Алгоритмы и структуры данных

Лекция 8. Связанные списки (1)

Антон Штанюк (к.т.н, доцент)

14 апреля 2022 г.

Нижегородский государственный технический университет им. Р.Е. Алексеева
Институт радиоэлектроники информационных технологий
Кафедра ”Компьютерные технологии в проектировании и производстве”

Содержание

Понятие связанного списка

Списковая структура представляет собой способ организации данных, в котором физический порядок следования элементов в памяти может не совпадать с логическим порядком.

Списковые структуры данных основаны на следующем принципе: элементы связаны друг с другом при помощи явного использования их адресов.

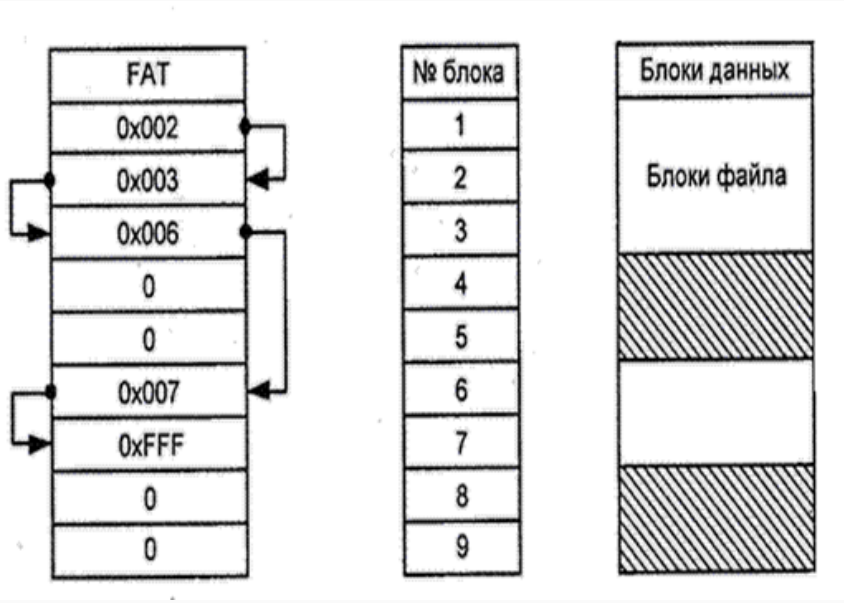
Списки являются линейными СД, поскольку сохраняется определенный порядок следования элементов: первый, второй, третий и т.д.

Списковая структура может быть реализована как с внешней адресацией (адреса следующих или предыдущих элементах хранятся отдельно от полезных данных) и внутренней адресацией (адреса хранятся рядом с полезными данными).

Список с внешней адресацией

Список с внешней адресацией

Рассмотрим пример реализации такого списка на примере широко извест-ной файловой системы FAT



Список с внешней адресацией

Все пространство памяти делится на блоки фиксированного размера. Каждый блок пронумерован и имеет свой адрес. В отдельной области памяти мы храним массив адресов, повторяющих структуру основных блоков. Если основной блок пустой, то элемент массива хранит 0. Если блок заполняется данными, принадлежащими новому файлу, в элемент массива записывается адрес блока, хранящего продолжение файла. Если все содержимое файла помещается в один блок, то в элементе массива помещается специальная метка 0xFFF (конец файла). В результате, в массиве адресов мы получаем цепочку, пройдя по которой можно считать содержимое файла.

Список с внешней адресацией

Несомненным достоинством такой организации является простота, а также возможность работы в условиях фрагментации, когда файлы занимают не смежные участки памяти.

Список с внутренней адресацией

Список с внутренней адресацией

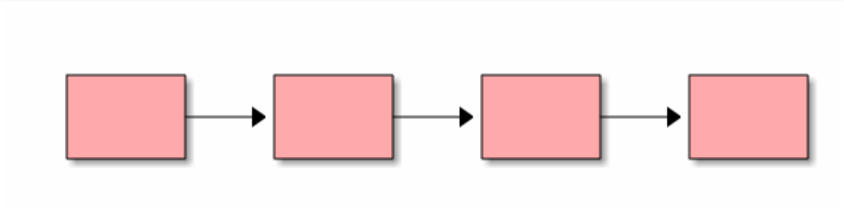
Список с внутренней адресацией состоит из элементов (звеньев), в каждом из которых мы храним полезные данные и адрес следующего (и предыдущего элемента).

В соответствии с характером связей, мы можем выделить несколько видов списка:

1. Односвязный список
2. Двусвязный список
3. Кольцевой список

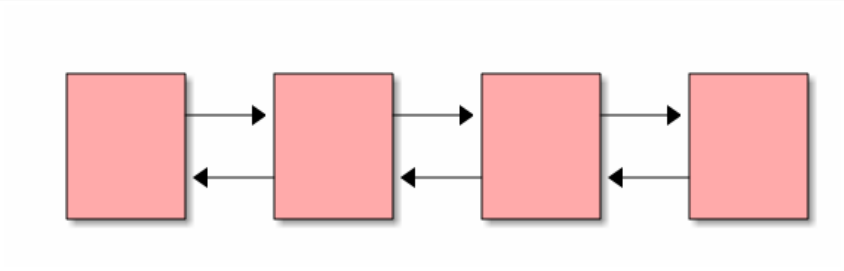
Односвязный список

В таком списке каждый элемент хранит адрес следующего элемента



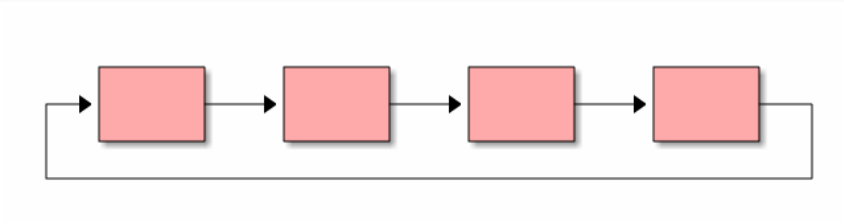
Двусвязный список

В таком списке каждый элемент хранит адрес следующего элемента и адрес предыдущего



Кольцевой список

Кольцевой список может быть и односвязным и двусвязным, главное, чтобы он позволял перейти с последнего элемента на первый (и/или с первого на последний).



Программная реализация односвязного списка

Основу программной реализации простого односвязного списка составляет класс, который хранит указатели на головной и хвостовой элементы. Список состоит из звеньев, которые описываются вложенным классом **ITEM**. В каждый элемент мы помещаем экземпляр данных типа **T** и указатель на следующее звено.

```
#include <cassert>
template<typename T>
class LList {
    struct ITEM {
        T data;
        ITEM * next;
    };
public:
    ...
```

Программная реализация односвязного списка

```
...
    LList():head(nullptr),tail(nullptr){}
    LList(const T&);
    LList(const LList&);
    ~LList();
    void addTail(const T&);
    void addHead(const T&);
    T rmHead();
    void print() const;
private:
    LList::ITEM* create(const T&);
    ITEM *head;
    ITEM *tail;
};
```

Программная реализация односвязного списка

В качестве программного интерфейса мы используем:

- **create()** - метод для создания нового звена.
- **addTail()** - метод добавления нового звена в хвост списка.
- **addHead()** - метод добавления нового звена в голову списка.
- **rmHead()** - метод удаления головного элемента.
- **rmtail()** - метод удаления хвостового элемента.

Программная реализация односвязного списка

Опишем реализацию стандартных методов класса (конструкторов и деструктора)

```
template<typename T>
LList<T>::LList(const T& data) {
    head=create(data);
    tail=head;
}
template<typename T>
LList<T>::~LList() {
    while(head)
        rmHead();
}
```

Программная реализация односвязного списка

Напишем реализацию метода создания нового звена:

```
template<typename T>
typename LList<T>::ITEM* LList<T>::create(const T& data) {
   ITEM *item=new ITEM;
   item->data=data;
   item->next=nullptr;
   return item;
}
```

Программная реализация односвязного списка

Теперь напишем методы добавления нового элемента в голову и хвост списка:

```
template<typename T>
void LList<T>::addTail(const T& data) {
    if(tail && head) {
        tail->next=create(data);
        tail=tail->next;
    }
    else {
        head=create(data);
        tail=head;
    }
}
```

Программная реализация односвязного списка

```
template<typename T>
void LList<T>::addHead(const T& data) {
    if(tail && head) {
        ITEM *temp=create(data);
        temp—>next=head;
        head=temp;
    }
    else {
        head=create(data);
        tail=head;
    }
}
```

Программная реализация односвязного списка

Теперь приведем в качестве примера метод удаления головного элемента списка:

```
template<typename T>
T LList<T>::rmHead() {
    if(head) {
        ITEM *temp=head->next;
        T data=head->data;
        delete head;
        head=temp;
        return data;
    }
    else {
        throw std::string("Empty!");
    }
}
```

Программная реализация односвязного списка

Для того, чтобы можно было распечатывать список, напишем метод **print**:

```
template<typename T>
void LList<T>::print() const {
    ITEM *temp=head;
    while(temp) {
        std::cout<<temp->data<<"␣";
        temp=temp->next;
    }
    std::cout<<std::endl;
}
```

Пример использования односвязного списка

```
#include "tlist.h"

int main() {
    int i;
    LList<int> list;
    for(i=1;i<10;i++) {
        list.addTail(i);
        list.print();
    }
    for(i=10;i<15;i++) {
        list.addHead(i);
        list.print();
    }
    while(list.rmHead())
        list.print();
    return 0;
}
```

Пример использования односвязного списка

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
1 2 3 4 5 6 7
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8 9
10 1 2 3 4 5 6 7 8 9
11 10 1 2 3 4 5 6 7 8 9
12 11 10 1 2 3 4 5 6 7 8 9
13 12 11 10 1 2 3 4 5 6 7 8 9
14 13 12 11 10 1 2 3 4 5 6 7 8 9
13 12 11 10 1 2 3 4 5 6 7 8 9
12 11 10 1 2 3 4 5 6 7 8 9
11 10 1 2 3 4 5 6 7 8 9
```

Список литературы

Список литературы i

- Кормен Т.,Лейзерсон Ч., Ривест Р.
  Алгоритмы: построение и анализ
  МЦНМО, Москва, 2000
- **Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ.**
  2-е изд. — М.: «Вильямс», 2006
- **Википедия**
  Алгоритм
  http://ru.wikipedia.org/wiki/Алгоритм
- **Википедия**
  Список алгоритмов
  http://ru.wikipedia.org/wiki/Список_алгоритмов
- **Традиция**
  Задача коммивояжёра
  http://traditio.ru/wiki/Задача

Список литературы ii

- **Википедия**
  NP-полная задача
  http://ru.wikipedia.org/wiki/NP-полная
- **Серджвик Р.**
  Фундаментальные алгоритмы на C++. Части 1-4
  Diasoft,2001
- **Седжвик Р.**
  Фундаментальные алгоритмы на C. Анализ/Структуры данных/Сортировка/Поиск
  СПб.: ДиаСофтЮП, 2003
- **Седжвик Р.**
  Фундаментальные алгоритмы на C. Алгоритмы на графах
  СПб.: ДиаСофтЮП, 2003
- **Ахо А., Хопкрофт Д., Ульман Д. Структуры данных и алгоритмы.**
  Издательский дом «Вильямс», 2000

Список литературы iii

- **Кнут Д.**
  Искусство программирования, том 1. Основные алгоритмы
  3-е изд. — М.: «Вильямс», 2006
- **Кнут Д.**
  Искусство программирования, том 2. Получисленные методы
  3-е изд. — М.: «Вильямс», 2007
- **Кнут Д.**
  Искусство программирования, том 3. Сортировка и поиск
  2-е изд. — М.: «Вильямс», 2007
- **Кнут Д.**
  Искусство программирования, том 4, выпуск 3. Генерация всех сочетаний и разбиений
  М.: «Вильямс», 2007

Список литературы iv

Кнут Д.
Искусство программирования, том 4, выпуск 4. Генерация всех деревьев. История комбинаторной генерации
М.: «Вильямс», 2007