Методическое пособие

1 Обзор архитектуры исполнения и компиляции программы.

2 Сложность вычислений. О-нотация.

2.1 Методы сравнения программ

- Время работы
- Количеством использованной памяти

Как можно сравнивать программы решающие одну и туже задачу? Первое что хочется сказать это сколько времени программа отрабатывает на компьютере, но это не совсем так, так как вместе с вашей программой на нем работает еще очень много других программ. Для этого будем использовать не просто время работы программы, а время которое затратил процессор на нашу программу. Дальше всегда будем предполагать, что время работы программы это время затраченной процессором на выполнения только нашей программы.

От чего зависит время работы программы? Конечно же время работы программы зависит от мощности компьютера и тому подобное, но так-как мы договорились, что время работы это количество времени которое затратит процессор, то здесь уже явная зависимость от компьютера отпадает. На самом деле нас интересует зависимость времени работы программы от входных данных, а точнее от их размера.

В дальнейшем, чтобы отойти немного от компьютера введем понятия алгоритм.

Определение 1: Алгоритм - это некоторая последовательность действий, которая преобразует начальные данные (input data) в выходные данные (output data).

Введя понятия алгоритм мы будем сравнивать не программы выполняемые на компьютере а некоторые математические модели.

Для простоты сравнения алгоритмов рассматривается асимптотическое сравнение двух алгоритмов. То есть как ведет себя алгоритм при изменении размера входных данных.

Определение 2: Говорят, что $f(n) \in O(g(n))$, если

$$\exists C, N : \forall n > N \to f(n) < Cg(n), \tag{1}$$

Определение 3: Говорят, что $f(n) \in \Omega(g(n))$, если

$$\exists C, N : \forall n > N \to f(n) > Cg(n), \tag{2}$$

Определение 4: Говорят, что $f(n) \in \Theta(g(n))$, если $f(n) \in \Omega(g(n))$ и $f(n) \in O(g(n))$.

Введем некоторые функции, которые будут полезны для оценки сложности алгоритмов. Первой функцией является функция времени работы алгоритма от размера входа. Второй функцией является функция памяти используемой алгоритмом от размера входа.

Определение 5: $T(n)$ — это время работы алгоритма на входе (input data) длины n, где под длинной n подразумевается длина двоичной записи входных данных.

Определение 6: $M(n)$ — это объем дополнительной памяти требуемой алгоритмом на входе (input data) длины n, где под длинной n подразумевается длина двоичной записи входных данных. Под дополнительной памятью подразумевается вся память кроме входа.

2.2 Примеры асимптотик

2.3 Задача

$sum = 0$
for $0 < i < n$ *do*
....$sum+ = 1$

Пусть время затраченное на прибавление единицы равно t. Найти асимптотическое время работы алгоритма $T_1(n)$ для этой задачи.

$$T_1(n) = \Theta(n \cdot t)$$

2.4 Задача

$sum = 0$
for $0 < i < n$ *do*
....for $0 < j < n$ *do*
........$sum+ = 1$

Пусть время затраченное на прибавление единицы равно t. Найти асимптотическое время работы алгоритма $T_2(n)$ для этой задачи.

$$T_2(n) = \Theta(n^2 \cdot t)$$

2.5 Задача

def f(char*str)
....if(len(str) > 0)
........return f($str + 1$)
....else
........return 0

Пусть время затраченное на прибавление единицы к индексу строки равно t_1, время нахождения длины строки равно t_2. Найти асимптотическое время работы алгоритма $T_3(n)$ для этой задачи, где n это длина строки.

$$T_3(n) = O(n \cdot (t_1 + t_2))$$

2.6 Задача

$min = 2^{32}$
arr — массив размера n, где все числа меньше 2^{32}
for $0 < i < n$ *do*
....if($arr[i] < min$)
.....$min = arr[i]$

Пусть время затраченное на сравнение двух чисел равно t. Найти асимптотическое время работы алгоритма $T_4(n)$.

$$T_4(n) = O(n \cdot t)$$

2.7 Задача

arr — массив размера n
for $0 < i < n$ *do*
....for $0 < j < n$ *do*
.....if($arr[j] > arr[j + 1]$)
.....swap($arr[j], arr[j + 1]$)

Пусть время затраченное на сравнение двух чисел равно t_1, а время затрачиваемое на перемещения двух рядом стоящих элементов массива это t_2. Найти асимптотическое время работы алгоритма $T_5(n)$.

$$T_5(n) = O(n^2 \cdot (t_1 + t_2))$$

3 Классы. Введение в ООП.

4 Базовые структуры данных.

4.1 Список

- Каждый элемент знает только свое значение.
- Каждый элемент знает где находиться следующий элемент списка.
- * В случае двусвязного списка каждый элемент знает где находится предыдущий элемент.

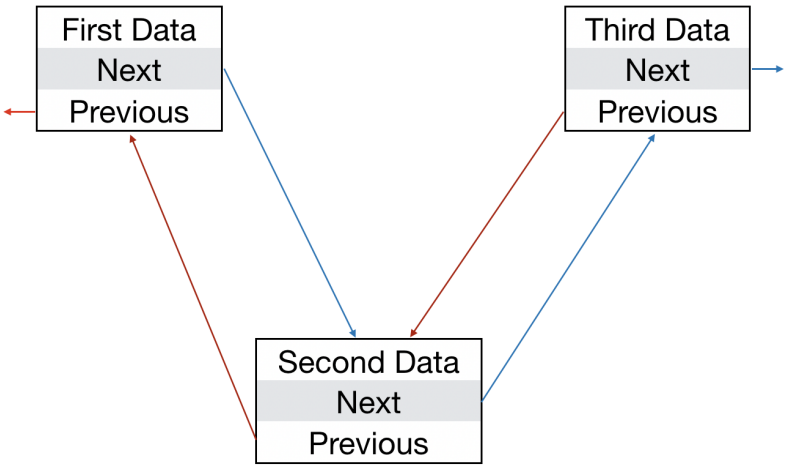

Рис. 1: Изображение списка

На рис. 1 изображена общая структура списка. Если на рис. 1 оставить только синие линии то получим простой список, если же оставить все линии, то получаем двусвязный список.

Основные операции:

- append() — добавить в конец списка,
- pop(index) — вытащить элемент со списка с индексом index,
- len() — возвращает длину списка.

4.2 Стэк

- Главное свойство, кто первый попадает в стэк, тот последним из него выходит.

На рис. 2 изображена общая структура стэка. Как видно сначала из стэка достается элемент который в него был поставлен в самом конце и последним достается элемент который был поставлен первым. На рис. 2 синяя стрелочка указывает, на место куда будет вставлен следующий элемент и какой элемент будет изъят следующим.

Основные операции:

- push() — добавить в вверх стэка,
- pop() — вытащить верхний элемент стэка,
- len() — возвращает глубину стэка.

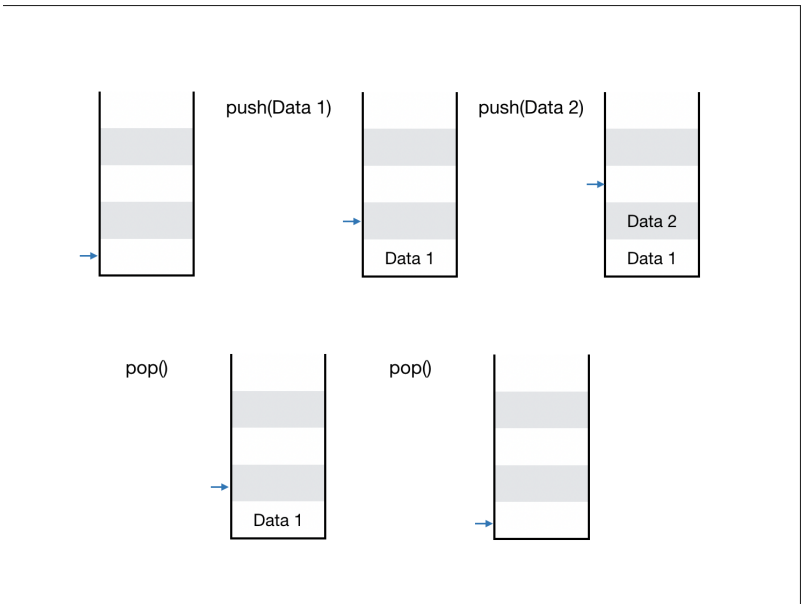

Рис. 2: Изображение стэка

4.3 Очередь

- Главное свойство, кто первый попадает в очередь, тот первым из нее выходит в отличии от стэка.

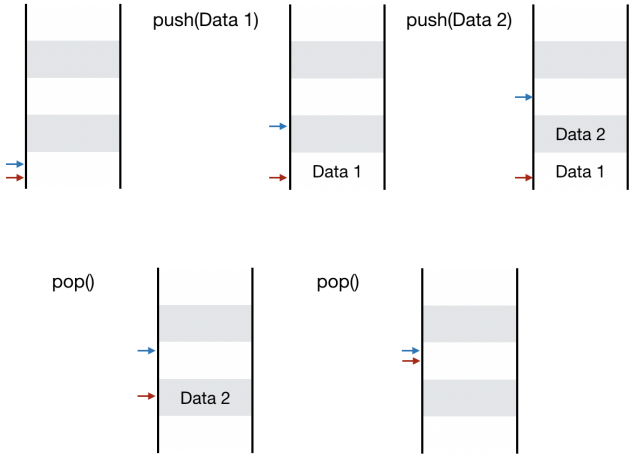

Рис. 3: Изображение стэка

На рис. 3 изображена общая структура очереди. Синяя срелочка показывает в какое место будет вставлен следующий элемент, а красная стрелочка указывает откуда будет браться

следующий элемент. Как видно, первый элемент который попал в очередь первый из нее выходит.

Основные операции:

- push() — добавить в конец очередь,
- pop() — вытащить первый элемент очереди,
- len() — возвращает длину очереди

4.4 Set

- Это структура данных, которая отвечает такой математической структуре как множество.
- Главная задача этой структуры, это ответить на вопрос есть ли элемент в множестве.
- В отличии от структур 1—3 эта структура не имеет порядка.

Основные операции:

- insert() — добавить элемент в множество,
- find() — найти элемент в множестве,

Операция find() вернет либо 0 либо 1 в зависимости от того, принадлежит элемент множеству или нет.

4.5 Map

- map и set очень похожие структуры данных.
- map отличается от set, только тем, что в нем храниться не только информация есть ли элемент а еще и каждому элементу ставиться в соответствии еще один элемент.
- Каждая пара состоит из key и data, где key должен быть уникальным.

Пример использования map очень просто, например мы хотим сделать перепись книг в библиотеке. В качестве уникальных key будет выступать название книги, а в качестве data будет выступать числа книг соответствующих этому названию.

4.6 Hash

- Хэш таблица — способ хранения данных

- Нужна hash функция для данных которые вы собираетесь хранить

Хэш таблица это просто массив фиксированного размера в котором хранятся некоторые данные, но доступ к этим данным происходит не по простой индексации, а при помощи hash функции.

Hash функция это некоторое отображения данных которые нужно сохранить в натуральное число — индекс массива.

Основной минус, что количество данных обычно много больше чем размер массива, поэтому на самом деле элемент массива это список тех элементов у которых Hash функция дает один и тот же результат. Список решает проблему так называемой коллизии.

5 Рекурсивные алгоритмы.

6 Динамическое програмирование.

7 Простейший пример поиска в ширину на таблице.

8 Задача поиска подматриицы с задаными свойствами.

9 Понятие графа. Представление графа в компьютере.

10 Алгоритм Дейскстры.

11 Алгоритм Флойда-Уоршала.

12 Поиск в шириу. Поиск в глубину.

13 Понятие дерева. Представление дерева в памяти компьютера.

14 Работа с деревьями. Бинарное дерево с упорядочеными вершинами.

15 Работа с деревьями. Бинарное дерево поиска.

16 Работа с деревьями. Проверка, что граф это дерево и нахождение высоты дерева.