Связные списки

1 Введение

Связный список - линейная структура данных, порядок которых задается не физическим положением в памяти, а при помощи указателей в самих элементах. Другими словами это структура данных, состоящая из узлов с указателями друг на друга.

Немного забегая вперед, рассмотрим время работы основных функций в сравнении с динамическими массивами (таб. 1). В отличие от массивов списки не предоставляют произвольный доступ к элементам. По этой причине индексация и вставка в произвольное место работают за линию. Работа с концом списка зависит от наличия указателя на конец. Еще одно отличие списков – возможность быстрой вставки и удаления с начала списка, что важно в реализации некоторых структур данных.

Таблица 1: Сравнение времени работы основных операций.

Операция	**Св. списки**	**Дин. массивы**
Индексация	$O(n)$	$O(1)$
Добавление/удаление с начала	$O(1)$	$O(n)$
Добавление/удаление с конца	$O(1)$ or $O(n)$	$O(1)$ аморт.
Добавление/удаление в середине	$O(n)$	$O(n)$

Связные списки обычно используются для реализации:

- стека,
- очереди,
- дека,
- ассоциативных массивов.

В зависимости от использования, связные списки реализуют по разному. Рассмотрим основные реализации.

Примечание: при написании кода мы будем использовать словари для представления узлов. Для создания нового узла с заданным значением будем использовать функцию get_node().

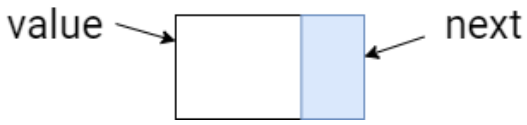

Рис. 1: Структура узла односвязного списка

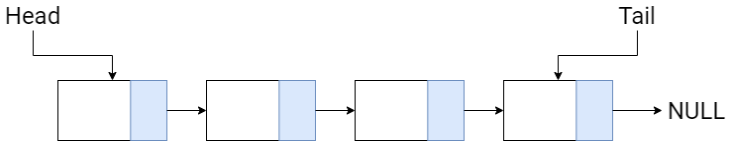

Рис. 2: Односвязный список

2 Односвязный список

Односвязный список – это связный список, где каждый узел содержит один указатель – указатель на следующий узел в списке (рис. 1). Последний узел списка всегда указывает на `NULL`. Общий вид односвязного списка представлен на рисунке 2. Обратите внимание, что есть два указателя – `head` и `tail`. Первый указывает на начало списка, второй – на конец. В некоторых задачах `tail` не используется и его можно не поддерживать. Рассмотрим написание односвязного списка в контексте реализации стека и очереди.

2.1 Стек

Стек – абстрактная структура данных, которая работает по принципу LIFO (last in, first out; последним пришел – первым ушел), т.е. позволяет проводить операции только с верхним элементом. Стек поддерживает следующие опреации:

- push – добавить в стек,
- pop – удалить из стека,
- peek – просмотреть верхний элемент,
- length – размер стека.

Для реализации стека достаточно использовать динамический массив или односвязный список без указателя на конец. Для определении размера стека на списках надо просто поддреживать дополнительную переменную, хранящую текущий размер стека.

```
def get_node(v):
    return {
```

```
        "value": v,
        "next": None
    }


def enqueue(head, tail, value):
    tmp = get_node(value)
    if head is None:
        return tmp, tmp
    tail["next"] = tmp
    return head, tmp


def dequeue(head, tail):
    if head is None:
        print("Queue is empty")
        return None, None, None
    if id(head) == id(tail):
        return None, None, head["value"]
    return head["next"], tail, head["value"]


def peek(head):
    if head is None:
        print("Queue is empty")
        return None
    return head["value"]
```

2.2 Очередь

Очередь – абстрактная структура данных, которая работает по принципу FIFO (first in, first out; первым пришел – первым ушел), т.е. первым будет удален тот элемент, который был первым добавлен в очередь. Очередь поддерживает следующие операции:

- enqueue – добавить в очередь,
- dequeue – удалить из очереди,
- peek – показать первый элемент,
- length – размер очереди.

Для реализации очереди достаточно использовать односвязный список с указателем на конец.

```
def get_node(v):
    return {
```

```
        "value": v,
        "next": None
    }


def push(head, value):
    tmp = get_node(value)
    tmp["next"] = head
    return tmp


def pop(head):
    if head is None:
        print("Stack is empty")
        return None, None
    return head["next"], head["value"]


def peek(head):
    if head is None:
        print("Stack is empty")
        return None
    return head["value"]
```

3 Двусвязный список

Двусвязный список – это связный список, где каждый узел содержит два указателя – указатель на следующий узел и указатель на предыдущий узел (рис. 3). Последний и первый узлы списка указывают на `NULL`. Общий вид двусвязного списка представлен на рисунке 4.

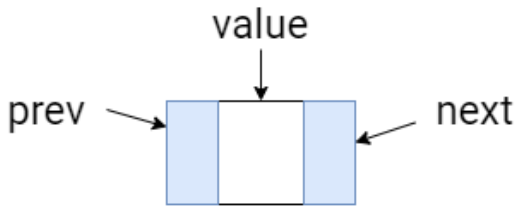

Рис. 3: Структура узла двусвязного списка

Двусвязные списки могут быть использованы для написания стека и очереди, но это является не оптимальным решением, т.к. в этом случае наличие обратной ссылки является избыточной тратой памяти. Однако для дека это необходимо.

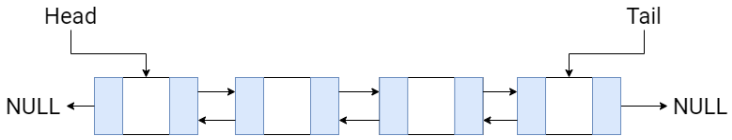

Рис. 4: Двусвязный список

3.1 Дек

Дек или **двунаправленная очередь** – абстрактная структура данных, которая является обобщением очереди. Он предоставляет доступ к элементам с обоих концов. Дек поддерживает следующие опреации:

- push front/back – добавить в начало/конец дека,
- pop front/back – удалить из начала/конца дека,
- peek front/back – просмотреть первый/последний элемент,
- length – размер дека.

Для реализации дека нужен двусвязный список с указателем на конец.

```
def get_node(v):
    return {
        "value": v,
        "next": None,
        "prev": None
    }


def push_front(head, tail, value):
    tmp = get_node(value)
    if head is None:
        return tmp, tmp
    tmp["next"] = head
    head["prev"] = tmp
    return tmp, tail


def push_back(head, tail, value):
    tmp = get_node(value)
    if head is None:
        return tmp, tmp
    tmp["prev"] = tail
    tail["next"] = tmp
```

```
    return head, tmp


def pop_front(head, tail):
    if head is None:
        print("Deque is empty")
        return None, None, None
    if id(head) == id(tail):
        return None, None, head["value"]
    head["next"]["prev"] = None
    return head["next"], tail, head["value"]


def pop_back(head, tail):
    if head is None:
        print("Deque is empty")
        return None, None, None
    if id(head) == id(tail):
        return None, None, head["value"]
    tail["prev"]["next"] = None
    return head, tail["prev"], tail["value"]


def peek_front(head):
    if head is None:
        print("Deque is empty")
        return None
    return head["value"]


def peek_back(tail):
    if tail is None:
        print("Deque is empty")
        return None
    return tail["value"]
```

4 Кольцевой список

Кольцевой список – это связный список, где последний элемент связан с первым. Фактически, у такого списка нет конца, а начало определяется только указателем `head`. Кольцевой список может быть представлен как односвязным (рис. 5), так и двусвязным (рис. 6) списком.

Реализация функций в кольцевом списке не сильно отличается от его нециклических аналогов. Сами по себе кольцевые списки применяются редко. Например, для поддержания структуры многоугольника, когда

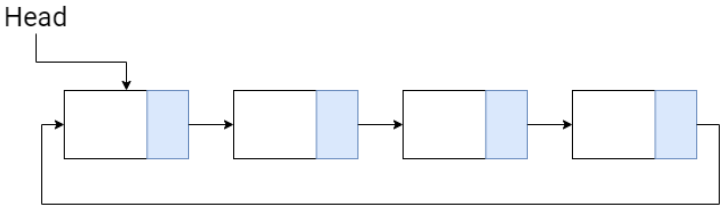

Рис. 5: Кольцевой список на основе односвязного спика

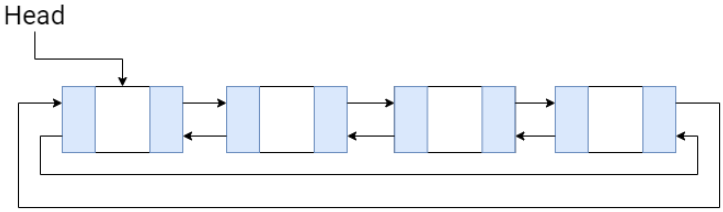

Рис. 6: Кольцевой список на основе двусвязного списка

важно знать порядок вершин в нем.