Разбор первых домашних заданий

очереди, деки, амортизационный анализ

Артем Оганджанян

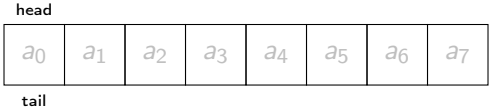
Q-Bit

21 апреля 2020 г.

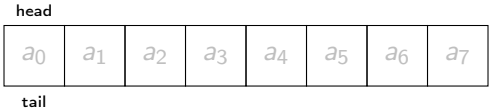
Оглавление

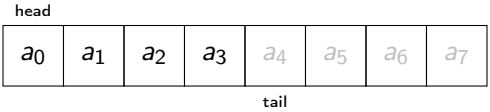
Оглавление

Алгоритм

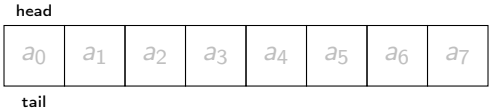


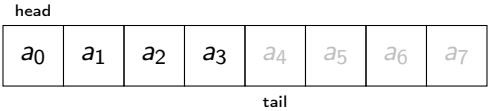
Алгоритм

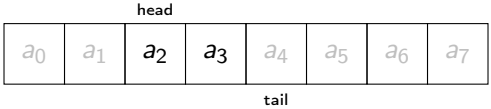




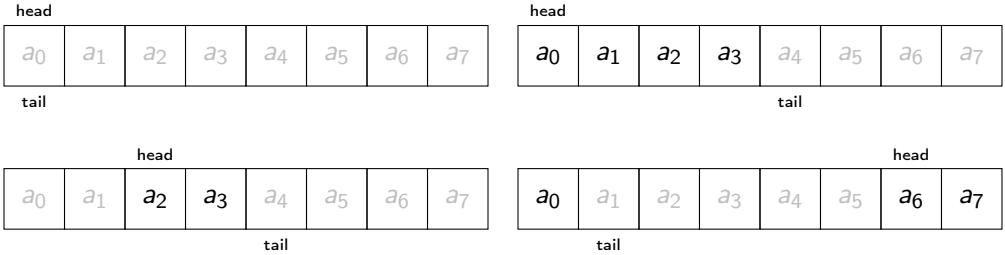
Алгоритм


head
a_0 a_1 a_2 a_3 a_4 a_5 a_6 a_7
tail


head
a_0 a_1 a_2 a_3 a_4 a_5 a_6 a_7
tail


head
a_0 a_1 a_2 a_3 a_4 a_5 a_6 a_7
tail

Алгоритм


head
a_0 a_1 a_2 a_3 a_4 a_5 a_6 a_7
tail
head
a_0 a_1 a_2 a_3 a_4 a_5 a_6 a_7
tail
head
a_0 a_1 a_2 a_3 a_4 a_5 a_6 a_7
tail
head
a_0 a_1 a_2 a_3 a_4 a_5 a_6 a_7
tail

Поля

Стек:

```
int capacity = DEFAULT_CAPACITY;
int *stack = new int[capacity];
int size_ = 0;
```

Поля

Стек:

```
int capacity = DEFAULT_CAPACITY;
int *stack = new int[capacity];
int size_ = 0;
```

Очередь:

```
int capacity = DEFAULT_CAPACITY;
int *queue = new int[capacity];
int head = 0, tail = 0;
```

int size()

Стек:

```
return size_;
```

int size()

Стек:

```
return size_;
```

Очередь:

```
return (tail - head + capacity) % capacity;
```

void change_capacity(int new_capacity)

Стек:

```
int *new_stack = new int[new_capacity];
for (int i = 0; i < size_; ++i) {
  new_stack[i] = stack[i];
}
delete[] stack;
stack = new_stack;
capacity = new_capacity;
```

void change_capacity(int new_capacity)

Очередь:

```
int *new_queue = new int[new_capacity];
if (tail >= head) {
  for (int i = head; i < tail; ++i) {
    new_queue[i - head] = queue[i];
  }
} else {
  for (int i = head; i < capacity; ++i) {
    new_queue[i - head] = queue[i];
  }
  for (int i = 0; i < tail; ++i) {
    new_queue[capacity - head + i] = queue[i];
  }
}
```

void change_capacity(int new_capacity)

Очередь:

```
tail = size();
head = 0;
delete[] queue;
queue = new_queue;
capacity = new_capacity;
```

void push(int value)

Стек:

```
ensure_capacity(size_ + 1);
stack[size_++] = value;
```

void push(int value)

Стек:

```
ensure_capacity(size_ + 1);
stack[size_++] = value;
```

Очередь:

```
ensure_capacity(size() + 2);
queue[tail] = value;
tail = (tail + 1) % capacity;
```

int pop()

Стек:

```
int result = stack[--size_];
ensure_capacity(size_);
return result;
```

int pop()

Стек:

```
int result = stack[--size_];
ensure_capacity(size_);
return result;
```

Очередь:

```
int result = queue[head];
head = (head + 1) % capacity;
ensure_capacity(size());
return result;
```

Оглавление

void push_front(int value)

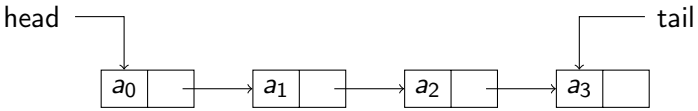
```
ensure_capacity(size() + 2);
head = (head - 1 + capacity) % capacity;
deque[head] = value;
```

int pop_back()

```
tail = (tail - 1 + capacity) % capacity;
int result = deque[tail];
ensure_capacity(size());
return result;
```

Оглавление

Односвязный список



Узлы

Стек:

```
struct stack_node {
  int element;
  stack_node *prev;
};
```

Узлы

Стек:

```
struct stack_node {
  int element;
  stack_node *prev;
};
```

Очередь:

```
struct queue_node {
  int element;
  queue_node *next;
};
```

Поля

Стек:

```
stack_node *top = NULL;
int size_ = 0;
```

Поля

Стек:

```
stack_node *top = NULL;
int size_ = 0;
```

Очередь:

```
queue_node *head = NULL, *tail = NULL;
int size_ = 0;
```

void push(int value)

Стек:

```
stack_node *new_top = new stack_node{value, top};
top = new_top;
++size_;
```

void push(int value)

Стек:

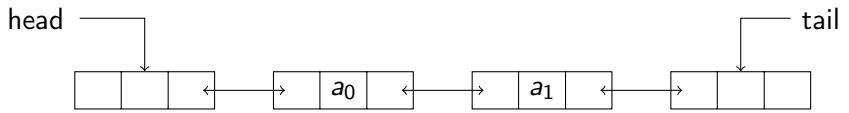
```
stack_node *new_top = new stack_node{value, top};
top = new_top;
++size_;
```

Очередь:

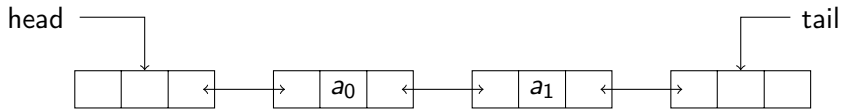
```
queue_node *new_tail = new queue_node{value, NULL};
if (tail != NULL) {
  tail->next = new_tail;
  tail = new_tail;
} else {
  head = tail = new_tail;
}
++size_;
```

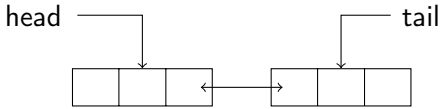
Оглавление

Двусвязный список



Двусвязный список





Узлы

```
struct deque_node {
  int element;
  deque_node *next;
  deque_node *prev;
};
```

Поля

```
deque_node *head = NULL, *tail = NULL;
int size_ = 0;

deque() {
  head = new deque_node{0, NULL, NULL};
  tail = new deque_node{0, NULL, NULL};
  head->next = tail;
  tail->prev = head;
}
```

void push(deque_node *next, int value)

```
deque_node *node = new deque_node{value, next, next->prev};
next->prev = node;
node->prev->next = node;
++size_;
```

int pop(deque_node *node)

```
auto [result, next, prev] = *node;
delete node;
next->prev = prev;
prev->next = next;
--size_;
return result;
```

push

```
void push_front(int value) {
  push(head->next, value);
}

void push_back(int value) {
  push(tail, value);
}
```

pop

```
int pop_front() {
  return pop(head->next);
}

int pop_back() {
  return pop(tail->prev);
}
```

front, back

```
int front() {
  return head->next->element;
}

int back() {
  return tail->prev->element;
}
```

Бонус

Список:

```
deque_node *head = NULL, *tail = NULL;
int size_ = 0;
```

Бонус

Список:

```
deque_node *head = NULL, *tail = NULL;
int size_ = 0;
```

Циклический список:

```
deque_node *fake = NULL;
int size_ = 0;
deque() {
  fake = new deque_node{0, NULL, NULL};
  fake->next = fake;
  fake->prev = fake;
}
```

Оглавление

Оглавление

Интуиция

capacity $= 2$

Интуиция

capacity $= 2$

$n = 3, \text{time} = 3 + 2 = 5$

Интуиция

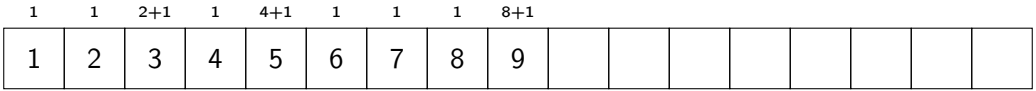
capacity $= 2$

$n = 3, \text{time} = 3 + 2 = 5$

$n = 5, \text{time} = 5 + 4 + 2 = 11$

Интуиция

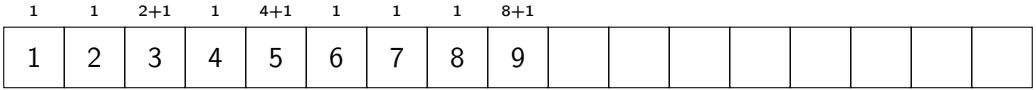
capacity $= 2$



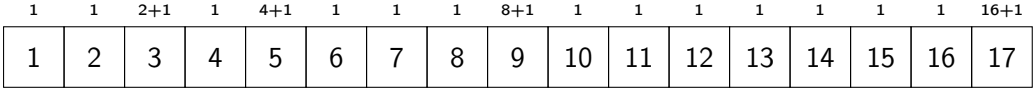
$n = 9, \text{time} = 9 + 8 + 4 + 2 = 23$

Интуиция

capacity $= 2$



$n = 9, \text{time} = 9 + 8 + 4 + 2 = 23$



$n = 17, \text{time} = 17 + 16 + 8 + 4 + 2 = 47$

Формула

$$n = 2^k + 1$$

Формула

$$n = 2^k + 1$$

$$S_n = \sum_{i=1}^{n} b_i = \frac{b_1(q^n - 1)}{q - 1}$$

Формула

$$n = 2^k + 1$$

$$S_n = \sum_{i=1}^{n} b_i = \frac{b_1(q^n - 1)}{q - 1}$$

$$\begin{aligned} \text{time}(n) &= n + \sum_{i=1}^{k} 2^i \\ &= n + 2(2^k - 1) \\ &= n + 2(n - 2) \\ &= 3n - 4 \\ &= O(n) \end{aligned}$$

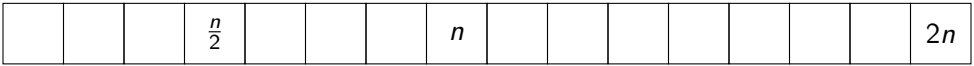
Оглавление

Доказательство

Рассмотрим операции между двумя изменениями размера массива.

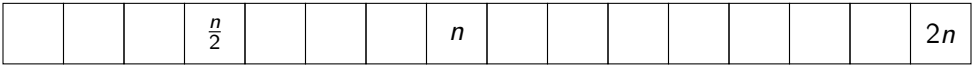
Доказательство

Рассмотрим операции между двумя изменениями размера массива.
n элементов с реальным размером массива $2n$.



Доказательство

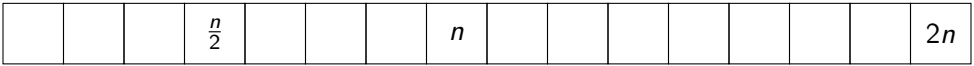
Рассмотрим операции между двумя изменениями размера массива.
n элементов с реальным размером массива $2n$.



Рассмотрим только операции удаления или только операции добавления.

Доказательство

Рассмотрим операции между двумя изменениями размера массива.
n элементов с реальным размером массива $2n$.

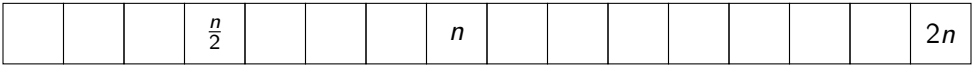


Рассмотрим только операции удаления или только операции добавления.

- Стало $\frac{n}{2}$.

Доказательство

Рассмотрим операции между двумя изменениями размера массива.
n элементов с реальным размером массива $2n$.

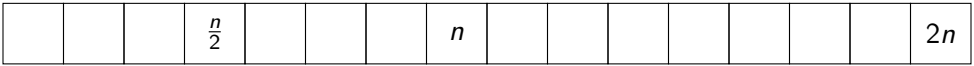


Рассмотрим только операции удаления или только операции добавления.

- Стало $\frac{n}{2}$. Потратили времени $\frac{n}{2} + \frac{n}{2} = n$

Доказательство

Рассмотрим операции между двумя изменениями размера массива.
n элементов с реальным размером массива $2n$.

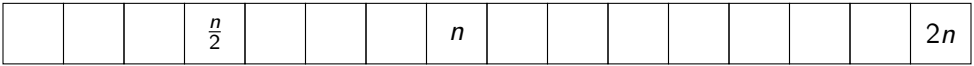


Рассмотрим только операции удаления или только операции добавления.

- Стало $\frac{n}{2}$. Потратили времени $\frac{n}{2} + \frac{n}{2} = n$
- Стало $2n$.

Доказательство

Рассмотрим операции между двумя изменениями размера массива.
n элементов с реальным размером массива $2n$.



Рассмотрим только операции удаления или только операции добавления.

- Стало $\frac{n}{2}$. Потратили времени $\frac{n}{2} + \frac{n}{2} = n$
- Стало $2n$. Потратили времени $n + 2n = 3n$.

Конец!

Вопросы?