LITS

Оптимізація iOS аппи: швідкодія, пам'ять, батарея

Що ми розглянемо

- Стратегії розробки, що призводять до прийнятної продуктивності
- Вузькі місця, які можуть потребувати оптимізації
- Процес оптимізації швидкодії

LITS

Чому це важливо

Чому це важливо

- “Responsiveness”
  - тобто, швидкість реакції на дію користувача
- Економія ресурсів телефону (зокрема, батареї)

Стратегічно

Як забезпечити високу продуктивність

Стратегії

1. Використовувати високорівневе API

   - та дослухатися до порад Apple з використання API

Стратегії

2. Прогнозувати, на яких об’ємах даних буде працювати та чи інша функціональність

- та заздалегідь, на етапі проектування, забезпечити ефекту роботу в тих частинах, де даних багато

- Використовувати структури даних зі швидким доступом

Стратегії

- Вибір правильної структури даних
  - Знати, які операції будуть виконуватися часто, а які - іноді
- Підібрати відповідну колекцію
  - Array, Dictionary, Set
  - їх комбінація, або кастомна реалізація

Стратегії

- Хешовані колекції
  - Dictionary, Set
    - Вміст - об’єкти Equatable, Hashable
    - dictionary[key] - операція, близька до атомарної, за умови вдалої хеш-функції
      - також insert/delete операції

LITS

Стратегії

- Equatable, Hashable
- Хеш-функція:
  - a == b **=>** a.hashValue == b.hashValue

Стратегії

- Хешовані колекції
  - Dictionary, Set
    - dictionary[key] - операція, близька до атомарної, за умови вдалої хеш-функції
- Хеш-функція:
  - a == b **=>** a.hashValue == b.hashValue
    - a.hashValue != b.hashValue **=>**

Стратегії

- Хешовані колекції
  - Dictionary, Set
    - dictionary[key] - операція, близька до атомарної, за умови вдалої хеш-функції
- Хеш-функція:
  - a == b **=>** a.hashValue == b.hashValue
    - a.hashValue != b.hashValue **=>** a != b

Стратегії

- Потреба в повнотекстовому пошуку
  - Зовнішня індексація даних
    - Тобто, додатковий вказівник, що по таких рядках, дані містяться там-то
- База даних з такою індексацією
  - Наприклад, CoreData

Стратегії

2. Прогнозувати, на яких об’ємах даних буде працювати та чи інша функціональність

- **Не треба** оптимізувати роботу з **невеликими** об’ємами даних

Стратегії

3. Знати типові “вузькі місця” (performance bottlenecks)

- та уникати їх заздалегідь, ще на етапі проектування аппи

Стратегії

4. Якщо якась операція за своєю природою є довгою, і це очікувано для юзера

- наприклад
  - скачування великих файлів
  - конвертація документів між форматами
  - завантаження великого масиву даних
- Не треба нічого оптимізувати!

Стратегії

4. Якщо якась операція за своєю природою є довгою, і це очікувано для юзера

- Замість оптимізації - зробити операцію комфортною / корисною
- Показати прогрес виконання операції
- Видавати користувачу часткові результати виконання операцій
  - Ідеал - щоб він міг вже щось з ними робити (e.g. Twitter feed)
- Або, передбачити можливість переходу в інші частини аппи і нотифікувати, коли операція завершиться

Тактика

Як покращити продуктивність

Пам’ятка

1. Продуктивність - це **вимірювана** річ, в **конкретних умовах**

    - (напр., на конкретних даних, на конкретному пристрої)

Пам’ятка

2. Треба знати, яка продуктивність нас задовольнить

- Тобто, оптимізація має **мету**
  - Така операція має виконуватися не більш, як стільки-то мілісекунд
    - На такому-то об’ємі даних
    - На такій серії пристроїв

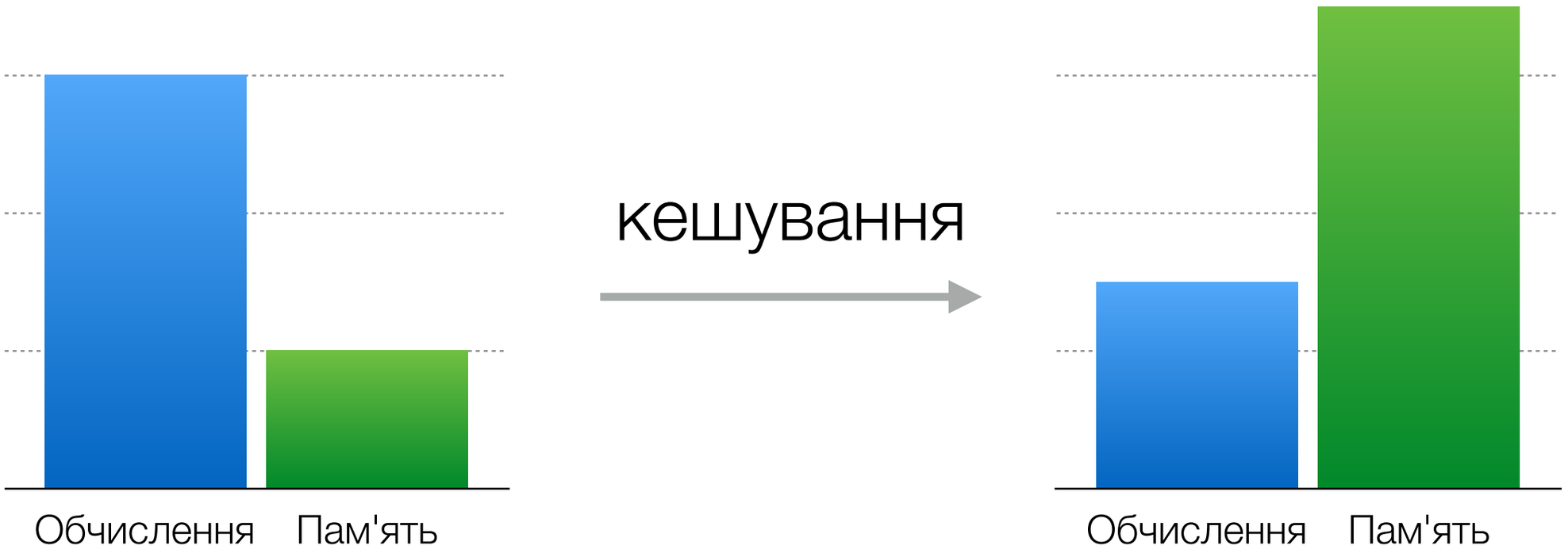
Пам’ятка

2. Треба знати, яка продуктивність нас задовольнить

- Це означає, що ми погоджуємося на зниження швидкодії на більшому об’ємі даних, на слабшому пристрої

Пам’ятка

3. Треба розуміти, що (за винятком помилкового використання API) оптимізація за одним параметром - погіршує інший



Процес оптимізації

1. ???

Процес оптимізації

1. Виміряти

Процес оптимізації

1. Виміряти

- Timestamp:

```
let startTime = NSDate()

operationWeSuspectBeingSlow()

let endTime = NSDate()
let duration = endTime - startTime

debugPrint(“operationWeSuspectBeingSlow: \(duration)”)
```

Процес оптимізації

1. Виміряти

2. ???

Процес оптимізації

1. Виміряти

2. Визначити вузьке місце

Процес оптимізації

1. Виміряти

2. Визначити вузьке місце

   в реальності!!

   якщо “здогадалися” - перевірити, чи це так

LITS

Instruments - Time Profiler

Процес оптимізації

1. Виміряти

2. Визначити вузьке місце

Не треба “оптимізувати” код перш, ніж ми визначимо, що це справді слабке місце

Процес оптимізації

1. Виміряти

2. Визначити вузьке місце

Не треба “оптимізувати” код перш, ніж ми визначимо, що це справді слабке місце

- врата зайвого часу

- менш зрозумілий код

- не впливає на швидкодію реального слабкого місця

Процес оптимізації

1. Виміряти
2. Визначити вузьке місце
3. ???

Процес оптимізації

1. Виміряти
2. Визначити вузьке місце
3. Гіпотеза, що покращить

Процес оптимізації

1. Виміряти
2. Визначити вузьке місце
3. Гіпотеза, що покращить
4. Зробити покращення

“Вузькі місця”

- Робота з файлами
  - Наприклад,
    - завантаження/збереження файлів
    - (плюс навантаження на зчитування формату файла)
    - завантаження/збереження багатьох, хоч і невеликих файлів
    - NSUserDefaults synchronize()

“Вузькі місця”

- Робота з файлами
  - Збереження
    - Лише коли потрібно
  - Завантаження
    - Уникати зайвого завантаження, коли можна закешувати в пам’яті
- В фоновому режимі, коли це має сенс (GCD, NSOperation)

“Вузькі місця”

- Rendering
  - А особливо, передача даних між нашою пам’яттю і відеопам’яттю
  - Обчислення антіаліасінгу
  - Обчислення перетинів кривих Безьє

“Вузькі місця”

- Алгоритмічно: layout і rendering тексту
  - зокрема тому, що гліфи (графічні зображення символів) - це досить складні криві
  - алгоритми layout тексту, перенесення слів (hyphenation) потребують багато проходів

“Вузькі місця”

- Батарея: Доступ до hardware
  - Багато рендерингу, або обчислень на відеокарті
  - Location tracking
  - Motion
  - Доступ до Bluetooth
  - Підсвітка екрану

Уникнення зайвих обчислень

- Кешування
  - збереження в пам’яті результатів складних обчислень для подальшого використання

Уникнення зайвих обчислень

- Кешування
  - збереження в пам’яті результатів складних обчислень для подальшого використання
- Результати вимірювань будуть ділитися на:
  - перший раз виконуємо операцію / побудова чи перебудова кеша
  - подальші операції

Кешування

```
var calculatedValue: Int {



    let result = // Складні обчислення

    return result
}
```

Кешування

```
private var cachedValue: Int?
var calculatedValue: Int {
    if let value = cachedValue {
        return value
    }

    let result = // Складні обчислення
    cachedValue = result
    return result
}
```

Кешування

- “Інвалідація” кеша - при зміні параметрів, від яких ці залежить результат обчислення

```
private var cachedValue: Int?
var calculatedValue: Int {
    if let value = cachedValue {
        return value
    }

    let result = // Складні обчислення
    cachedValue = result
    return result
}
```

Кешування

```
var thumbnailSize = CGSizeMake(40, 40)

private var thumbnailsCache: [String: UIImage] = [:]

func thumbnailForKey(key: String) -> UIImage {
    if let image = thumbnailsCache[key] {
        return image
    }

    let thumbnail = ImageLoader.thumbnailForKey(key,
                                    withSize: thumbnailSize)
    thumbnailsCache[key] = thumbnail
    return thumbnail
}
```

Кешування

```
var thumbnailSize = CGSizeMake(40, 40) {
    didSet {
        thumbnailsCache.removeAll() // Invalidate cache
    }
}

private var thumbnailsCache: [String: UIImage] = [:]

func thumbnailForKey(key: String) -> UIImage {
    if let image = thumbnailsCache[key] {
        return image
    }

    let thumbnail = ImageLoader.thumbnailForKey(key,
                                    withSize: thumbnailSize)
    thumbnailsCache[key] = thumbnail
    return thumbnail
}
```

Кешування - NSCache

```
var thumbnailSize = CGSizeMake(40, 40) {
    didSet {
        thumbnailsCache.removeAllObjects()
    }
}

private var thumbnailsCache: NSCache = NSCache()
```

```
func thumbnailForKey(key: String) -> UIImage {
    if let image = thumbnailsCache.objectForKey(key) {
        return image
    }

    let thumbnail = ImageLoader.thumbnailForKey(key,
                                    withSize: thumbnailSize)
    thumbnailsCache.setObject(thumbnail, forKey:key)
    return thumbnail
}
```

Уникнення багатьох обчислень одночасно

- “Lazy” обчислення / завантаження
- На першому доступі до даних

“Lazy” обчислення

```
var dictionary: [String: [String]] {
    // … Load from file
}
```

LITS

“Lazy” обчислення

```
lazy var dictionary: [String: [String]] = {
    // … Load from file
}()
```

“Lazy” обчислення

```
var _cache: [String: [String]]?

var dictionary: [String: [String]] = {
    if let dictionary = _cache {
        return dictionary
    }

    let result = // … Load from file
    _cache = result
    return result
}
```

“Lazy” обчислення

```
var _cache: [String: [String]]?

var dictionary: [String: [String]] = {
    if let dictionary = _cache {
        return dictionary
    }

    let result = // … Load from file
    _cache = result
    return result
}

func invalidateCache() {
    _cache = nil
}
```

Memory Warning

- UIViewController на didReceiveMemoryWarning

```
override func didReceiveMemoryWarning() {
    super.didReceiveMemoryWarning()


    _cache.removeAll()
    _cachedValue = nil

}
```

Memory Warning

- Не в UIViewController

```
UIApplicationDidReceiveMemoryWarningNotification
```

Memory Warning

- iOS Simulator:
  - Hardware - Simulate Memory Warning

Процес оптимізації

1. Виміряти

2. Визначити вузьке місце

3. Гіпотеза, що покращить

4. Зробили покращення

Далі - ???

Процес оптимізації

1. Виміряти
2. Визначити вузьке місце
3. Гіпотеза, що покращить
4. Зробили покращення
5. Виміряти - в тих самих умовах

Процес оптимізації

1. Виміряти

2. Визначити вузьке місце

3. Гіпотеза, що покращить

4. Зробили покращення

5. Виміряти - в тих самих умовах

6. Відкинути гіпотезу і шукати кращу (3) АБО з’явилося нове слабке місце (2) АБО додати код в проект

Продуктивність - конкретна

1. **Виміряти**
2. Визначити вузьке місце
3. Гіпотеза, що покращить
4. Зробили оптимізацію
5. **Виміряти** - в тих самих умовах
6. Прийняти рішення щодо цієї оптимізації

LITS

Отже

- Знати свою задачу, “середні” умови використання
- Вибір правильних структур даних
- Вимірювання - Аналіз - Покращення - Вимірювання
- Знати API та архітектуру системи - писати в цілому прийнятний код
- В деталях, оптимізувати лише за потреби