Курс «Алгоритмы и алгоритмические языки»

Лекция 14

Очередь

- Очередь (*queue*) – это линейный список информации, работа с которой происходит по принципу *FIFO*.

  Для списка можно использовать статический массив: количество элементов массива (*MAX*) = наибольшей допустимой длине очереди.

- Работа с очередью осуществляется с помощью **двух функций**:

  `qstore()` – *поместить* элемент в конец очереди;

  `qretrieve()` – *удалить* элемент из начала очереди;

  и **двух глобальных переменных**:

  `spos` (индекс первого свободного элемента очереди: его значение <= `MAX`)

  `rpos` (индекс очередного элемента, подлежащего удалению: «кто первый?»)

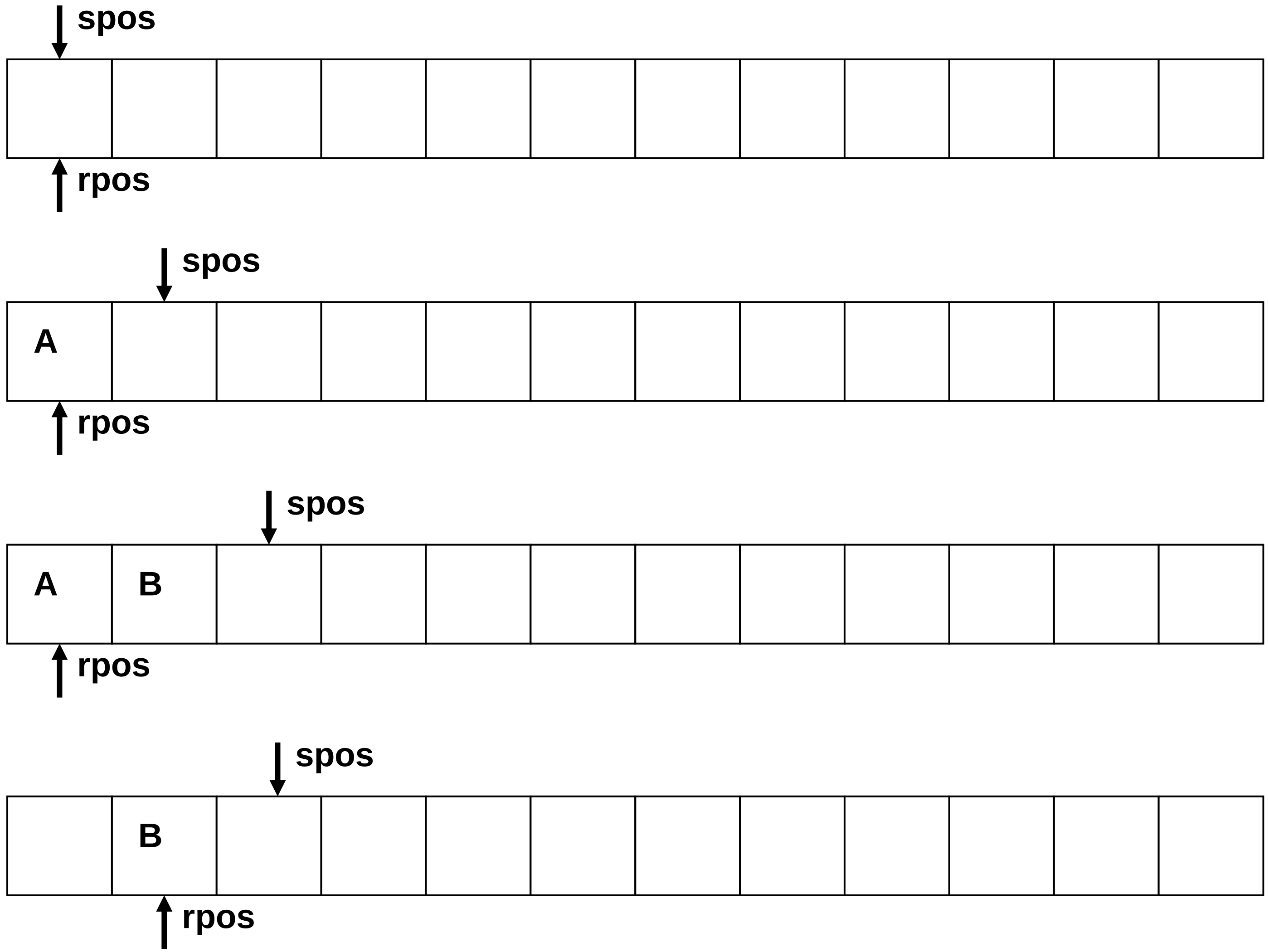
Очередь

Начальное состояние

`qstore('A')`

`qstore('B')`

`qretrieve()`



Очередь

◊ Тексты функций **qstore()** и **qretrieve()**

```
#define MAX    67
int queue[MAX];
int spos = 0, rpos = 0;

int qstore (int q) {
  if (spos == MAX) {
   /* Можно расширить очередь, см. реализацию стека */
    printf ("Очередь переполнена\n");
    return 0;
  }
  queue[spos++] = q;
  return 1;
}

int qretrieve (void) {
  if (rpos == spos) {
    printf ("Очередь пуста \n");
    return -1;
  }
  return queue[rpos++];
}
```

Улучшение – «зацикленная» очередь

```
enum { MAX = 67 };
int queue[MAX];
int spos = 0, rpos = 0;
```

```
int qstore (int q) {
  if (spos + 1 == rpos
      || (spos + 1 == MAX && !rpos) {
    printf ("Очередь переполнена \n");
    return 0;
  }
  queue[spos++] = q;
  if (spos == MAX)
    spos = 0;
  return 1;
}
```

Улучшение – «зацикленная» очередь

- ```
  int qretrieve (void) {
    if (rpos == spos) {
      printf ("Очередь пуста \n");
      return -1;
    }
    if (rpos == MAX - 1) {
      rpos = 0;
      return queue[MAX – 1];
    }
    return queue[rpos++];
  }
  ```
- Зацикленная очередь переполняется, когда **spos** находится непосредственно перед **rpos**, так как в этом случае запись приведет к **rpos == spos**, т.е. к пустой очереди.
  ```

Сортировка

- ***Постановка задачи***

  Сортировка – это упорядочение наборов однотипных данных, для которых определено отношение линейного порядка (например, <) по возрастанию или по убыванию.
  Здесь будут рассматриваться целочисленные данные и отношение порядка "<".

- В стандартную библиотеку **`stdlib`** входит функция **`qsort`**:

```
void qsort (void *buf, size_t num, size_t size,
int(*compare)(const void *, const void *));
```

  Функция **`qsort`** сортирует (по возрастанию) массив по указателю **`buf`**, используя алгоритм быстрой сортировки *Ч.Э.Р. Хоара*, который считается одним из лучших алгоритмов сортировки общего назначения.

  Параметр **`num`** задает количество элементов массива **`buf`**, параметр **`size`** – размер (в байтах) элемента массива **`buf`**.

  Параметр **`int(*compare)(const void *, const void *)`** задает правило сравнения элементов массива **`num`**.

Сортировка

- Функция, указатель на которую передается в **`qsort`** в качестве аргумента, соответствующего параметру **`int(*compare)(const void *, const void *)`**, должна возвращать:
  - целое < 0, если ***arg1*** < ***arg2***,
  - целое = 0, если ***arg1*** = ***arg2***
  - целое > 0, если ***arg1*** > ***arg2***

Сложность алгоритмов

- ◊ ***Размер входа***: числовая величина, характеризующая количество входных данных (например – длина битовой записи чисел-параметров алгоритма)
- ◊ ***Сложность в наихудшем случае***: функция размера входа, отражающая максимум затрат на выполнение алгоритма для данного размера
  - ♦ временнАя сложность
  - ♦ пространственная сложность (затраты памяти)
  - ♦ часто оценивают не все затраты, а только самые “дорогие” операции
- ◊ ***Сложность в среднем*** : функция размера входа, отражающая средние затраты на выполнение алгоритма для входа данного размера (учет вероятностей входа)
- ◊ ***Асимптотические оценки сложности***: O-нотация (оценка сверху), точная O-оценка, Θ-оценка.
- ◊ Подробности: С.А. Абрамов. Лекции о сложности алгоритмов. М.: МЦНМО, 2009

Сортировка

- ***Простейший алгоритм сортировки***: сведение сортировки к задаче нахождения максимального (минимального) из *n* чисел.

  Нахождение максимума *n* чисел (*n* сравнений):

  Числа содержатся в массиве **`int a[n];`**

```
max = a[0];
for (i = 1; i < n; i++)
  if (a[i] > max)
    max = a[i];
```

- ***Алгоритм сортировки***: находим максимальное из *n* чисел, получаем последний элемент отсортированного массива (*n* сравнений);
  находим максимальное из *n* – 1 оставшихся чисел, получаем предпоследний элемент отсортированного массива (еще *n* – 1 сравнений); и так далее.
- ***Общее количество сравнений***: 1 + 2 + … + *n*-1 + *n* = *n*(*n* - 1)/2.

  Сложность алгоритма $O(n^2)$.

◊ ***Классификация алгоритмов сортировки***

Различают *внешнюю* и *внутреннюю* сортировку.

Рассматривается только *внутренняя сортировка*: сортируемый массив находится в основной памяти компьютера. *Внешняя сортировка* применяется к записям на внешних файлах.

3 общих метода внутренней сортировки:

(1) *сортировка обменами*: рассматриваются соседние элементы сортируемого массива и при необходимости меняются местами;

(2) *сортировка выборкой*: идея описана на предыдущем слайде

(3) *сортировка вставками*: сначала сортируются два элемента массива, потом выбирается третий элемент и вставляется в нужную позицию относительно первых двух и т.д.

Сортировка

- ***Сортировка обменами (пузырьком)***

  Общее количество сравнений (действий): $n(n - 1)/2$, так как внешний цикл выполняется (n - 1) раз, а внутренний – в среднем n/2 раза.

```
void bubble_sort (int *a, int n) {
  int i, j, tmp;
  for (j = 1; j < n; ++j)
    for (i = n - 1; i >= j; --i) {
      if (a[i - 1] > a[i]) {
        tmp = a[i - 1];
        a[i - 1] = a[i];
        a[i] = tmp;
      }
    }
}
```

Сортировка

- ***Сортировка вставками***

  Количество сравнений зависит от степени перемешанности массива **a**. Если массив **a** уже отсортирован, количество сравнений равно n - 1. Если массив **a** отсортирован в обратном порядке (наихудший случай), количество сравнений имеет порядок n^2.

```
void insert_sort (int *a, int n) {
  int i, j, tmp;

  for (j = 1; j < n; ++j) {
    tmp = a[j];
    for (i = j - 1; i >= 0 && tmp < a[i]; i--)
      a[i + 1] = a[i];
    a[i + 1] = tmp;
  }
}
```

Сортировка

- ***Оценка сложности алгоритмов сортировки***
- Скорость сортировки определяется количеством сравнений и количеством обменов (обмены занимают больше времени). Эти показатели интересны для худшего и лучшего случаев, а также интересно их среднее значение.
- Кроме скорости оценивается «естественность» алгоритма сортировки: естественным считается алгоритм, который на уже отсортированном массиве работает минимальное время, а на не отсортированном работает тем дольше, чем больше степень его неупорядоченности.
- Важным показателем является и объем дополнительной памяти для хранения промежуточных данных алгоритма. Для рекурсивных алгоритмов расход памяти связан с необходимостью дублировать стек, в котором расположены некоторые промежуточные данные.

Сортировка

- ***Оценка сложности алгоритмов сортировки***.
- ***Теорема***. *Для любого алгоритма S внутренней сортировки сравнением массива из n элементов количество сравнений*

$$C_S \geq O(n \cdot \log_2(n))$$

- Доказательство.

(1) *Для любого алгоритма S внутренней сортировки сравнением массива из n элементов количество сравнений*

$$C_S \geq \log_2(n!).$$

(a) Алгоритм S можно представить в виде двоичного дерева сравнений.
Так как любая перестановка индексов рассматриваемого массива может быть ответом в алгоритме, она должна быть приписана хотя бы одному листу дерева сравнений.
Таким образом, дерево сравнений будет иметь не менее $n!$ листьев.

Сортировка

- ***Оценка сложности алгоритмов сортировки*.**
- ***Теорема***. *Для любого алгоритма S внутренней сортировки сравнением массива из n элементов количество сравнений*

$$C_S \geq O(n \cdot \log_2(n))$$

- Доказательство.

(1) *Для любого алгоритма S внутренней сортировки сравнением массива из n элементов количество сравнений*

$$C_S \geq \log_2(n!). \qquad (*)$$

(б) Для высоты h_m двоичного дерева с m листьями имеет место оценка:

$$h_m \geq \log_2 m.$$

Любое двоичное дерево высоты h можно достроить до полного двоичного дерева высоты h, а у полного двоичного дерева высоты h 2^h листьев.

Применив полученную оценку к дереву сравнений, получим оценку (*)

Сортировка

- ***Оценка сложности алгоритмов сортировки***.
- ***Теорема***. *Для любого алгоритма S внутренней сортировки сравнением массива из n элементов количество сравнений*

$$C_S \geq O(n \cdot \log_2(n))$$

- Доказательство.

(2) К $\log_2(n!)$ применим формулу Стирлинга

$$n! = \sqrt{2\pi n} \cdot n^n e^{-n} e^{\vartheta(n)}$$

$$|\vartheta(n)| \leq \frac{1}{12n} \qquad (**)$$

Логарифмируя (**), получаем

$$\log(n!) = \frac{1}{2}\log(2\pi n) + n \cdot \log(n) - n + \vartheta(n)$$

$$\log(n!) \geq O(n \cdot \log(n))$$