Информатика

Массивы, строки

© Марченко Антон Александрович 2017 г.
Абрамский Михаил Михайлович

Необходимость хранения данных

- Не все задачи можно решить с O(1) памяти
  - Сортировка данных
  - Работа с матрицами/таблицами
  - Длинная арифметика
  - Вычисление булевой функции от n переменных
    - ДНФ, полином Жегалкина

Переменные примитивных типов

- Имея только переменные примитивных типов, невозможно динамически управлять размером памяти, выделяемого на задачу
  - можно решить только простые задачи
    - почти все алгоритмы с константной памятью – выполняются за константное время

Константная память

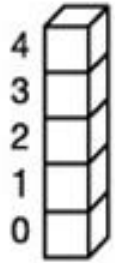
- Все задачи первой семестровки:
  - Вычисление простых формул
  - Нахождение максимума, минимума, медианы в массиве
  - Нахождение k максимумов (k-константа)
  - Потоковые вычисления

Динамическая память

- Нужно уметь записывать входные данные для обработки
- *Размер памяти зависит от входа* – нужно динамически выделять память во время работы
  - Не объявляем массив из 10000 элементов, а затем вводим размер массива 5
  - сразу выделяем столько памяти, сколько нужно

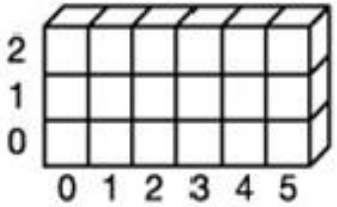
Массивы

One-Dimensional Arrays

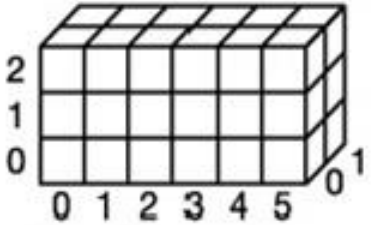


One-Dimensional
`int[5]`

Rectangular Arrays

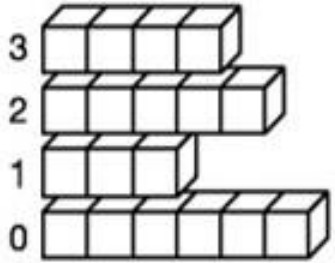


Two-Dimensional
`int[3,6]`



Three-Dimensional
`int[3,6,2]`

Jagged Arrays



Jagged Array
`int[4][]`

Массив

- Упорядоченная последовательность однотипных данных
- Объявление: `int[] array;`
- Динамическое выделение памяти:
  arr = new `int`[размер];
  - размер – целое число, вычисляющееся заранее
  - элементы массива инициализируются по умолчанию
    - 0 – типы значения
    - null – ссылочные типы

Доступ и индексация

- **Доступ к элементу по номеру (индексу)** array[i] – ***единственная операция*** над массивами
- Индексация элементов массива размера n от 0 до n-1

```
double x[8];
```

Array x

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
16.0	12.0	6.0	8.0	2.5	12.0	14.0	–54.5

Важные моменты

- Адрес массива совпадает с адресом первого элемента (с индексом 0)
- Элементы (ячейки) одного размера (т.к. одного типа)
- Адреса – числа (32бит или 64бит)

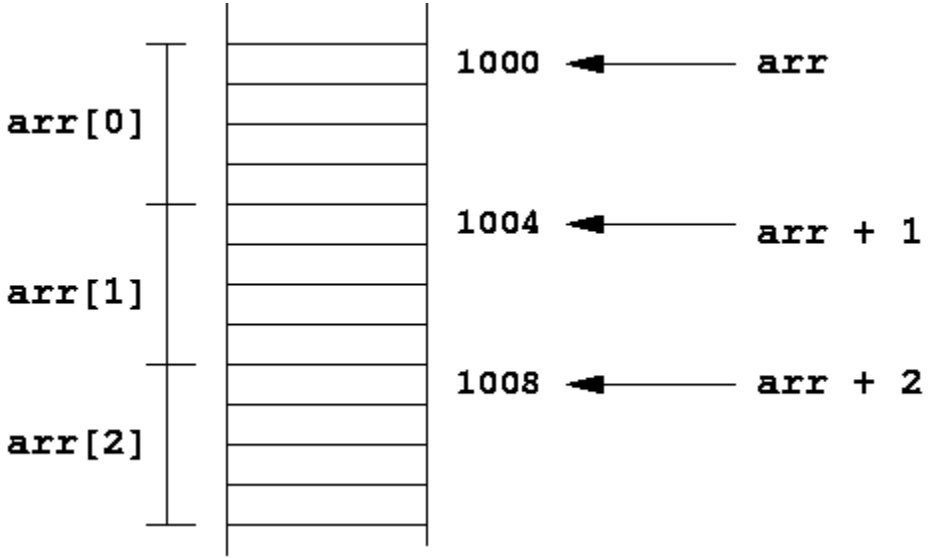
Как получить адрес i-го элемента?

Что стоит за a[i]

- a[i] – обращение к элементу массива a с номером I
- Обратиться нужно по адресу, который легко вычислить:
  - адрес(i) = адрес(0) + i*size
    - адрес(0) = адрес массива = a
    - size – размер элемента массива (типа данных)

Что стоит за оператором a[i]

- какова сложность доступа к любому элементу?



Длина и индексация

- array.Length – длина массива
  - переменная, в которой хранится длина
  - доступ за O(1)
  - нет необходимости хранить самостоятельно
- индексация с нуля
  - длина n
  - первый элемент 0, второй 1, …
  - последний элемент n-1

Расширение массива

- ***Массивы имеют фиксированный размер***
- Если требуется расширить массив, нужно *создать новый массив* большего размер, *скопировать туда содержимое* старого
  - это тяжелая операция и требует памяти
  - так работают списки List (ArrayList, Vector в C++)

Обход массива

- Цикл по индексам, с возможностью изменения элементов

```
int[] array = {1, 2, 3, 4, 5};

for (int i = 0; i < 5; i++)
    array[i] *= 2;
```

Обход массива

- Цикл ***foreach*** без возможности изменения элементов

```
foreach (var x in array)
{
    //x - readonly
    Console.WriteLine(x);
}
```

Инициализация в коде

- Зачем?
- Как узнать размер?

```
int[] a = new int[5] {1, 2, 3, 4, 5};
int[] b = new int[] {1, 2, 3, 4, 5};
int[] c = new[]{1, 2, 3, 4, 5};
var d = new[]{1, 2, 3, 4, 5};
int[] e = {1, 2, 3, 4, 5};
```

Типичные ошибки

- Копирование через присваивание
  - ждем что скопируются значения, но какого типа массив?
  - что скопируется?

```
int[] a = {1, 2, 3, 4, 5};
int[] b = a;
```

- Решение: копирование Array.Copy, a.CopyTo

```
int[] b=new int[a.Length];
Array.Copy(a, b, a.Length);
a.CopyTo(b, 0);
```

- Решение: клонирование

```
int[] b = (int[]) a.Clone();
```

Выход за границы

- Неверная работа с индексами

```
int[] a = {1, 2, 3, 4, 5};
for (int i = 0; i < 10; i++)
    Console.WriteLine(a[i]);
```

- Решение – работа со свойством Length

```
for (int i = 0; i < a.Length; i++)
    Console.WriteLine(a[i]);
```

Вывод на экран

- **Нельзя выводить массив** просто **через ссылку** на него (по умолчанию у составных типов выводится имя типа)

```
int[] a = {1, 2, 3, 4, 5};
Console.WriteLine(a);
```

- **Нужно выводить поэлементно**

```
for (int i = 0; i < a.Length; i++)
    Console.WriteLine(a[i]);
foreach (var x in a)
    Console.WriteLine(x);
```

- **Или** использовать **string.Join** или Array.Foreach и выводить в одну строку

```
Console.WriteLine(string.Join(" ", a));
Array.ForEach(a,Console.WriteLine);
```

Проверка на равенство

- **Наивная проверка не работает** – сравниваются ссылки

```
int[] a = {1, 2, 3, 4, 5};
int[] b = {1, 2, 3, 4, 5};
bool equal= a == b;
```

- **Нужно сравнивать поэлементно**

```
bool equal = a.Length == b.Length;
for (int i = 0; i < a.Length; i++)
    if (a[i] != b[i])
    {
        equal = false;
        break;
    }
```

- **Или использовать SequenceEqual**

```
bool equal = a.SequenceEqual(b);
```

Библиотека (класс) System.Array

- Методы для работы с массивами:
  - создания, копирования
  - изменения (преобразования типов, разворот)
  - выполнения действий над элементами
  - поиска элементов/индексов
  - сортировки

Поиск индекса

```
int[] a = {1, 2, 3, 4, 5};
int index;

//По предикату
index = Array.FindIndex(a, IsPrime);
index = Array.FindLastIndex(a, IsOdd);

//По значению
index = Array.IndexOf(a, 5);
index = Array.LastIndexOf(a, 2);
index = Array.BinarySearch(a, 3);
```

Поиск элементов

```
int[] a = {1, 2, 3, 4, 5};
int value;

//Поиск элемента по предикату
value=Array.Find(a, IsOdd);
value = Array.FindLast(a, IsOdd);
//Поиск всех элементов по предикату
int[] primes;
primes=Array.FindAll(a,IsPrime);
```

Преобразования

```
int[] a = {1, 2, 3, 4, 5};

//Преобразование всех элементов
Array.ConvertAll(a,Convert.ToDouble);

//Копирование массива
int[] b=new int[a.Length];
Array.Copy(a,b,a.Length);
```

Действия

```
int[] a = {1, 2, 3, 4, 5};

//Выполнение операции для каждого элемента
Array.ForEach(a, WriteLine);

//Разворот
Array.Reverse(a);

//Сортировка (Introsort: Quick&Merge&Insertion)
Array.Sort(a);
```

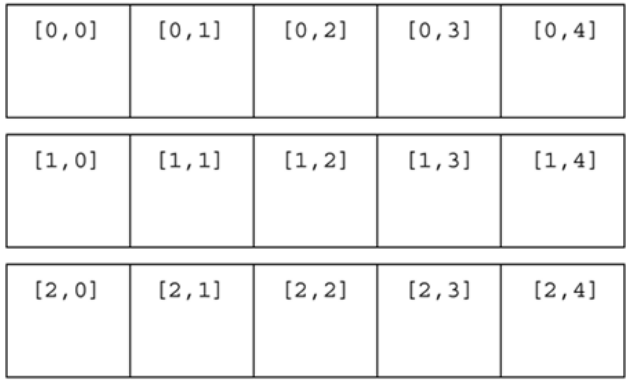
Проверка кванторов ∃, ∀

```
int[] a = {1, 2, 3, 4, 5};

bool check;

//Проверка существования
check=Array.Exists(a, IsPrime);

//Проверки всеобщности
check = Array.TrueForAll(a, IsOdd);
```

Многомерные массивы

Объявление

```
int[,] matrix = new int[5, 3];
```



Размерности

```
int rank = matrix.Rank; // Количество измерений
int n=matrix.GetLength(0); //Количество строк - 5
int m=matrix.GetLength(1); //Количество столбцов - 3
```

Обход многомерных массивов

```
int n=matrix.GetLength(0); //Количество строк - 5
int m=matrix.GetLength(1); //Количество столбцов - 3
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < m; j++)
        //Доступ к элементу – matrix[i,j]
        Console.Write(matrix[i, j] + " ");
    Console.WriteLine();
}

// Обход всех элементов массива
// в естественном порядке.
foreach (var x in matrix)
    Console.WriteLine(x);
```

Инициализация в коде

Почти как в одномерном массиве

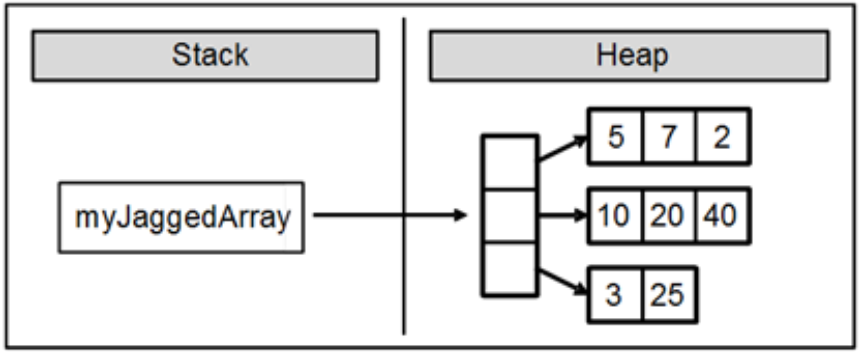
```
int[,] matrix = new int[5, 3]
{
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9},
    {10, 11, 12},
    {13, 14, 15}
};
```

Или даже

```
int[,] matrix1 = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
var matrix2 = new [,] {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
```

Ступенчатые массивы

- Массивы массивов
  инициализация в коде




```
int[][] myJaggedArray= new int[3][];
jagged[0] = new int[3] {5,7,2};
jagged[1] = new int[] {10,20,40};
jagged[2] = new [] {3,25};
```

Обход ступенчатых массивов

```
// Обход циклами for.
for (int i = 0; i < jagged.Length; i++)
{
    for (int j = 0; j < jagged[i].Length; j++)
        Console.WriteLine(jagged[i][j] + " ");
    Console.WriteLine();
}
```

Обход ступенчатых массивов

```
// Обход циклами foreach.
foreach (int[] array in jagged)
{
    foreach (int item in array)
        Console.WriteLine(item + " ");
    Console.WriteLine();
}
```

Трюк

- В ступенчатом массиве хранятся массивы
- Можно по ссылке работать с ними как с обычными массивами

```
// Обход ступенчатого массива в две строки.
foreach (int[] array in jagged)
    Console.WriteLine(string.Join(" ", array));
```

Проверка кванторов

- Проверка существует ли строка в матрице, в которой все элементы - чётные

```
// Короткая запись с методами класса Array
// и лямбда-выражениями для inline предикатов.
bool checkEA=Array.Exists(jagged,
        array => Array.TrueForAll
           (array, item => item%2 == 0));

// Самая короткая запись - с помощью LINQ.
bool checkEA = jagged.Any(array =>
       array.All(item => item%2 == 0));
```

```
bool exists = false;
foreach(var array in jagged)
{
    var forall = true;
    foreach (var item in array)
        if(item%2!=0)
        {
            forall = false;
            break;
        }
    exists = forall;
    if (exists)
        break;
}
```

jagged vs multidimensional

- **Ступенчатые быстрее** многомерных
- Элементы многомерных массивов хранятся в куче последовательно
(один блок в куче)
  - индексация – сложная
  - в CLR одномерные массивы сильно оптимизированы
- Строки ступенчатых массивов хранятся в куче раздельно
(много блоков в куче)
  - ступенчатый массив требует больше памяти

Массив символов

- `char[]` - В C это и называлось строкой.
- C#: Строка – отдельный тип (ссылочный) с большим количеством полезных функций (методов)
  - Есть класс задач, решаемых на текстовых данных

Символы (повторение)

- `char c = 'a';`
- Символы имеют свой код
  - сперва был ASCII (american standard code for information interchange) 7 бит
  - в C# char – 2Байта – 16 битный код Unicode

Escape characters

Управляющие последовательности

- Если в символе \, значит у него особый смысл:
  - \n – перенос строки
  - \t – табуляция
  - \b – отмена предыдущего символа
  - …
- Также \ применяется для вывода некоторых символов
  - \\
  - \”
  - \’

Класс string

- ***Неизменяемая*** строка (immutable)
  `string str="abc";`
- Доступ к символу по индексу: `str[i]`
- Длина строки `str.Length`
- Конкатенация (соединение) +
  – создаёт новую строку
- `string str ="Hello, " + "world";`

Объявление

- Не создаём через конструктор и оператор new (как у другого ссылочного типа)
- `string str = new string("abc"); - ошибка!`
- Используем сокращение
- `string str = "abc";`

Особенности сравнения строк

- При сравнении ссылочных типов операторами == и != сравниваются ссылки (идентичность)
- **Для строки операторы == и !=** перегружены и **сравнивают значения**
- Если нужно сравнить строки <, >, <=, >=, следует использовать string.Compare или метод CompareTo

Обычные и буквальные строки

- Обычная строка (regular string)

```
string columns = "Column 1\tColumn 2\tColumn 3";
//Output: Column 1    Column 2    Column 3
string title = "\"The \u00C6olean Harp\", by Samuel Taylor Coleridge";
//Output: "The Æolean Harp", by Samuel Taylor Coleridge
```

- Буквальная строка (verbatim string)

  игнорируются escape-последовательности

```
string filePath = @"C:\Users\scoleridge\Documents\";
//Output: C:\Users\scoleridge\Documents\
```

Форматирующие строки

- string.Format

```
int a = 2, b = 2;
Console.WriteLine(string.Format("{0}x{1}={2}", a, b, a*b));
// Output: 2x2=4
Console.WriteLine(string.Format("{0:0.0000}", Math.PI));
// Output: 3,1416
Console.WriteLine(string.Format("{0:d}", DateTime.Now));
// Output: 30.06.2016


// Сокращённый вариант.
Console.WriteLine("{0}x{1}={2}", a, b, a*b);
Console.WriteLine("{0:0.0000}", Math.PI);
Console.WriteLine("{0:d}", DateTime.Now);
```

Интерполяция строк

- Шаблонные строки, содержащие выражения
- Используются для создания строк

```
int a = 2, b = 2;
// Сокращённый вариант.
Console.WriteLine($"{a}x{b}={a*b}");
// Output: 2x2=4

Console.WriteLine($"{Math.PI:0.0000}");
// Output: 3,1416

Console.WriteLine($"{DateTime.Now:d}");
// Output: 30.09.2016
```

Методы класса string

В классе string много полезных методов, таких как:

Contains, EndsWith, Insert, Remove, Replace, Split, Join, ToLower, ToUpper, Trim и т.д.

Смотрите справочники

StringBuilder

- Изменяемая последовательность символов
- Позволяет модифицировать подстроки и отдельные символы
- Автоматически расширяет память при добавлении символов или подстрок
  - работает как List<T>: удваивает ёмкость при необходимости

StringBuilder

```
StringBuilder sb = new StringBuilder();
sb.Append("This is the beginning of a sentence, ");
sb.Replace("the beginning of ", "");
sb.Insert(sb.ToString().IndexOf("a ") + 2, "complete ");
sb.Replace(",", ".");
Console.WriteLine(sb.ToString());
// Output: This is a complete sentence
```

Алгоритмы на массивах

- Поиск в неупорядоченном массиве
- Поиск в упорядоченном массивы
- Упорядочивание массива (сортировка)
  - Пузырьком (Bubble)
  - Выбором (Selection)
  - Вставками (Insertion)

Поиск в массиве

```
// Поиск индекса элемента
// с помощью класса Array.
int idx = Array.IndexOf(array, 3);

// Поиск индекса элемента вручную.
// Начинаем с 0;
idx = 0;
for (int i = 0; i < array.Length; i++)
    if (array[i] == 3)
    {
        // Если находим, фиксируем индекс.
        idx = i;
        // Прекращаем поиск.
        break;
    }
```

Поиск в упорядоченном массиве

Идея:

Сверяем элемент в середине массива с искомым

- Если нашли – возвращаем индекс середины
- Если средний элемент больше – **ищем в левой части**
- Если средний элемент меньше – **ищем в правой части**

Бинарный поиск

```
static int BinarySearch(int[]array, int value)
{
    int l = 0;
    int r = array.Length;
    while (r > l)
    {
        int mid = l + (r - l)/2;
        if (array[mid] == value) return mid;
        if (array[mid] > value)
            r = mid - 1;
        else
            l = mid + 1;
    }
    return -1;
}
```

Сортировка пузырьком

```
public static void BubbleSort(int[] array)
{
    for (int i = 0; i < array.Length; i++)
    {
        bool stop = true;
        for (int j = 0; j < array.Length - i - 1; j++)
        {
            if (array[j] > array[j + 1])
            {
                int temp = array[j + 1];
                array[j + 1] = array[j];
                array[j] = temp;
                stop = false;
            }
        }
        if (stop) break;
    }
}
```

16643,9104 Ordered

Сортировка выбором

```
public static void SelectionSort(int[] array)
{
    for (int i = 0; i < array.Length; i++)
    {
        int index = i;
        int min = array[i];
        for (int j = i + 1; j < array.Length; j++)
            if (array[j] < min)
            {
                min = array[j];
                index = j;
            }
        int tmp = array[i];
        array[i] = array[index];
        array[index] = tmp;
    }
}
```

3845,8864 Ordered

Сортировка вставками

```
public static void SwapInsertionSort
                   (int[] array)
{
    for (var counter = 1;
         counter< array.Length;
         ++counter)
    {
        var index = counter - 1;

        while (index >= 0 &&
            array[index + 1] < array[index])
        {
            var temp = array[index];
            array[index] = array[index + 1];
            array[index + 1] = temp;
            --index;
        }
    }
}
```

3407,9721 Ordered

```
private static void MoveInsertionSort
                   (int[] array)
{
    for (int counter = 1;
         counter < array.Length;
         ++counter)
    {
        int index = counter - 1;
        int x = array[counter];
        while (index >= 0 &&
            x < array[index])
        {
            array[index + 1] = array[index];
            --index;
        }
        array[index + 1] = x;
    }
}
```

1872,1584 Ordered

Вопросы?

e-mail: marchenko@it.kfu.ru