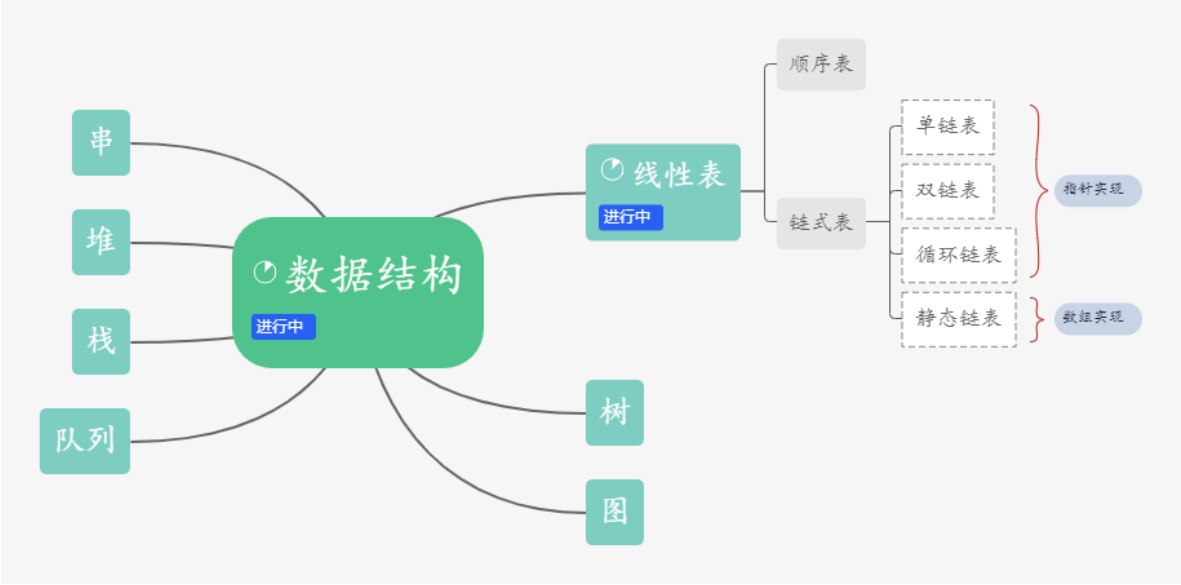
数据结构文档(二):向量初探

Author: Limzh

Section1. 序言

1.1 知识图谱



1.2 课程大纲对照

本文档涉及的内容已用`对勾`标记。

> - [x] 线性表的定义和基本运算
> - [x] 线性表的顺序存储结构及查找、插入和删除等基本算法的实现
> - [ ] 线性表的链式存储结构和查找、插入和删除等基本算法的实现
> - [ ] 循环列表、静态链表和双向链表的基本概念
> - [ ] 跳表的基本原理
> - [ ] 理解并能选择使用线性结构
> - [ ] 线性表的应用

重点内容:

> - [x] 线性表的存储以及顺序表的元素插入和删除操作
> - [ ] 线性表的链式存储以及单链表的查找、插入和删除等基本运算的实现
> - [ ] 链表的应用
> - [ ] 循环链表

1.3 如何认识顺序表?

学习`Python`时，我们在繁驳的语法中抽丝出`变量引用机制（深浅赋值）`、`作用域`和`面向对象`等最底层、最本质的几个核心机制。类似地，数据结构也有其核心设计思想。但与`Python`时所采取的以核心机制为主干，语法语法为辅的学习策略不同，学习数据结构我们将以各个结构为主干，核心设计思想为提炼来学习。

1.3.1 技术上的认识

各种数据结构都可以看做是由若干数据项组成的集合，同时对若干数据项定义一组标准的操作方法。学习数据结构，不是在学习一个又一个繁杂的数据项和操作，而是在学习设计数据项与数据项之间的层次性——`结构`。而我们凭借什么标准来设计数据与数据的之间的结构呢？又怎么去度量结构的工作效率？结构这个抽象的概念具体又体现在什么地方？这是三个贯穿数据结构学习始终的终极之问。但目前，让我们先大喊一句至理名言，再慢慢体会这种感觉：

> 离散数据间逻辑上、空间上、操作上的关联就是数据结构。

介绍顺序表，让我们先从数组开始。

C、C++和Java等程序设计语言，都将数组作为一种内置的数据类型，支持对一组相关元素的存储组织与访问操作。具体地，若集合S由n个元素组成，且各元素之间具有一个线性次序，则可将它们存放于起始于地址A、物理位置连续的一段存储空间，并统称作数组（array），通常以A作为该数组的标识。具体地，若数组A[]存放空间的起始地址为A，且每个元素占用s个单位的空间，则元素A[i]对应的物理地址为：

$$A[i] = A + i \times s$$

因其中元素的物理地址与其下标之间满足这种线性关系，故亦称作线性数组（linear array）

如果我们将线性数组抽象泛化，不再限定同一个向量中个元素都属于同一基本类型，并封装成一个具有自定义方法的对象，那么我们将获得一个仍旧具有线性次序的数据结构。对线性数组抽线泛化出的数据结构，我们称之为`顺序表`，也称之为`向量`，为与教学大纲同步，我接下来中文称为`顺序表`，当使用英文的时候，为方便起见使用`vector`。

因此，我们可以不那么严谨地认为，`顺序表`就是一种拓展版的数组，它的本质特点是（也就是线性数组的本质特点）：

- 逻辑上相邻的元素，其实际物理地址也相邻。(本质上是满足物理地址的线性关系)

它的泛化特点是：

- 同一个顺序表中的元素不一定都属于同一个基本类型。（顺序表可以不满足这个特点，只要满足本质特点我们都认为是线性表）
- 顺序表是一种具有自定义方法的数据类型。（同上）

不妨认为，线性数组就是一种最简单、最基础的顺序表。

1.3.2 生活上的认识

场景： 仓库管理员盲人老李的工作

老李作为盲人虽然看不见东西，但是他的空间感知力特别强大，也就是说：给定某坐标，他可以像正常人一样去往该坐标。令人惊叹的是，他还是一位优秀的仓库管理员，工作效率丝毫不输年轻小伙。

基本的管理仓库的货物的工作有查找、添加和删除。

查找是最重要的工作了，具体指：快递员告诉老李，我要第n个存入仓库的货物。老李去把第n个货物取出交给快递员。

但他既不会使用计算机，也没有额外的助手，货物零散地分布在仓库的各个角落。所以，他很自然地想到，只要我把每一个货物的顺序和位置记住，要用的时候在脑子里检索就可以了！但仓库的货物越来越多，不是长久之计。他陷入沉思，如何才能记少量的东西，当快递员给我第n个货物的时候我能快速找到它呢？

老李灵光一现，有了！将原本零散的货物按照存入仓库的时间依次、按顺序排开，并使货物与货物间紧密相连，并在每一个货物上面写下它的宽度，这下我只要记住第一个货物的位置，货物的总数两个数据就可以啦！当快递员说：嘿！给我第8个存入仓库的货物，我就从从第一个货物开始走，每一步的步子就是当前货物宽度，这样只要走n-1步，我的手边就是第n个存入仓库的货物了。不错不错，我实在是个小天才。

他又继续深入思考下去，哎呀！这不就是一种顺序表的形式吗？

在我的脑子里，货物按照存入仓库的顺序依次排序，他们是逻辑相邻的。实际的空间上，他们也是依次、紧密靠在一起。这样才保证我的每一步都能找到一个货物！如果他们在空间上隔一个，空一格，那么我肯定会有一步找不到货物，我也就不知道接下来应该迈出多大的步子了，这个方法就失效了。因此，我这个方法成功的第一个要义就是：逻辑上紧密相邻，空间上亦然。

我要记录在脑中的数据是第一个货物的地址，也即是`基地址`，售货员告诉我的第n个货物，也即是`指定货物到基地址的偏差`。给定输入的偏差，第n个货物的位置符合公式

$$A[n] = A_0 + l_0 + l_1 + \ldots + l_{n-1}$$

假如货物之间的尺寸一样，也即符合

$$A[n] = A_0 + l * n$$

这就是顺序表的本质公式。

1.3.3 本质上的认识

离散的数据不可称之为一种结构，赋予离散数据某种位置关系，便使之成为一种结构体。本质上，

$$A[n] = A_0 + l * n$$

刻画的便是第n个元素和第1个元素之间的位置关系，需要注意的是，逻辑上具有这样位置关系的式子在物理地址上可能不具有这样的位置关系（这也就是我们之后要说的链表）。因此，避免引起混淆，我们要区分逻辑上的数据结构和物理地址上的数据结构。

线性表的本质关系是:

$$A_0 \rightarrow A_1 \rightarrow A_2 \rightarrow A_3 \rightarrow \ldots \rightarrow A_{n-1}$$

这是逻辑上的结构，箭头在这里仅仅代表次序！！所以线性表是一种表示元素次序的逻辑数据结构，这种元素次序体现在顺序表的实现中，就是物理地址上的结构：次序近的元素彼此相邻。

Section2.顺序表ADT的操作方法

还是从老李和快递员的例子出发，在这个例子里我们不妨认为快递员是`工程的使用者`，他的脑海里存在的结构是`逻辑上的线性表`，他只在乎对存入货物的操作而不在乎具体的存放、寻址过程。盲人老李是`工程的开发者`，他在乎的是`物理地址上的顺序表的实现`。他们之间通过快递员的口令来实现交互。这个口令我们称之为`接口`。整个仓库（我们不妨叫它`杨光仓库`）就是`顺序表`这一数据结构的一个对象。现在我们分析一下这个数据结构具有什么属性。

- 盲人老李：他只记住了第一个元素的位置、所有元素的数目以及整个仓库的容量。
- 货物： 按照线性顺序放置的货物。

```
class Vector{
private:
    int* _elem; int _size; int _capacity; //数据区、规模和容量
public:
    Vector(int size, int capacity, int value); // 构造函数 （创建时给定规模、容量以及初始化填充的数值）
    ~Vector(); // 析构函数（销毁该对象）
};
```

2.1 构造函数

最简单的构造函数就是创建一段连续的物理空间，并且记录这个空间的大小以及填充元素的数目。（想象一下在一片空地上创建一个仓库，我们要记录仓库的大小以及里面元素的数目）。

```
Vector::Vector(int size = 0, int capacity = 10, int value = 0){
    _elem = new int[capacity];
    for(int i = 0; i < size; i++) _elem[i] = value;
    _size = size;
    _capacity = capacity;
}
```

2.2 析构函数

析构函数是释放空间的函数.

```
Vector::~Vector(){delete [] _elem;}
```

2.3 API接口

一个顺序表至少还需要以下接口

函数	描述	对象
`int size()`	返回当前数据规模	顺序表
`int get(int r)`	给定某一索引r,返回r位置上的元素值	顺序表
`void insert(int r, int const & e)`	给定某一索引r，在该位置插入一元素e，并将所有在r之后的元素后移一格	顺序表
`void remove(int r, int const & e)`	删除r上的元素，并将所有在r之后的元素前移一格	顺序表
`void replace(int r, int const & e)`	用元素e替换掉r上的元素	顺序表

在这里请思考，以上这些方法在使用的时候可能会出现一些内存问题，是什么呢？（提示：数组是静态的，向量是动态的）

2.4 实现

```
/*This is a header file containing the implementation
  of Vector for int type.*/
#define DEFAULT_CAPACITY 10 // 向量的默认大小
typedef int Rank; // Rank秩 本质上就是线性表的索引
class Vector{
protected:
    Rank _size; int _capacity; int* _elem; // 规模，容量，数据区
public:
    Vector(int capacity, int size, int value);//初始函数（构造函数）
    ~Vector();//析构函数
    Rank size() const{return _size;}
    bool empty() const{return !_size;}
    void expand();
    Rank insert(Rank r, int const& e);
    Rank insert(int const& e){return insert(_size, e);}
    int remove(Rank r);
    int get(Rank r);
    void replace(Rank r, int const& e);
};//注意不要漏掉分号

Vector::Vector(int capacity = DEFAULT_CAPACITY, int size = 0, int value = 0){
    _capacity = capacity;
    _size = size;
    _elem = new int[_capacity];
    for(int i = 0; i < _capacity; i++) _elem[i] = value;
}

Vector::~Vector() {delete [] _elem;}

void
Vector::expand(){
    if(_size < _capacity) return;
    if(_capacity < DEFAULT_CAPACITY) _capacity = DEFAULT_CAPACITY;
    int* oldElem = _elem; _elem = new int[_capacity << 1];
    for(int i = 0; i < _size; i++) _elem[i] = oldElem[i];
    delete [] oldElem;
    _capacity = _capacity << 1;
}

Rank
Vector::insert(Rank r, int const& e){
    expand();
    for(int i = _size - 1; i >= r; i--) _elem[i+1] = _elem[i];
    _elem[r] = e;
    _size ++;
}

int
Vector::remove(Rank r){
    for(int i = r; i < _size - 1; i++) _elem[i] = _elem[i+1];
    _size --;
}
```

```
int
Vector::get(Rank r){return _elem[r];}

void
Vector::replace(Rank r, int const& e){
    _elem[r] = e;
}
```

以上是最最最最基础的顺序表，下一文档我们将在这个顺序表的基础上引入算法，并增添新的方法。

2.5 练习

- 请问如果你是顺序表的开发者，你会添加什么样的新操作方法?

Section3. 总结

本文档仅仅是对顺序表的初探，介绍了什么是顺序表以及数据结构的基本思想。基于最粗浅的理解，我们实现了一种最简单的顺序表结构。在接下来的两篇文档里，我们会进一步地完善线性表的操作方法，并引入算法分析的知识，衡量顺序表的工作效能。

本文档最核心的知识归纳如下：

- 终极三问：数据结构的`结构`体现在哪里？ 如何设计并实现一种数据结构(的操作方法)？如何度量数据结构的工作效能?
- 数据结构分为逻辑上的结构关系和物理实现上的结构关系，线性表是一种逻辑结构，顺序表是对线性表的一种物理实现。
- 线性表的公式（逻辑结构，重点是`次序`）： $A_0 \rightarrow A_1 \rightarrow A_2 \rightarrow A_3 \rightarrow \ldots \rightarrow A_{n-1}$
- 顺序表的公式（物理结构）： $A[n] = A_0 + l_0 + l_1 + \ldots + l_{n-1} = A_0 + l * n$

实现代码也已一并放到了附件中，必要时请浏览。