Программирование в задачах радиолокации

КМБО-02-19

21 июня 2020 г.

1 Массивы, векторы, связные списки. Устройство, основные операции, способы реализации

1.1 Массив

Определение 1 ***Массив*** *- базовая структура данных - набор однотипных элементов, расположенных в памяти последовательно друг за другом.*

Пояснять тут больше нечего...

1.2 Вектор

Определение 2 ***Вектор*** *- структура данных, являющаяся надстройкой над массивом и поддерживающая следующие операции:*

- *Доступ к элементу;*
- *Вставка элемента;*
- *Удаление элемента;*
- *Поиск.*

1.2.1 Устройство вектора

Вектор содержит в себе динамический массив размера *capacity*. Текущее заполнение этого массива описывается параметром *size*.

Когда *size* достигает определённого размера относительно *capacity*(об этом поговорим чуть позже), происходит перевыделение памяти: либо выделяется блок памяти размером поменьше, либо размером побольше - и туда перемещается *size* элементов массива. Одновременно с этим меняется и *capacity*.

1.2.2 Изменение *capacity* в векторе

Существует две стратегии изменения *capacity*:

- Аддитивная: $capacity = oldCapacity + \Delta$. При этом Δ задаётся либо по умолчанию, либо в момент создания вектора.
- Мультипликативная: $capacity = coef * oldCapacity$. При этом $coef$ задаётся либо по умолчанию, либо в момент создания вектора.

1.2.3 Условие изменения *capacity* в векторе

Здесь вволится следующая величина:

$$loadFactor = \frac{size}{capacity}$$

Если выбрана аддитивная стратегия:

$\begin{cases} if(loadFactor \geq 0.8) \Rightarrow while(loadFactor \geq 0.8) capacity := capacity + \Delta; \end{cases}$

Если выбрана мультипликативная стратегия:

$\begin{cases} if(loadFactor \leq \frac{1}{coef^2}) \Rightarrow \text{обрезаем память} \\ if(loadFactor \geq 0.8) \Rightarrow \text{Перевыделяем память} \end{cases}$

1.2.4 Оценки операций с вектором

	Лучший случай	Средний случай	Худший случай
Доступ	$O(1)$	$O(1)$	$O(1)$
Вставка	$Amort(O(1))$ [вставка в конец]	$O(n)$ [вставка не в конец]	$O(n)$ [вставка в начало]
Удаление	$Amort(O(1))$ [удаление с конца]	$O(n)$ [удаление не из конца]	$O(n)$ [удаление первого элемента]
Поиск	$O(1)$[искомый – первый]	$O(n)$ [искомый - не первый]	$O(n)$ [искомый - последний]

1.2.5 Резюме

Достоинства:

$\begin{cases} \text{Минимальный } overhead; \\ \text{Простота реализации;} \\ \text{Быстрый доступ к элементам коллекции.} \end{cases}$

Недостатки:

$\begin{cases} \text{Необходимо иметь нефрагментированную память;} \\ \text{При большой загрузке долгое перевыделение памяти.} \end{cases}$

Используется в задачах, где необходим *гарантированно* быстрый доступ к данным.

1.3 Связный список

Определение 3 ***Связный список*** *- структура данных, в которой каждый элемента содержит указатель на следующий элемент (случай односвязного списка) или и на следующий, и на предыдущий (двусвязный список).*

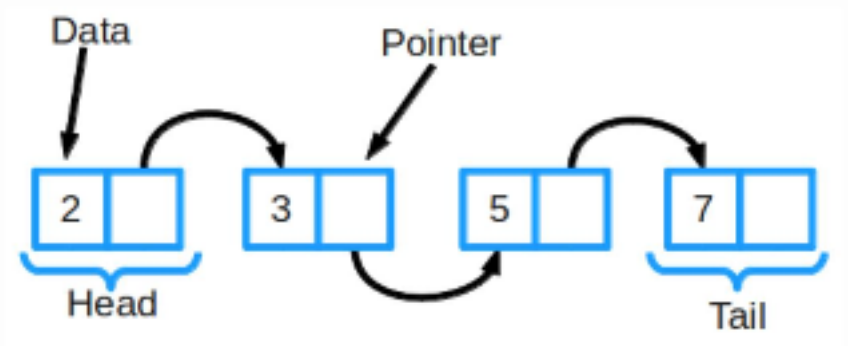

Рис. 1: Односвязный список

1.3.1 Основные операции над списками

- Доступ к элементу;
- Вставка элемента;
- Удаление элемента;
- Поиск.

1.3.2 Оценка операций

	Лучший случай	Средний случай	Худший случай
Доступ	$O(1)$	$O(n)$	$O(n)$
Вставка (**если знаем**, куда вставлять)	$O(1)$ [вставка в конец]	$O(1)$	$O(1)$
Удаление (**если знаем**, откуда удалять)	$O(1)$	$O(1)$	$O(1)$
Поиск	$O(1)$[искомый – первый]	$O(n)$ [искомый - не первый]	$O(n)$ [искомый - последний]

1.4 Резюме

Достоинства:

$$\begin{cases} \text{Быстрое удаление;} \\ \text{Быстрая вставка;} \\ \text{Нет ограничений на фрагментацию памяти.} \end{cases}$$

Недостатки:

$$\begin{cases} \text{Довольно приличный } overhead; \\ \text{Долгий доступ к элементая коллекции.} \end{cases}$$

Используется в задачах, где необходимы *гарантированно* быстрые добавление и удаление узлов.