структура, що визначає заголовок багатосегментної таблиці може бути записана наступним чином:

```
struct sgTbStr   // структура сегмента
{int nRsEl;    //    кількість зарезервованих елементів
  int nFlEl;    //    кількість використаних елементів
  struct sgTbStr*pNxtSg;//адреса наступного сегмента
  struct recrd* pRcPtr; //покажчик на сегмент записів
};
```

ВИМОГИ УНІКАЛЬНОСТІ КЛЮЧІВ В БАГАТОСЕГМЕНТНИХ ТАБЛИЦЯХ

Здебільшого таблиці в системних програмах працюють за вимоги унікальності ключів таблиць :

- Спосіб доступу повинен в більшості випадків давати можливість гарантії унікальності ключів, щоб запобігти можливим неоднозначностям при роботі з унікальними об’єктами;

Підсумки

- Використання хеш-функцій в багатьох випадках істотно підвищує швидкість пошуку в таблицях
- Хеш-функції дозволяють будувати індекси для швидкого доступу до даних і зв'язування таблиць
- Роботи з індексами дозволяє розділити роботу по заповненню і корекції таблиць на маніпуляції з рядками таблиць в довільному порядку та наступним підтриманням порядку в індексах