Федеральное государственное автономное образовательное
учреждение высшего образования
Университет ИТМО

Факультет инфокоммуникационных технологий

Алагритмы и структуры данных:

Отчёт по лабораторной работе №1: Жадные алгоритмы. Динамическое программирование №2

Выполнил:
Бочкарь Артём Артёмович

Группа: **K33392**
Вариант: **13**

Преподаватели:
Артамонова В. Е.

Санкт-Петербург 2023 г.

Введение:
Программы реализовывал на языке Swift. Ввод и вывод реализовал через консоль по причине того, что использовал web-версию среды разработки Swift: среда. Тесты реализовал прямо в коде решения, результат тестов также выводится с решением.

Задание №1: Максимальная стоимость добычи (0.5 балла)

Вор находит гораздо больше добычи, чем может поместиться в его сумке. Помогите ему найти самую ценную комбинацию предметов, предполагая, что любая часть предмета добычи может быть помещена в его сумку.
Цель - реализовать алгоритм для задачи о дробном рюкзаке.

Формат ввода / входного файла (input.txt).
В первой строке входных дан ных задано целое число n - количество предметов, и W - вместимость сумки. Следующие n строк определяют значения веса и стоимости предметов. В i-ой строке содержатся целые числа pi и wi – стоимость и вес i-го предмета, соответственно.

Формат вывода / выходного файла (output.txt).
Выведите максимальное значение стоимости долей предметов, которые помещаются в сумку. Абсолютная погрешность между ответом вашей программы и оптимальным значением должно быть не более 10−3. Для этого выведите свой ответ как минимум с четырьмя знаками после запятой (иначе ваш ответ, хотя и будет рассчитан правильно, может оказаться неверным из-за проблем с округлением).

Код программы:

```swift
import Foundation

// Структура для хранения информации о предмете
struct Item {
    let weight: Int
    let value: Int
}

// Функция для сортировки предметов по убыванию отношения ценности к весу
func sortByValueToWeightRatio(_ items: [Item]) -> [Item] {
    return items.sorted { (item1, item2) -> Bool in
        let ratio1 = Double(item1.value) / Double(item1.weight)
        let ratio2 = Double(item2.value) / Double(item2.weight)
        return ratio1 > ratio2
    }
}

// Функция для решения задачи о дробном рюкзаке
func fractionalKnapsack(_ items: [Item], capacity: Int) -> Double {
    // Сортируем предметы по убыванию отношения ценности к весу
    let sortedItems = sortByValueToWeightRatio(items)

    // Инициализируем переменную для хранения текущего веса рюкзака
    var currentWeight = 0

    // Инициализируем переменную для хранения максимальной стоимости
```

```
    var maxValue = 0.0

    // Перебираем предметы в отсортированном порядке
    for item in sortedItems {
        // Если текущий вес рюкзака плюс вес предмета не превышает вместимость рюкзака,
        // то добавляем предмет в рюкзак
        if currentWeight + item.weight <= capacity {
            currentWeight += item.weight
            maxValue += Double(item.value)
        }
        // Иначе добавляем в рюкзак только часть предмета, которая помещается
        else {
            let remainingCapacity = capacity - currentWeight
            maxValue += Double(item.value) * Double(remainingCapacity) / Double(item.weight)
            break
        }
    }

    return maxValue
}

// Считываем входные данные
print("Введите количество предметов и вместимость сумки через пробел")
let input = readLine()!.components(separatedBy: " ").map { Int($0)! }
let n = input[0]
let capacity = input[1]

// Создаем массив предметов
print("Введите вес и стоимость предметов через пробел")
var items: [Item] = []
for _ in 0..<n {
    let input = readLine()!.components(separatedBy: " ").map { Int($0)! }
    items.append(Item(weight: input[1], value: input[0]))
}

// Запускаем таймер
let startTime = Date()

// Вычисляем максимальную стоимость
let maxValue = fractionalKnapsack(items, capacity: capacity)

// Останавливаем таймер
let endTime = Date()

// Выводим результат
print("\nМаксимальное значение стоимости доли добычи:")
print(String(format: "%.4f", maxValue))

// Выводим время работы алгоритма
let timeInterval = endTime.timeIntervalSince(startTime)
print("Время работы алгоритма: \(timeInterval) секунд")
```

Результат работы программы:

```
swift /tmp/QIT1Ippuaj.swift
Введите количество предметов и вместимость сумки через пробел
3 50
Введите вес и стоимость предметов через пробел
60 20
100 50
120 30

Максимальное значение стоимости доли добычи:
180.0000
Время работы алгоритма: 0.0005389451980590 82 секунд
```

```
swift /tmp/QIT1Ippuaj.swift
Введите количество предметов и вместимость сумки через пробел
1 10
Введите вес и стоимость предметов через пробел
500 30

Максимальное значение стоимости доли добычи:
166.6667
Время работы алгоритма: 0.0002720355987548828 секунд
```

Задание №2: Заправки (0.5 балла)

Вы собираетесь поехать в другой город, расположенный в d км от вашего родного города. Ваш автомобиль может проехать не более m км на полном баке, и вы начинаете с полным баком. По пути есть заправочные станции на расстояниях stop1, stop2, ..., stopn из вашего родного города. Какое минимальное количество заправок необходимо?

Формат ввода / входного файла (input.txt).
В первой строке содержится d - целое число. Во втройо строке - целое число m. В третьей находится количество заправок на пути - n. И, наконец, в последней строке целые числа через пробел - остановки stop1, stop2, ..., stopn.

Формат вывода / выходного файла (output.txt).

Предполагая, что расстояние между городами составляет d км, автомобиль может проехать не более m км на полном баке, а заправки есть на расстояниях stop1, stop2, ..., stopn по пути, выведите минимально необходимое количество заправок. Предположим, что машина начинает ехать с полным баком. Если до места назначения добраться невозможно, выведите −1.

Код программы:

```
import Foundation


func minRefills(d: Int, m: Int, n: Int, stops: [Int]) -> Int {
    let startTime = Date() // Записываем время начала выполнения алгоритма

    var numRefills = 0
    var currentRefill = 0
    var stops = stops
    stops.insert(0, at: 0)
    stops.append(d)

    while currentRefill <= n {
        let lastRefill = currentRefill

        while currentRefill <= n && stops[currentRefill + 1] - stops[lastRefill] <= m {
            currentRefill += 1
        }

        if currentRefill == lastRefill {
            return -1
        }

        if currentRefill <= n {
            numRefills += 1
        }
    }

    let endTime = Date() // Записываем время окончания выполнения функции
    let executionTime = endTime.timeIntervalSince(startTime) // Вычисляем время выполнения функции
    print("\nВремя выполнени алгоритма: \(executionTime) секунд")

    return numRefills
}

print("Введите полную дистанцию (d):")
if let distanceInput = readLine(), let d = Int(distanceInput) {
    print("Введите расстояние, которое может преодалет автомобиль без остановок (m):")
    if let maxDistanceInput = readLine(), let m = Int(maxDistanceInput) {
        print("Введите количество остановок:")
        if let stopsCountInput = readLine(), let stopsCount = Int(stopsCountInput) {
            var stops = [Int]()
            for i in 1...stopsCount {
                print("Введите километр, на котором находится остановка \(i):")
                if let stopInput = readLine(), let stopPosition = Int(stopInput) {
                    stops.append(stopPosition)
                } else {
                    print("Ошибка при считывании расположения остановки. Пожалуйста, введите допустимое
целочисленное значение.")
```

```
                break
            }
        }

        let stopsNeeded = minRefills(d: d, m: m, n: stopsCount, stops: stops)

        if stopsNeeded == -1 {
            print("Невозможно проехать всю дистанцию.\nОтвет: \(stopsNeeded)")
        } else {
            print("Нужно: \(stopsNeeded) остановок")
        }
    } else {
        print("Ошибка при считывании количества остановок. Пожалуйста, введите допустимое целое значение.")
    }
  } else {
    print("Ошибка при показании максимального расстояния между остановками. Пожалуйста, введите допустимое целочисленное значение.")
  }
} else {
  print("Ошибка при считывании общего расстояния. Пожалуйста, введите допустимое целочисленное значение.")
```

Результат работы программы:

```
swift /tmp/QIT1Ippuaj.swift
Введите полную дистанцию (d):
950
Введите расстояние, которое может преодалет автомобиль без остановок (m):
400
Введите количество остановок:
4
Введите километр, на котором находится остановка 1:
200
Введите километр, на котором находится остановка 2:
375
Введите километр, на котором находится остановка 3:
550
Введите километр, на котором находится остановка 4:
750

Время выполнени алгоритма: 5.507469177246094e-05 секунд
Нужно: 2 остановок
```

```
swift /tmp/QIT1Ippuaj.swift
Введите полную дистанцию (d):
10
Введите расстояние, которое может преодалет автомобиль без остановок (m):
3
Введите количество остановок:
4
Введите километр, на котором находится остановка 1:
1
Введите километр, на котором находится остановка 2:
2
Введите километр, на котором находится остановка 3:
5
Введите километр, на котором находится остановка 4:
9
Невозможно проехать всю дистанцию.
Ответ: -1
```

```
swift /tmp/QIT1Ippuaj.swift
Введите полную дистанцию (d):
200
Введите расстояние, которое может преодалет автомобиль без остановок (m):
250
Введите количество остановок:
2
Введите километр, на котором находится остановка 1:
100
Введите километр, на котором находится остановка 2:
150

Время выполнени алгоритма: 5.507469177246094e-05 секунд
Нужно: 0 остановок
```

Задание №3: Максимальный доход от рекламы (0.5 балла)

У вас есть n объявлений для размещения на популярной интернет-странице. Для каждого объявления вы знаете, сколько рекламодатель готов платить за один клик по этому объявлению. Вы настроили n слотов на своей странице и оценили ожидаемое количество кликов в день для каждого слота. Теперь ваша цель распределить рекламу по слотам, чтобы максимизировать общий доход.

Постановка задачи.
Даны две последовательности a1, a2, ..., an (ai - прибыль за клик по i-му объявлению) и b1, b2, ..., bn (bi - среднее количество кликов в день i-го слота), нужно разбить их на n пар (ai, bj) так, чтобы сумма их произведений была максимальной.

Формат ввода / входного файла (input.txt).
В первой строке содержится целое число n, во второй - последовательность целых чисел a1, a2, ..., an, в третьей - последовательность целых чисел b1, b2, ..., bn.

Формат вывода / выходного файла (output.txt).
Выведите максимальное значение Pni=1 aici , где c1, c2, ..., cn является перестановкой b1, b2, ..., bn.

Код программы:

```
import Foundation

// Считываем входные данные
print("Введите количество объявлений")
let n = Int(readLine()!)!
print("Введите прибыль за клик")
var a = readLine()!.split(separator: " ").map { Int($0)! }
print("Введите среднее количество кликов в день")
var b = readLine()!.split(separator: " ").map { Int($0)! }

// Сортируем массивы в порядке убывания
a.sort(by: >)
b.sort(by: >)

// Запускаем таймер
let startTime = Date()

// Вычисляем максимальный доход
var maxIncome = 0
for i in 0..<n {
    maxIncome += a[i] * b[i]
}

// Останавливаем таймер
let endTime = Date()

// Выводим максимальный доход и время выполнения алгоритма
print("\nМаксимальный доход:")
print(maxIncome)
print("Время выполнения алгоритма:")
print(String(endTime.timeIntervalSince(startTime)) + " секунд")
```

Результат работы программы:

```
swift /tmp/QIT1Ippuaj.swift
Введите количество объявлений
1
Введите прибыль за клик
23
Введите среднее количество кликов в день
39

Максимальный доход:
897
Время выполнения алгоритма:
0.0 секунд
```

```
swift /tmp/QIT1Ippuaj.swift
Введите количество объявлений
3
Введите прибыль за клик
1 3 -5
Введите среднее количество кликов в день
-2 4 1

Максимальный доход:
23
Время выполнения алгоритма:
1.0728836059570312e-06 секунд
```

Задание №4: Сбор подписей (0.5 балла)

Вы несете ответственность за сбор подписей всех жильцов определенного здания. Для каждого жильца вы знаете период времени, когда он или она находится дома. Вы хотите собрать все подписи, посетив здание как можно меньше раз.
Математическая модель этой задачи следующая. Вам дан набор отрезков на прямой, и ваша цель - отметить как можно меньше точек на прямой так, чтобы каждый отрезок содержал хотя бы одну отмеченную точку.

Постановка задачи.
Дан набор из n отрезков [a0, b0], [a1, b1], ..., [an−1, bn−1] с координатами на прямой, найдите минимальное количество m точек такое, чтобы каждый отрезок содержал хотя бы одну точку. То есть найдите набор целых чисел X минимального размера такой, чтобы для любого отрезка [ai, bi] существовала точка x ∈ X такая, что ai ≤ x ≤ bi.

Формат ввода / входного файла (input.txt).
Первая строка входных данных содержит количество отрезков n. Каждая из следующих n строк содержит два целых числа ai и bi (через пробел), определяющие координаты концов i-го отрезка.

Формат вывода / выходного файла (output.txt).
Выведите минимальное количество m точек в первой строке и целочисленные координаты этих m точек (через пробел) во второй строке. Вывести точки можно в любом порядке. Если таких наборов точек несколько, можно вывести любой набор. (Нетрудно видеть, что всегда существует множество точек минимального размера, для которых все координаты точек - целые числа.)

Код программы:

```
import Foundation

// Считываем входные данные
print("Введите количество отрезков")
let n = Int(readLine()!)!
var segments = [(Int, Int)]()
print("Введите координаты концов отрезков")
for _ in 0..<n {
    let line = readLine()!.split(separator: " ").map { Int($0)! }
    segments.append((line[0], line[1]))
}

// Запускаем таймер
let startTime = Date()

// Сортируем отрезки по правому концу
segments.sort { $0.1 < $1.1 }

// Вычисляем минимальное количество точек
var points = [Int]()
var right = -1
for segment in segments {
    if segment.0 > right {
        points.append(segment.1)
        right = segment.1
    }
}

// Останавливаем таймер
let endTime = Date()

// Выводим результат
```

```swift
print("\nМинимальное количество точек")
print(points.count)
print("Целочисленные координаты этих точек")
print(points.map { String($0) }.joined(separator: " "))

// Выводим время работы алгоритма
let timeInterval = endTime.timeIntervalSince(startTime)
print("Время работы алгоритма: \(timeInterval) секунд")
```

Результат работы программы:

```
swift /tmp/QIT1Ippuaj.swift
Введите количество отрезков
3
Введите координаты концов отрезков
1 3
2 5
3 6

Минимальное количество точек
1
Целочисленные координаты этих точек
3
Время работы алгоритма: 0.00039803981781005860 секунд
```

```
swift /tmp/QIT1Ippuaj.swift
Введите количество отрезков
4
Введите координаты концов отрезков
4 7
1 3
2 5
5 6

Минимальное количество точек
2
Целочисленные координаты этих точек
3 6
Время работы алгоритма: 0.00037395954132080008 секунд
```

Задание №5: Максимальное количество призов (0.5 балла)

Вы организуете веселый конкурс для детей. В качестве призового фонда у вас есть n конфет. Вы хотели бы использовать эти конфеты для раздачи k лучшим местам в конкурсе с естественным ограничением, заключающимся в том, что чем выше место, тем больше конфет. Чтобы осчастливить как можно больше детей, вам нужно найти наибольшее значение k, для которого это возможно.

Постановка задачи.
Необходимо представить заданное натуральное число n в виде суммы как можно большего числа попарно различных натуральных чисел. То есть найти максимальное k такое, что n можно записать как a1 + a2 + ... + ak, где a1, ..., ak - натуральные числа и $a_i \neq a_j$ для всех $1 \leq i < j \leq k$.

Формат ввода / входного файла (input.txt).
Входные данные состоят из одного целого числа n.

Формат вывода / выходного файла (output.txt).
В первой строке выведите максимальное число k такое, что n можно представить в виде суммы k попарно различных натуральных чисел. Во второй строке выведите эти k попарно различных натуральных чисел, которые в сумме дают n (если таких представлений много, выведите любое из них).

Код программы:

```swift
import Foundation

func findMaxK(_ n: Int) -> (Int, [Int]) {
    var sum = 0
    var k = 0
    var numbers: [Int] = []

    while sum < n {
        k += 1
        sum += k
        numbers.append(k)
    }

    if sum > n {
        let diff = sum - n
        if diff != k {
            if let index = numbers.firstIndex(of: diff) {
                numbers.remove(at: index)
            }
        }
    }

    return (numbers.count, numbers)
}

func readInput() -> Int? {
    print("Введите число n:")
    if let input = readLine(), let n = Int(input) {
        return n
    } else {
        return nil
    }
}

func printOutput(_ k: Int, _ numbers: [Int]) {
    print("\nКоличество чисел k: \(k)")
    print("Натуральные числа, которые в сумме дают n:")
    print(numbers.map { String($0) }.joined(separator: " "))
}

if let n = readInput() {
    let startTime = Date()
    let result = findMaxK(n)
    let endTime = Date()
    let timeElapsed = endTime.timeIntervalSince(startTime)

    printOutput(result.0, result.1)
    print("Время выполнения программы: \(timeElapsed) секунд")
```

Результат работы программы:

```
swift /tmp/QIT1Ippuaj.swift
Введите число n:
6

Количество чисел k: 3
Натуральные числа, которые в сумме дают n:
1 2 3
Время выполнения программы: 4.208087921142578e-05 секунд
```

```
swift /tmp/QIT1Ippuaj.swift
Введите число n:
8

Количество чисел k: 3
Натуральные числа, которые в сумме дают n:
1 3 4
Время выполнения программы: 5.2928924560546875e-05 секунд
```

```
swift /tmp/QIT1Ippuaj.swift
Введите число n:
2

Количество чисел k: 1
Натуральные числа, которые в сумме дают n:
2
Время выполнения программы: 5.197525024414062e-05 секунд
```

Задание №6: Максимальная зарплата (0.5 балла)

В качестве последнего вопроса успешного собеседования ваш начальник дает вам несколько листков бумаги с цифрами и просит составить из этих цифр наибольшее число. Полученное число будет вашей зарплатой, поэтому вы очень заинтересованы в максимизации этого числа. Как вы можете это сделать?
На лекциях мы рассмотрели следующий алгоритм составления наибольшего числа из

заданных однозначных чисел.

```
def LargestNumber(Digits):
    answer = ''
    while Digits:
        maxDigit = float('-inf')
        for digit in Digits:
            if digit >= maxDigit:
                maxDigit = digit
        answer += str(maxDigit)
        Digits.remove(maxDigit)
    return answer
```

К сожалению, этот алгоритм работает только в том случае, если вход состоит из однозначных чисел. Например, для ввода, состоящего из двух целых чисел 23 и 3 (23 не однозначное число!) возвращается 233, в то время как наибольшее число на самом деле равно 323. Другими словами, использование наибольшего числа из входных данных в качестве первого числа не является безопасным ходом.
Ваша цель в этой задаче – настроить описанный выше алгоритм так, чтобы он работал не только с однозначными числами, но и с произвольными положительными целыми числами.

Постановка задачи.
Составить наибольшее число из набора целых чисел.

Формат ввода / входного файла (input.txt).
Первая строка входных данных содержит целое число n. Во второй строке даны целые числа a1, a2, ..., an.

Формат вывода / выходного файла (output.txt).
Выведите наибольшее число, которое можно составить из a1, a2, ..., an.

Код программы:

```
import Foundation

// Чтение ввода с консоли
print("Введите кол-во чисел")
let n = Int(readLine()!)!
print("Введите сами числа через пробел")
var numbers = readLine()!.split(separator: " ").map { Int($0)! }

// Функция для сравнения двух чисел
func compare(a: Int, b: Int) -> Bool {
    // преобразование Интов в строки
    let aString = String(a)
    let bString = String(b)
```

```swift
    // Создание двух разных строк путем соединения двух строк в разном порядке
    let ab = aString + bString
    let ba = bString + aString

    // Сравнивание двух новых строк
    return ab > ba
}

// Запуск таймера
let startTime = Date()

// Сортировка чисел в порядке убывания
numbers.sort(by: compare)

// Остановка таймера
let endTime = Date()

// Вычисление затраченного на работу алгоритма времени
let elapsedTime = endTime.timeIntervalSince(startTime)

// Вывод наибольшего числа и затраченного времени
print("\nНаибольшее число из ваших чисел:")
print(numbers.map { String($0) }.joined())
print("Затраченное время: \(elapsedTime) секунд")
```

Результат работы программы:

```
swift /tmp/QIT1Ippuaj.swift
Введите кол-во чисел
2
Введите сами числа через пробел
21 2

Наибольшее число из ваших чисел:
221
Затраченное время: 1.8000602722167 97e-05 секунд
```

```
swift /tmp/QIT1Ippuaj.swift
Введите кол-во чисел
3
Введите сами числа через пробел
23 39 92

Наибольшее число из ваших чисел:
923923
Затраченное время: 2.5033950805664062e-05 секунд
```

Задание №8: Расписание лекций (1 балл)

Постановка задачи.
Вы наверно знаете, что в ИТМО лекции читают одни из лучших преподаватели мира. К сожалению, лекционных аудиторий у нас не так уж и много, особенно на Биржевой, поэтому каждый преподаватель составил список лекций, которые он хочет прочитать студентам. Чтобы студенты, в начале февраля, увидели расписание лекций, необходимо его составить прямо сейчас. И без вас нам здесь не справиться. У нас есть список заявок от преподавателей на лекции для одной из аудиторий. Каждая заявка представлена в виде временного интервала [si , fi) - время начала и конца лекции. Лекция считается открытым интервалом, то есть какая-то лекция может начаться в момент окончания другой, без перерыва. Необходимо выбрать из этих заявок такое подмножество, чтобы суммарно выполнить максимальное количество заявок. Учтите, что одновременно в лекционной аудитории, конечно же, может читаться лишь одна лекция.

Формат ввода / входного файла (input.txt).
В первой строке вводится натуральное число N - общее количество заявок на лекции. Затем вводится N строк с описаниями заявок - по два числа в каждом si и fi для каждой лекции i. Гарантируется, что si < fi. Время начала и окончания лекции - натуральные числа, не превышают 1440 (в минутах с начала суток).

Формат вывода / выходного файла (output.txt).
Выведите одно число – максимальное количество заявок на проведение лекций, которые можно выполнить.

Код программы:
```
import Foundation

// Чтение входных данных
print("Введите количество лекций")
let n = Int(readLine()!)!
```

```swift
print("Введите интервалы лекций")
var intervals = [(Int, Int)]()
for _ in 0..<n {
    let line = readLine()!.split(separator: " ").map { Int($0)! }
    intervals.append((line[0], line[1]))
}

// Измерение времени выполнения алгоритма
let startTime = Date()

// Сортировка интервалов по времени окончания
intervals.sort { $0.1 < $1.1 }

// Вычисление максимального количества лекций
var maxCount = 0
var currentEnd = 0
for interval in intervals {
    if interval.0 >= currentEnd {
        maxCount += 1
        currentEnd = interval.1
    }
}

// Вывод результата
print("\nМаксимальное количество заявок:")
print(maxCount)

// Измерение времени выполнения алгоритма
let endTime = Date()
let timeInterval = endTime.timeIntervalSince(startTime)
print("Время выполнения алгоритма: \(timeInterval) секунд")
```

Результат работы программы:

```
swift /tmp/QIT1Ippuaj.swift
Введите количество лекций
1
Введите интервалы лекций
5 10

Максимальное количество заявок:
1
Время выполнения алгоритма: 0.0007909536361694336 секунд
```

```
swift /tmp/QIT1Ippuaj.swift
Введите количество лекций
3
Введите интервалы лекций
1 5
2 3
3 4

Максимальное количество заявок:
2
Время выполнения алгоритма: 0.00051701068878173 83 секунд
```

Задание №11: Максимальное количество золота (1 балл)

Вам дается набор золотых слитков, и ваша цель - набрать как можно больше золота в свою сумку. Существует только одна копия каждого слитка, и для каждого слитка вы можете либо взять его, либо нет (т.е. вы не можете взять часть слитка).

Постановка задачи.
Даны n золотых слитков, найдите максимальный вес золота, который поместится в сумку вместимостью W.

Формат ввода / входного файла (input.txt).
Первая строка входных данных содержит вместимость W сумки и количество n золотых слитков. В следующей строке записано n целых чисел w0, w1, ..., wn−1, определяющие вес золотых слитков.

Формат вывода / выходного файла (output.txt).
Выведите максимальный вес золота, который поместится в сумку вместимости W.
Код программы:

```swift
import Foundation

// Читаем входные данные
print("Введите вместимость сумки W и количество золотых слитков n:")
let input = readLine()!.components(separatedBy: " ")
let W = Int(input[0])!
let n = Int(input[1])!
print("Введите вес каждого золотого слитка:")
let weights = readLine()!.components(separatedBy: " ").map { Int($0)! }

// Создаем таблицу для хранения результатов
var dp = Array(repeating: Array(repeating: 0, count: W + 1), count: n + 1)

// Запускаем таймер
let startTime = Date()
```

```
// Заполняем таблицу
for i in 1...n {
    let weight = weights[i - 1]
    for j in 0...W {
        if weight <= j {
            dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - weight] + weight)
        } else {
            dp[i][j] = dp[i - 1][j]
        }
    }
}

// Останавливаем таймер
let endTime = Date()

// Выводим результат
print("\nМаксимально возможный вес:")
print(dp[n][W])

// Выводим время работы алгоритма
let timeInterval = endTime.timeIntervalSince(startTime)
print("Время работы алгоритма: \(timeInterval) секунд")
```

Результат работы программы:

```
swift /tmp/QIT1Ippuaj.swift
Введите вместимость сумки W и количество золотых слитков n:
10 3
Введите вес каждого золотого слитка:
1 4 8

Максимально возможный вес:
9
Время работы алгоритма: 2.09808349609375e-05 секунд
```

Задание №13: Сувениры (1.5 балла)

Вы и двое ваших друзей только что вернулись домой после посещения разных стран. Теперь вы хотели бы поровну разделить все сувениры, которые все трое накупили.

Формат ввода / входного файла (input.txt).
В первой строке дано целое число n. Во второй строке даны целые числа v1, v2, ..., vn, разделенные пробелами.

Формат вывода / выходного файла (output.txt).
Выведите 1, если можно разбить v1, v2, ..., vn на три подмножества с одинаковыми

суммами и 0 в противном случае.

Код программы:

```
import Foundation

// Считываем количество сувениров
print("Введите количество сувениров")
let n = Int(readLine()!)!

// Считываем значения сувениров
print("Введите стоимости каждого сувенира ")
let souvenirs = readLine()!.split(separator: " ").map { Int($0)! }

// Проверяем, можно ли разделить сувениры поровну
var sum = 0
for souvenir in souvenirs {
    sum += souvenir
}

print("\nОтвет:")
if sum % 3 != 0 {
    print(0)
} else {
    let targetSum = sum / 3
    var subsetSums = [0, 0, 0]
    var used = Array(repeating: false, count: n)

    // Запускаем таймер
    let startTime = Date()

    func backtrack(_ index: Int) {
        if index == n {
            if subsetSums[0] == targetSum && subsetSums[1] == targetSum && subsetSums[2] == targetSum {
                print(1)

                // Останавливаем таймер и выводим время выполнения
                let endTime = Date()
                let timeElapsed = endTime.timeIntervalSince(startTime)
                print("Время выполнения: \(timeElapsed) секунд")

                exit(0)
            }
            return
        }

        for i in 0..<3 {
            if used[index] {
                continue
            }

            used[index] = true
            subsetSums[i] += souvenirs[index]
            backtrack(index + 1)
            subsetSums[i] -= souvenirs[index]
            used[index] = false
        }
    }
```

```
backtrack(0)
print(0)
```

Результат работы программы:

```
swift /tmp/QIT1Ippuaj.swift
Введите количество сувениров
4
Введите стоимости каждого сувенира
3 3 3 3

Ответ:
0
```

```
swift /tmp/QIT1Ippuaj.swift
Введите количество сувениров
1
Введите стоимости каждого сувенира
40

Ответ:
0
```

```
swift /tmp/QIT1Ippuaj.swift
Введите количество сувениров
11
Введите стоимости каждого сувенира
17 59 34 57 17 23 67 1 18 2 59

Ответ:
1
Время выполнения: 0.0030149221420288086 секунд
```

```
swift /tmp/QIT1Ippuaj.swift
Введите количество сувениров
13
Введите стоимости каждого сувенира
1 2 3 4 5 5 7 7 8 10 12 19 25

Ответ:
1
Время выполнения: 0.006282925605773926 секунд
```

Задание №15: Удаление скобок (2 балла)

Постановка задачи.
Дана строка, составленная из круглых, квадратных и фигурных скобок. Определите, какое наименьшее количество символов необходимо удалить из этой строки, чтобы оставшиеся символы образовывали правильную скобочную последовательность.

Формат ввода / входного файла (input.txt).
Во входном файле записана строка, состоящая из s символов: круглых, квадратных и фигурных скобок (), [], {}. Длина строки не превосходит 100 символов.

Формат вывода / выходного файла (output.txt).
Выведите строку максимальной длины, являющейся правильной скобочной последовательностью, которую можно получить из исходной строки удалением некоторых символов.

Код программы:

```swift
import Foundation

// Функция для проверки правильности скобочной последовательности
func isBalanced(_ str: String) -> Bool {
    var stack = [Character]()
    for char in str {
        switch char {
        case "(":
            stack.append(")")
        case "[":
            stack.append("]")
        case "{":
            stack.append("}")
        case ")", "]", "}":
            if stack.isEmpty || stack.removeLast() != char {
                return false
            }
        default:
            continue
        }
    }
```

```
    return stack.isEmpty
}

// Функция для удаления минимального количества символов, чтобы получить правильную скобочную последовательность
func minDeletions(_ str: String) -> String {
    var dp = Array(repeating: -1, count: str.count)

    // Заполнение таблицы динамического программирования
    for i in 0..<str.count {
        if str[str.index(str.startIndex, offsetBy: i)] == "(" || str[str.index(str.startIndex, offsetBy: i)] == "[" || str[str.index(str.startIndex, offsetBy: i)] == "{" {
            dp[i] = 1
        } else {
            for j in 0..<i {
                if dp[j] != -1 && (str[str.index(str.startIndex, offsetBy: j)] == "(" && str[str.index(str.startIndex, offsetBy: i)] == ")") || (str[str.index(str.startIndex, offsetBy: j)] == "[" && str[str.index(str.startIndex, offsetBy: i)] == "]") || (str[str.index(str.startIndex, offsetBy: j)] == "{" && str[str.index(str.startIndex, offsetBy: i)] == "}") {
                    dp[i] = max(dp[i], dp[j] + 2)
                }
            }
        }
    }

    // Восстановление правильной скобочной последовательности
    var result = ""
    var i = str.count - 1
    while i >= 0 {
        if dp[i] != -1 {
            for j in 0..<i {
                if dp[j] != -1 && (str[str.index(str.startIndex, offsetBy: j)] == "(" && str[str.index(str.startIndex, offsetBy: i)] == ")") || (str[str.index(str.startIndex, offsetBy: j)] == "[" && str[str.index(str.startIndex, offsetBy: i)] == "]") || (str[str.index(str.startIndex, offsetBy: j)] == "{" && str[str.index(str.startIndex, offsetBy: i)] == "}") {
                    result = String(str[str.index(str.startIndex, offsetBy: j)]) + result + String(str[str.index(str.startIndex, offsetBy: i)])
                    i = j - 1
                    break
                }
            }
        }
        i -= 1
    }
    return result
}

// Получение входной строки из консоли
print("Введите строку скобок:")
let input = readLine()!

// Измерение времени выполнения программы
let startTime = Date()
if isBalanced(input) {
    print("Строка уже является правильной скобочной последовательностью.")
} else {
    // Удаление минимального количества символов
    let result = minDeletions(input)

    // Вывод результата в консоль
```

```
    print("\nМинимальное количество удалений:", input.count - result.count)
    print("Правильная скобочная последовательность:", result)
}
let endTime = Date()
let timeInterval = endTime.timeIntervalSince(startTime)

// Вывод времени выполнения программы
print("\nВремя выполнения программы:", timeInterval, "секунд")
```

Результат работы программы:

```
swift /tmp/QIT1Ippuaj.swift
Введите строку скобок:
([])

Минимальное количество удалений: 2
Правильная скобочная последовательность: []

Время выполнения программы: 0.0009258985519409 18 секунд
```

Задание №16: Продавец (2 балла)

Постановка задачи.
Продавец техники хочет объехать n городов, посетив каждый из них ровно один раз. Помогите ему найти кратчайший путь.

Формат ввода / входного файла (input.txt).
Первая строка входного файла содержит натуральное число n – количество городов. Следующие n строк содержат по n чисел – длины путей между городами. В i-й строке j-е число – ai,j – это расстояние между городами i и j.

Формат вывода / выходного файла (output.txt).
В первой строке выходного файла выведите длину кратчайшего пути. Во второй строке выведите n чисел – порядок, в котором нужно посетить города.

Код программы:

```
import Foundation

// Читаем входные данные
print("Введите количество городов")
let n = Int(readLine()!)!

var distances = Array(repeating: Array(repeating: 0, count: n), count: n)

print("Введите расстояние между городами через пробел")
for i in 0..<n {
    let line = readLine()!.split(separator: " ").map { Int($0)! }
```

```
    for j in 0..<n {
        distances[i][j] = line[j]
    }
}

// Находим кратчайший гамильтонов цикл, используя алгоритм ближайшего соседа
var visited = Array(repeating: false, count: n)
var tour = [Int]()
var currentCity = 0
var totalDistance = 0

// Запускаем таймер
let startTime = Date()

while visited.filter({ $0 == true }).count < n {
    // Поиск ближайшего непосещенного города
    var nearestCity = -1
    var nearestDistance = Int.max
    for i in 0..<n {
        if !visited[i] && distances[currentCity][i] < nearestDistance {
            nearestCity = i
            nearestDistance = distances[currentCity][i]
        }
    }

    // Добавление ближайший город в тур и отметка его посещенния
    tour.append(nearestCity)
    visited[nearestCity] = true

    // Обновление текущего города и общего расстояния
    currentCity = nearestCity
    totalDistance += nearestDistance
}

// Останавливаем таймер
let endTime = Date()

// Если мы хотим вернуться обратно в первый город, тогда нужно раскомментировать строку ниже
// totalDistance += distances[tour.last!][tour.first!]

// Print the total distance and the tour
print("\nОбщее расстояние:")
print(totalDistance)
print("Маршрут:")
print(tour.map { String($0 + 1) }.joined(separator: " "))

// Выводим время работы алгоритма
let timeInterval = endTime.timeIntervalSince(startTime)
print("\nВремя работы алгоритма: \(timeInterval) секунд")
```

Результат работы программы:

```
swift /tmp/QIT1Ippuaj.swift
Введите количество городов
5
Введите расстояние между городами через пробел
0 183 163 173 181
183 0 165 172 171
163 165 0 189 302
173 172 189 0 167
181 171 302 167 0

Общее расстояние:
666
Маршрут:
1 3 2 5 4

Время работы алгоритма: 0.00019907951354980 47 секунд
```

Задание №19: Произведение матриц (3 балла)

Постановка задачи.
В произведении последовательности матриц полностью расставлены скобки, если выполняется один из следующих пунктов:

– Произведение состоит из одной матрицы.
– Оно является заключенным в скобки произведением двух произведений с полностью расставленными скобками.

Полная расстановка скобок называется оптимальной, если количество операций, требуемых для вычисления произведения, минимально.
Требуется найти оптимальную расстановку скобок в произведении последовательности матриц.

Формат ввода / входного файла (input.txt).
В первой строке входных данных содержится целое число n - количество матриц. В n следующих строк содержится по два целых числа ai и bi - количество строк и столбцов в i-той матрице соответственно. Гарантируется, что bi = ai+1 для любого $1 \le i \le n - 1$.

Формат вывода / выходного файла (output.txt).
Выведите оптимальную расстановку скобок. Если таких расстановок несколько, выведите любую.

Код программы:

```swift
import Foundation

// Структура для хранения информации о матрице
struct Matrix {
    let rows: Int
    let columns: Int
}

// Функция для вычисления оптимального заключения в круглые скобки
func optimalParenthesization(_ matrices: [Matrix]) -> (String, Int) {
    // Создание таблицы для хранения результатов
    var dp = Array(repeating: Array(repeating: 0, count: matrices.count), count: matrices.count)

    // Создание таблицы для хранения времени, затраченного на вычисление каждой подзадачи
    var time = Array(repeating: Array(repeating: 0.0, count: matrices.count), count: matrices.count)

    // Инициализация таблицы
    for i in 0..<matrices.count {
        dp[i][i] = 0
        time[i][i] = 0.0
    }

    // Вычисление оптимальной формы заключения в круглые скобки
    for l in 1..<matrices.count {
        for i in 0..<matrices.count - l {
            var minCost = Int.max
            var minTime = Double.greatestFiniteMagnitude

            for j in i..<i + l {
                let cost = dp[i][j] + dp[j + 1][i + l] + matrices[i].rows * matrices[j].columns * matrices[i + l].columns
                let t = time[i][j] + time[j + 1][i + l]
                if cost < minCost || (cost == minCost && t < minTime) {
                    minCost = cost
                    minTime = t
                }
            }

            dp[i][i + l] = minCost
            time[i][i + l] = minTime
        }
    }

    // Возвращение оптимальной формы заключения в круглые скобки
    return (restoreParenthesization(matrices, dp, 0, matrices.count - 1), Int(time[0][matrices.count - 1]))
}

// Функция для возвращения оптимальной постановки в скобки
func restoreParenthesization(_ matrices: [Matrix], _ dp: [[Int]], _ i: Int, _ j: Int) -> String {
    if i == j {
        return "A\(i + 1)"
    }

    // Использование значения minIndex, для нахождения оптимальной формы заключения в круглые
скобки
    let minIndex = findMinIndex(dp, i, j)

    return "(\(restoreParenthesization(matrices, dp, i, minIndex)))(\(restoreParenthesization(matrices, dp,
minIndex + 1, j)))"
```

```
}

// Функция для определения индекса точки разделения минимальной стоимости
func findMinIndex(_ dp: [[Int]], _ i: Int, _ j: Int) -> Int {
    for k in i..<j {
        if dp[i][j] == dp[i][k] + dp[k + 1][j] + matrices[i].rows * matrices[k].columns * matrices[j].columns {
            return k
        }
    }

    return -1
}

// Получение входных данных
print("Введите количество матриц")
let n = Int(readLine()!)!
var matrices = [Matrix]()

print("Введите количество строк и столбцов в матрицах")
for _ in 0..<n {
    let line = readLine()!.split(separator: " ").map { Int($0)! }
    matrices.append(Matrix(rows: line[0], columns: line[1]))
}

// Вычисление оптимальной формы заключения в круглые скобки
let (result, time) = optimalParenthesization(matrices)

// Вывод результата в консоль
print("\nОптимальная расстановка в скобках:")
print("(" + result + ")")
print("Время работы программы: \(time) секунд")
```

Результат работы программы:

```
swift /tmp/QIT1Ippuaj.swift
Введите количество матриц
3
Введите количество строк и столбцов в матрицах
1 50
50 90
90 20

Оптимальная расстановка в скобках:
(((A1)(A2))(A3))
Время работы программы: 0 секунд
```

Вывод:

Решил задачи: 1(0.5б), 2(0.5б), 3(0.5б), 4(0.5б), 5(0.5б), 6(0.5б), 8(1б), 11(1б), 13(1.5б), 15(2б), 16(2б), 19(3б). В сумме 13.5 баллов