Алгоритмы и структуры данных

Лекция 6. Таблицы.

(c) Глухих Михаил Игоревич, glukhikh@mail.ru

Таблица

- Элемент = Ключ + Данные
- Ключи у разных элементов не совпадают
- Операции
  - Search (Key)
  - Insert (Key, Value)
  - Remove (Key)

Виды таблиц

- С прямой адресацией

Таблицы с прямой адресацией

- Ключ = Целое неотрицательное число или может быть к нему преобразован (взаимно-однозначно)
- Ключи не слишком большие

Таблицы с прямой адресацией

- Ключ = Целое неотрицательное число или может быть к нему преобразован (взаимно-однозначно)
- Ключи не слишком большие

- => Используем ключ как индекс массива, а данные храним в этом массиве
- Достоинства, Недостатки?

Таблицы с прямой адресацией

- Ключ = Целое неотрицательное число или может быть к нему преобразован (взаимно-однозначно)
- Ключи не слишком большие

- => Используем ключ как индекс массива, а данные храним в этом массиве
- Достоинства: трудоёмкость всюду O(1)
- Недостатки?

Таблицы с прямой адресацией

- Ключ = Целое неотрицательное число или может быть к нему преобразован (взаимно-однозначно)
- Ключи не слишком большие

- => Используем ключ как индекс массива, а данные храним в этом массиве
- Достоинства: трудоёмкость всюду O(1)
- Недостатки: ресурсоёмкость O(MaxKey)

Виды таблиц

- С прямой адресацией
- Хэш-таблицы
  - Используем для индексации hash(key) или hash(key) % 2^N

Виды таблиц

- С прямой адресацией
- Хэш-таблицы
  - Используем для индексации hash(key) или hash(key) % 2^N
    - В отличие от случая с прямой адресацией, здесь не требуется взаимной однозначности
  - Можем управлять размером нашего массива
  - НО! Коллизии

Виды таблиц

- С прямой адресацией
- Хэш-таблицы: как бороться с коллизиями?

Виды таблиц

- С прямой адресацией
- Хэш-таблицы: как бороться с коллизиями?
  - С помощью цепочек
  - Каждому значению кода ставится в соответствие не одна пара Ключ + Значение, а несколько, объединённых в связанный список
  - Достоинства = Размерность исходного массива ограничивается
  - Недостатки = ?

Виды таблиц

- С прямой адресацией
- Хэш-таблицы: как бороться с коллизиями?
  - С помощью цепочек
  - Каждому значению кода ставится в соответствие не одна пара Ключ + Значение, а несколько, объединённых в связанный список
  - Достоинства = Размерность исходного массива ограничивается
  - Недостатки = При фиксированной размерности массива трудоёмкость операций O(N) – в общем случае O(1 + N / m), где m – размерность массива
  - Как с этим бороться?

Виды таблиц

- С прямой адресацией
- Хэш-таблицы: как бороться с коллизиями?
  - С помощью цепочек
  - Каждому значению кода ставится в соответствие не одна пара Ключ + Значение, а несколько, объединённых в связанный список
  - Достоинства = Размерность исходного массива ограничивается
  - Недостатки = При фиксированной размерности массива трудоёмкость операций O(N) -- в общем случае O(1 + N / m), где m – размерность массива
  - Как с этим бороться? Понятно как – изменять размерность массива динамически

Таблица с открытой адресацией

- Нет никаких цепочек и связанных списков
- Всё хранится в едином массиве
- Если есть возможность, хэш-код (по модулю) используем как индекс в массиве
- Если возникает коллизия, то сдвигаем

HASH-INSERT(array[m], key)

- i = hash(key) % m
- start = i
- do:
  - if (array[i] == null):
    - array[i] = key
    - return i
  - i = (i + 1) % m
- while (i != start)
- Error(“Overflow”)

HASH-INSERT: пример

- Используем 3 бита хэша и массив из 8 элементов
- В качестве ключа используем символ

NN NN NN NN NN NN NN NN

HASH-INSERT: пример

- Используем 3 бита хэша и массив из 8 элементов
- В качестве ключа используем символ

```
NN  NN  NN  NN  NN  NN  NN  NN
NN  NN  2A  NN  NN  5B  NN  7C
```

HASH-INSERT: пример

- Используем 3 бита хэша и массив из 8 элементов
- В качестве ключа используем символ

```
NN NN NN NN NN NN NN NN
NN NN 2A NN NN 5B NN 7C

NN NN 2A 2D NN 5B NN 7C
```

HASH-SEARCH(array[m], key)

- i = hash(key) % m
- start = i
- do:
  - if (array[i] == key):
    - return i
  - if (array[i] == null):
    - return null
  - i = (i + 1) % m
- while (i != start)
- return null

HASH-DELETE

- Реализация «в лоб» не проходит

NN NN NN NN NN NN NN NN

NN NN 2A NN NN 5B NN 7C

NN NN 2A 2D NN 5B NN 7C

HASH-DELETE

- Реализация «в лоб» не проходит

```
NN NN NN NN NN NN NN NN
NN NN 2A NN NN 5B NN 7C
NN NN 2A 2D NN 5B NN 7C
// Теперь стираем 2A
NN NN NN 2D NN 5B NN 7C
```

HASH-DELETE

- Реализация «в лоб» не проходит

```
NN NN NN NN NN NN NN NN
NN NN 2A NN NN 5B NN 7C
NN NN 2A 2D NN 5B NN 7C
// Теперь стираем 2A
NN NN NN 2D NN 5B NN 7C
// И ищем 2D – NOT FOUND!!!
```

HASH-DELETE

- А как «не в лоб»? Отдельный вариант “DD” -- DELETED

```
NN NN NN NN NN NN NN NN
NN NN 2A NN NN 5B NN 7C
NN NN 2A 2D NN 5B NN 7C
// Теперь стираем 2A
NN NN DD 2D NN 5B NN 7C
```

HASH-DELETE

- А как «не в лоб»?
  Отдельный вариант “DD” – DELETED
- И при поиске DD не считается null

```
NN NN NN NN NN NN NN NN
NN NN 2A NN NN 5B NN 7C
NN NN 2A 2D NN 5B NN 7C
// Теперь стираем 2A
NN NN DD 2D NN 5B NN 7C
```

Таблица с открытой адресацией

- Трудоёмкость операций зависит от коэффициента заполнения A = n / m (n = число элементов, m = размерность масссива)

Таблица с открытой адресацией

- Трудоёмкость операций зависит от коэффициента заполнения A = n / m (n = число элементов, m = размерность масссива)
- И равна ~ 1 / (1 – A) для большинства операций

Таблица с открытой адресацией

- Трудоёмкость операций зависит от коэффициента заполнения A = n / m (n = число элементов, m = размерность масссива)
- И равна ~ 1 / (1 – A) для большинства операций

- Например, при таблице, заполненной на четверть, мы имеем трудоёмкость 4 / 3

Таблица с открытой адресацией

- Трудоёмкость операций зависит от коэффициента заполнения A = n / m (n = число элементов, m = размерность масссива)
- И равна ~ 1 / (1 – A) для большинства операций

- Например, при таблице, заполненной на четверть, мы имеем трудоёмкость 4 / 3
- А наполовину – уже 2

Таблица с открытой адресацией

- Трудоёмкость операций зависит от коэффициента заполнения $A = n / m$ (n = число элементов, m = размерность масссива)
- И равна ~ $1 / (1 - A)$ для большинства операций

- Например, при таблице, заполненной на четверть, мы имеем трудоёмкость $4 / 3$
- А наполовину – уже 2
- А на три четверти – уже 4
- И наконец, если заполнены все элементы, кроме одного, то требуется m операций

Таблица с открытой адресацией

- Последовательности проб: как мы переходим от ячейки к ячейке, если не нашли то, что нужно, в ячейке, соответствующей хэш-коду

Таблица с открытой адресацией

- Последовательности проб
  - H = hashCode % N, N = размер таблицы
  - Линейная: H, H+1, H+2, …, N-1, 0, 1, 2, …, H-1
    - Рассмотренный нами вариант

Таблица с открытой адресацией

- Последовательности проб
  - H = hashCode % N, N = размер таблицы
  - Линейная: H, H+1, H+2, …, N-1, 0, 1, 2, …, H-1
  - Двойное хэширование: H, H+s, H+2s, … (% N),
    N и s – взаимно простые
    (обычно достигается выбором простого N или $N=2^P$),
    s = anotherHashCode(key)

Итоги

- Рассмотрено
  - Таблицы и хэш-таблицы
    - На основе цепочек
    - С открытой адресацией
- Далее
  - Графовые алгоритмы