Лекция 8. Связанные списки

Казахский национальный университет имени аль-Фараби

Кафедра компьютерных наук

Алгоритмы и структуры данных

Лектор: старший преподаватель, PhD, Кенжебек Ержан Ғалымжанұлы

Цель и план лекции

Цель лекции

Изучить структуру связанного списка, его виды и основные операции — поиск, вставку и удаление, а также научиться оценивать их сложность.

План лекции

1. Понятие связного списка, указатели next и prev
2. Формы списков
3. Поиск в связанном списке
4. Вставка элемента
5. Удаление элемента
6. Практика: реализация на C++

Связный список

- Связный список — это структура данных, в которой объекты расположены в линейном порядке. Однако в отличие от массива, в котором линейный порядок определяется индексами массива, порядок в связанном списке определяется указателем в каждом объекте.
- L — это объект с ключом атрибута и двумя другими атрибутами-указателями: next и pre. Объект также может содержать другие спутниковые данные.

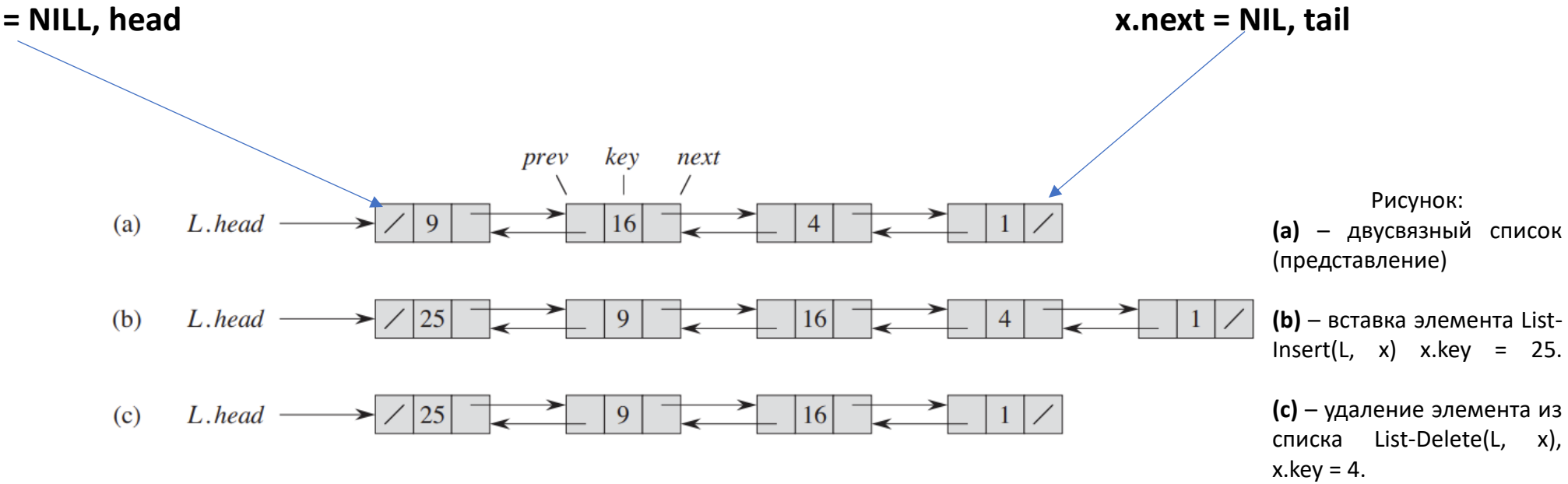


Рисунок:
(a) – двусвязный список (представление)

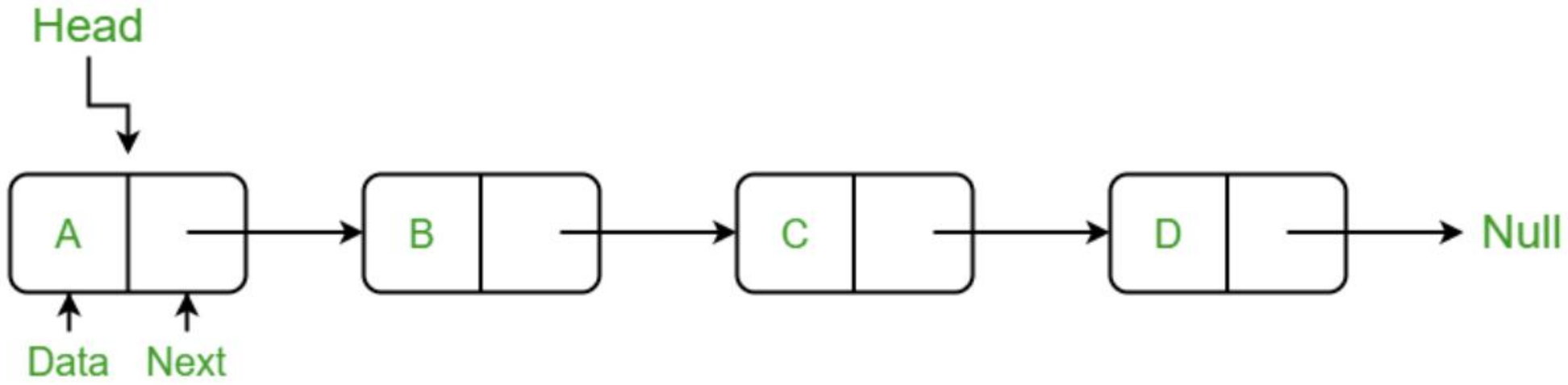
(b) – вставка элемента List-Insert(L, x) x.key = 25.

(c) – удаление элемента из списка List-Delete(L, x), x.key = 4.

If L.head = NIL, список является пустым.

Формы списков

- Список может иметь одну из нескольких форм. Он может быть **односвязным** или **двусвязным**, может быть **отсортированным** или **нет**, **циклическим** или **нет**. Если список односвязный, мы опускаем указатель pre в каждом элементе. Если список отсортирован, линейный порядок списка соответствует линейному порядку ключей, хранящихся в элементах списка; тогда минимальный элемент является головой списка, а максимальный элемент — хвостом. Если список не отсортирован, элементы могут появляться в любом порядке. В круговом списке указатель pre головы списка указывает на хвост, а следующий указатель хвоста списка указывает на голову.



Поиск в связанном списке

- Процедура *List-Search(L, k)* находит первый элемент с ключом *k* в списке L простым линейным поиском, возвращая указатель на этот элемент. Если в списке нет объекта с ключом *k*, то процедура возвращает *NIL*. Для связанного списка на рис предыдущем рисунке (а) вызов *LIST-SEARCH(L, 4)* возвращает указатель на третий элемент, и вызов *LIST-SEARCH(L, 7)* возвращает *NIL*.

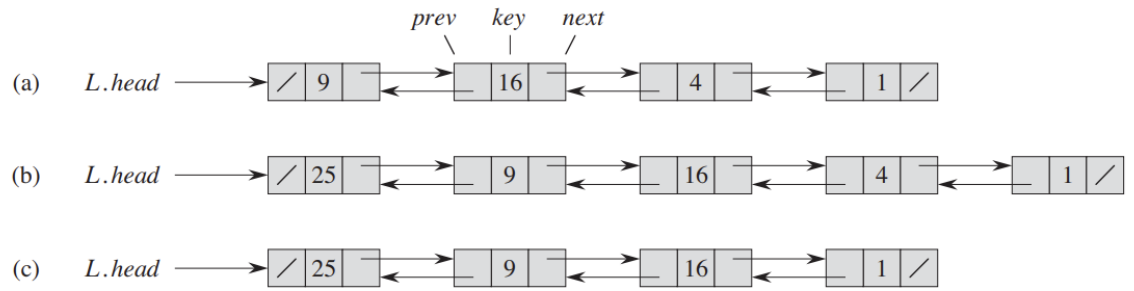
```
LIST-SEARCH(L, k)
1  x = L.head
2  while x ≠ NIL and x.key ≠ k
3      x = x.next
4  return x
```

- Поиск с помощью процедуры *List-Search(L, k) в наихудшем случае выполняется за O(n) время.*

Вставка в связанный список

- Для элемента *x*, *ключевой атрибут (key)* которого уже установлен, то процедура LIST-INSERT «вставляет» *x* в начало связанного списка. В рисунке вариант **(b). Время работы вставки со списком из n элементов O(1).**

```
LIST-INSERT(L, x)
1  x.next = L.head
2  if L.head ≠ NIL
3      L.head.prev = x
4  L.head = x
5  x.prev = NIL
```



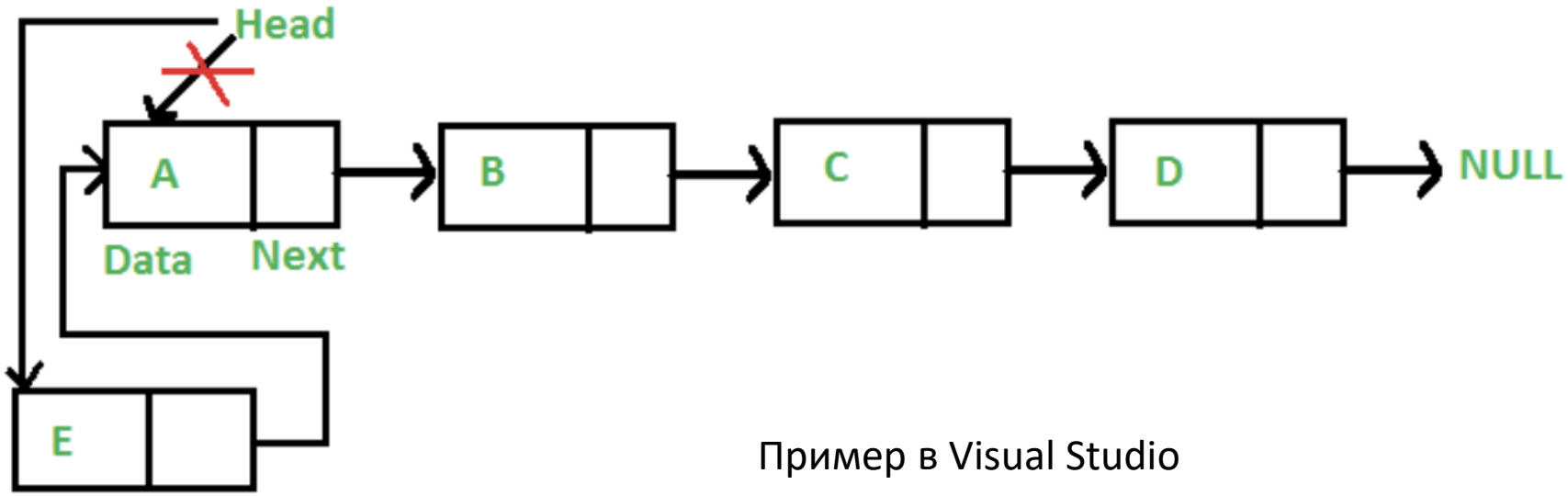
Удаление из связанного списка

- Процедура LIST-DELETE удаляет элемент ***x*** из связанного списка ***L***. Ему должен быть присвоен указатель на x, а затем «склеивается» x из списка путем обновления указателей. Если мы хотим удалить элемент с заданным ключом, мы должны сначала вызвать LIST-SEARCH, чтобы получить указатель на элемент.

```
LIST-DELETE(L, x)
1  if x.prev ≠ NIL
2        x.prev.next = x.next
3  else L.head = x.next
4  if x.next ≠ NIL
5        x.next.prev = x.prev
```

- LIST-DELETE выполняется за время O(1), но если мы хотим удалить элемент с заданным ключом, в худшем случае требуется O(n) времени, потому что мы должны сначала вызвать LIST-SEARCH, чтобы найти элемент.

Практика



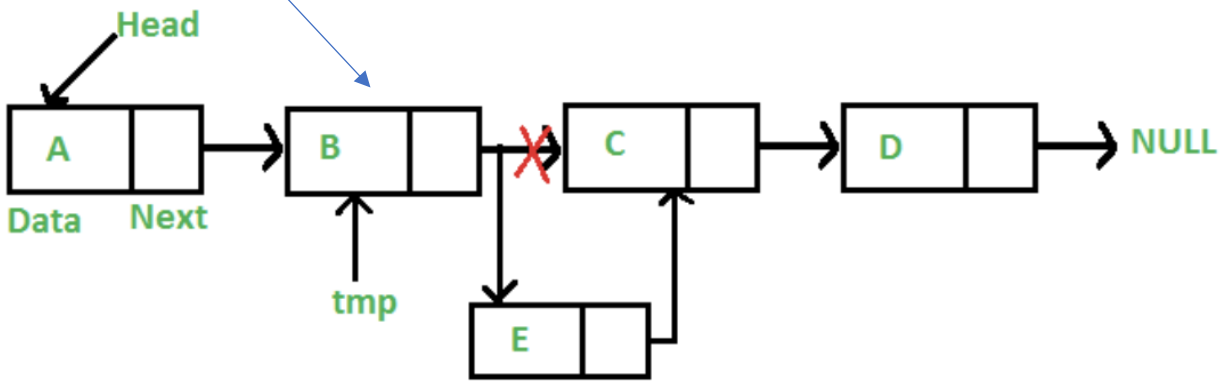
Пример в Visual Studio

Анализ сложности:
Временная сложность: O(1). У нас есть указатель на голову, и мы можем напрямую присоединить узел и изменить указатель. Таким образом, временная сложность вставки узла в положение головы составляет O (1), поскольку он выполняет постоянный объем работы.

Вспомогательное пространство: O(1)

Добавить узел после заданного узла

```
void insertAfter(Node* prev_node, int new_data)
{

    // 1. Check if the given prev_node is NULL
    if (prev_node == NULL) {
        cout << "The given previous node cannot be NULL";
        return;
    }


    // 2. Allocate new node
    Node* new_node = new Node();


    // 3. Put in the data
    new_node->data = new_data;


    // 4. Make next of new node as
    // next of prev_node
    new_node->next = prev_node->next;


    // 5. move the next of prev_node
    // as new_node
    prev_node->next = new_node;
}
```



Выводы

1. В связном списке порядок элементов определяется указателями, а в массиве — индексами.
2. Список может быть односвязным или двусвязным, отсортированным или нет, циклическим или нет.
3. Вставка в начало и удаление при известном указателе на узел выполняются за O(1), а поиск — за O(n).
4. Память выделяется динамически; при работе с указателями необходимо проверять граничные случаи (NULL).

Контрольные вопросы

1. Чем связный список отличается от массива?
2. Какие атрибуты имеет элемент двусвязного списка?
3. Перечислите формы связных списков.
4. Как выполняется поиск в списке и какова его сложность?
5. Как вставить элемент в начало списка? Почему операция выполняется за O(1)?
6. Какие проверки необходимы при вставке узла после заданного узла?

Список литературы

Основная литература

1. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ. 3-е изд. — М.: Вильямс, 2013.
2. Вирт Н. Алгоритмы и структуры данных. — М.: ДМК Пресс, 2010.
3. Седжвик Р., Уэйн К. Алгоритмы на Java. 4-е изд. — М.: Вильямс, 2013.

Дополнительная литература

4. Скиена С. Алгоритмы. Руководство по разработке. 2-е изд. — СПб.: БХВ-Петербург, 2011.
5. Aho A.V., Hopcroft J.E., Ullman J.D. Data Structures and Algorithms. — Addison-Wesley, 1983.
6. Страуструп Б. Программирование: принципы и практика с использованием C++. 2-е изд. — М.: Вильямс, 2016.

Электронные ресурсы

7. VisuAlgo — визуализация алгоритмов и структур данных: visualgo.net
8. Справочник по C++ и STL: en.cppreference.com

Спасибо за внимание!