



Алгоритмы и структуры данных

Лекция 10. Динамическое программирование.

(с) Глухих Михаил Игоревич, glukhikh@mail.ru

Динамическое программирование: идея

- ▶ Развитие идеи декомпозиции
- ▶ Задача разбивается на подзадачи...

Динамическое программирование: идея

- Развитие идеи декомпозиции
- Задача разбивается на подзадачи...
- + уже решённые подзадачи запоминаются (мемоизация)

Динамическое программирование: идея

- Развитие идеи декомпозиции
- Задача разбивается на подзадачи...
- + уже решённые подзадачи запоминаются (мемоизация)
- Мемоизация – сохранение результатов выполнения функций для предотвращения повторных вычислений
- Примеры – см. lesson6

Простой пример: числа Фибоначчи

➔ 0, 1, 1, 2, 3, 5, 8, 13, ...

Простой пример: числа Фибоначчи

- 0, 1, 1, 2, 3, 5, 8, 13, ...
- $F(0) = 0, F(1) = 1, F(N+2) = F(N+1) + F(N)$

Простой пример: числа Фибоначчи

- 0, 1, 1, 2, 3, 5, 8, 13, ...
- $F(0) = 0, F(1) = 1, F(N+2) = F(N+1) + F(N)$

- Рекурсивное вычисление

```
fun fib(n: Int): Int =  
    if (n < 2) n else fib(n - 1) + fib(n - 2)
```

Простой пример: числа Фибоначчи

- ▶ 0, 1, 1, 2, 3, 5, 8, 13, ...
- ▶ $F(0) = 0, F(1) = 1, F(N+2) = F(N+1) + F(N)$

- ▶ Рекурсивное вычисление

```
fun fib(n: Int): Int =  
    if (n < 2) n else fib(n-1) + fib(n-2)
```

fib(4) = fib(3) + fib(2) = fib(2) + fib(1) + fib(1) + fib(0) =
fib(1) + fib(0) + fib(1) + fib(1) + fib(0): 9 invocations

fib(6) = fib(5) + fib(4) = fib(4) + fib(3) + fib(4): 2 + 9 + 9 + 5 = 25 invocations

Простой пример: числа Фибоначчи

- 0, 1, 1, 2, 3, 5, 8, 13, ...
- $F(0) = 0, F(1) = 1, F(N+2) = F(N+1) + F(N)$
- Рекурсивное вычисление

```
fun fib(n: Int): Int =  
    if (n < 2) n else fib(n-1) + fib(n-2)
```
- Трудоёмкость = ???

Простой пример: числа Фибоначчи

- 0, 1, 1, 2, 3, 5, 8, 13, ...
- $F(0) = 0$, $F(1) = 1$, $F(N+2) = F(N+1) + F(N)$
- Рекурсивное вычисление

```
fun fib(n: Int): Int =  
    if (n < 2) n else fib(n-1) + fib(n-2)
```
- Трудоёмкость = $O(\text{fib}(n))$

Простой пример: числа Фибоначчи

- ▶ 0, 1, 1, 2, 3, 5, 8, 13, ...
- ▶ $F(0) = 0, F(1) = 1, F(N+2) = F(N+1) + F(N)$
- ▶ Мемоизация

```
private val storage = hashMapOf(0 to 0, 1 to 1)
```

```
fun fib(n: Int): Int {  
    val memo = storage[n]  
    return if (memo != null) memo  
    else {  
        val result = fib(n-1) + fib(n-2)  
        storage[n] = result  
        result  
    }  
}
```

Простой пример: числа Фибоначчи

➤ 0, 1, 1, 2, 3, 5, 8, 13, ...

➤ $F(0) = 0$, $F(1) = 1$, $F(N+2) = F(N+1) + F(N)$

➤ Мемоизация

```
private val storage = hashMapOf(0 to 0, 1 to 1)
fun fib(n: Int): Int =
    storage.getOrPut(n) { fib(n-1) + fib(n-2) }
```

➤ См. пример `lesson6.fibonacci`

Простой пример: числа Фибоначчи

- 0, 1, 1, 2, 3, 5, 8, 13, ...
- $F(0) = 0, F(1) = 1, F(N+2) = F(N+1) + F(N)$

- Мемоизация

```
private val storage = hashMapOf(0 to 0, 1 to 1)
fun fib(n: Int): Int =
    storage.getOrPut(n) { fib(n-1) + fib(n-2) }
```

- См. пример `lesson6.Fibonacci`
- Трудоемкость = ???

Простой пример: числа Фибоначчи

- 0, 1, 1, 2, 3, 5, 8, 13, ...
- $F(0) = 0$, $F(1) = 1$, $F(N+2) = F(N+1) + F(N)$

- Мемоизация

```
private val storage = hashMapOf(0 to 0, 1 to 1)
fun fib(n: Int): Int =
    storage.getOrPut(n) { fib(n-1) + fib(n-2) }
```

- См. пример `lesson6.Fibonacci`
- Трудоёмкость = $O(N)$

Динамическое программирование: нисходящий или восходящий подход

- Нисходящий вариант: рекурсивная реализация + мемоизация уже вычисленных результатов

Динамическое программирование: нисходящий или восходящий подход

- Восходящий вариант: последовательное (в цикле) продвижение от размерности 0 до размерности N , опять-таки с запоминанием результатов

Динамическое программирование: применимость

- ▶ Зависимость решения задачи от подзадач

Динамическое программирование: применимость

- Зависимость решения задачи от подзадач
- Перекрывающиеся подзадачи: снова и снова решаем одно и то же

Задача о разрезании стержня

- ▶ Есть стержень целочисленной длины N

Задача о разрезании стержня

- Есть стержень целочисленной длины N
- Его необходимо разрезать на несколько кусков (также целочисленной длины)...

Задача о разрезании стержня

- Есть стержень целочисленной длины N
- Его необходимо разрезать на несколько кусков (также целочисленной длины)...
- ... с тем, чтобы получить максимальную прибыль от их продажи

- Цены кусков стержня заданной длины представлены таблицей (или, в общем случае – функцией)

Задача о разрезании стержня

- Есть стержень целочисленной длины N
- Его необходимо разрезать на несколько кусков (также целочисленной длины)...
- ... с тем, чтобы получить максимальную прибыль от их продажи

- $\text{Income}(N) = \text{Max}(\text{Cost}(N), \text{Max}(\text{Cost}(i) + \text{Income}(N-i)))$
- $\text{Income}(1) = \text{Cost}(1)$
- $\text{Income}(0) = 0$

Задача о разрезании стержня -- решение

- Есть стержень целочисленной длины N
- Его необходимо разрезать на несколько кусков (также целочисленной длины)...
- ... с тем, чтобы получить максимальную прибыль от их продажи

- Аналогично числам Фибоначчи – но запоминаем мы $\text{Income}(i)$
- См. пример `rod/Cut.kt`
- Трудоёмкость = ???

Задача о разрезании стержня -- решение

- Есть стержень целочисленной длины N
- Его необходимо разрезать на несколько кусков (также целочисленной длины)...
- ... с тем, чтобы получить максимальную прибыль от их продажи

- Аналогично числам Фибоначчи – но запоминаем мы $\text{Income}(i)$
- См. пример `rod/Cut.kt`
- Трудоёмкость = $O(N^2)$

Задача о ранце

- ▶ Есть рюкзак грузоподъёмностью L
- ▶ Есть набор из M предметов ценностью C_i и весом W_i
- ▶ Необходимо выбрать, какие из них брать:
 $\text{Sum}(W_i) \leq L, \text{Sum}(C_i) \rightarrow \text{Max}$

Задача о ранце -- варианты

- Есть рюкзак грузоподъёмностью L
- Есть набор из M предметов ценностью C_i и весом W_i
- Необходимо выбрать, какие из них брать:
 $\text{Sum}(W_i) \leq L, \text{Sum}(C_i) \rightarrow \text{Max}$
- Задача 0/1 – каждый предмет либо берётся, либо нет

Задача о ранце -- варианты

- Есть рюкзак грузоподъёмностью L
- Есть набор из M предметов ценностью C_i и весом W_i
- Необходимо выбрать, какие из них брать:
 $\text{Sum}(W_i) \leq L, \text{Sum}(C_i) \rightarrow \text{Max}$

- Задача 0/1 – каждый предмет либо берётся, либо нет
- Ограниченная задача – каждый предмет можно брать не более N раз

Задача о ранце -- варианты

- Есть рюкзак грузоподъёмностью L
- Есть набор из M предметов ценностью C_i и весом W_i
- Необходимо выбрать, какие из них брать:
 $\text{Sum}(W_i) \leq L, \text{Sum}(C_i) \rightarrow \text{Max}$

- Задача 0/1 – каждый предмет либо берётся, либо нет
- Ограниченная задача – каждый предмет можно брать не более N раз
- Неограниченная задача – каждый предмет можно брать любое количество раз

0/1 задача о ранце – повторяющиеся задачи

- Вначале у нас есть рюкзак грузоподъёмностью L и набор из M предметов...

0/1 задача о ранце – повторяющиеся задачи

- Вначале у нас есть рюкзак грузоподъёмностью L и набор из M предметов...
- ... предположим, мы положили туда предмет $\#M$, теперь у нас есть рюкзак грузоподъёмностью $L - W_M$ и $M - 1$ предметов

0/1 задача о ранце – динамическое программирование

- Вначале у нас есть рюкзак грузоподъёмностью L и набор из M предметов...
- ... предположим, мы положили туда предмет $\#M$, теперь у нас есть рюкзак грузоподъёмностью $L - W_M$ и $M - 1$ предметов
- Определим таблицу $C(L, M)$

0/1 задача о ранце – динамическое программирование

- Вначале у нас есть рюкзак грузоподъемностью L и набор из M предметов...
- ... предположим, мы положили туда предмет $\#M$, теперь у нас есть рюкзак грузоподъемностью $L - W_M$ и $M - 1$ предметов
- Определим таблицу $C(L, M)$
 - $C(L, 0) = C(0, M) = 0$
 - $C(L, M) = C(L, M-1)$ if $W_M > L$
 - $C(L, M) = \text{Max}(C(L, M-1), C_M + C(L-W_M, M-1))$ if $W_M \leq L$

0/1 задача о ранце – динамическое программирование

- Вначале у нас есть рюкзак грузоподъёмностью L и набор из M предметов...
- ... предположим, мы положили туда предмет $\#M$, теперь у нас есть рюкзак грузоподъёмностью $L - W_M$ и $M - 1$ предметов
- Определим таблицу $C(L, M)$
 - $C(L, 0) = C(0, M) = 0$
 - $C(L, M) = C(L, M-1)$ if $W_M > L$
 - $C(L, M) = \text{Max}(C(L, M-1), C_M + C(L-W_M, M-1))$ if $W_M \leq L$
- Трудоёмкость = ???

0/1 задача о ранце – динамическое программирование

- Вначале у нас есть рюкзак грузоподъёмностью L и набор из M предметов...
- ... предположим, мы положили туда предмет $\#M$, теперь у нас есть рюкзак грузоподъёмностью $L - W_M$ и $M - 1$ предметов
- Определим таблицу $C(L, M)$
 - $C(L, 0) = C(0, M) = 0$
 - $C(L, M) = C(L, M-1)$ if $W_M > L$
 - $C(L, M) = \text{Max}(C(L, M-1), C_M + C(L-W_M, M-1))$ if $W_M \leq L$
- Трудоёмкость = $O(L * M)$

0/1 задача о ранце – динамическое программирование

- Вначале у нас есть рюкзак грузоподъёмностью L и набор из M предметов...
- ... предположим, мы положили туда предмет $\#M$, теперь у нас есть рюкзак грузоподъёмностью $L - W_M$ и $M - 1$ предметов
- Определим таблицу $C(L, M)$
 - $C(L, 0) = C(0, M) = 0$
 - $C(L, M) = C(L, M-1)$ if $W_M > L$
 - $C(L, M) = \text{Max}(C(L, M-1), C_M + C(L-W_M, M-1))$ if $W_M \leq L$
- Трудоёмкость = $O(L * M)$ – *псевдо-полиномиальная*
- NB: L и все W_i должны быть целыми числами!

Псевдо-полиномиальная: почему?

- ▶ Дело в том, что во многих вариантах задачи L (грузоподъёмность) растёт как 2^M (число предметов)

Псевдо-полиномиальная: почему?

- ▶ Дело в том, что во многих вариантах задачи L (грузоподъёмность) растёт как 2^M (число предметов)
- ▶ А в таком варианте задача о ранце – NP-полная

Псевдо-полиномиальная: почему?

- Дело в том, что во многих вариантах задачи L (грузоподъёмность) растёт как 2^M (число предметов)
- А в таком варианте задача о ранце – NP-полная
- Можно ещё решить (неточно!) жадным алгоритмом (как?)

Задача о ранце – жадный алгоритм

- ▶ Упорядочим все предметы по убыванию C_i / W_i – дорогие и лёгкие предметы идут первыми, дешёвые и тяжёлые – последними

Задача о ранце – жадный алгоритм

- ▶ Упорядочим все предметы по убыванию C_i / W_i – дорогие и лёгкие предметы идут первыми, дешёвые и тяжёлые – последними
- ▶ На каждом шаге пытаемся положить в рюкзак первый предмет из списка, если он туда помещается
- ▶ Если же он не помещается, то он удаляется из списка

Задача о ранце – жадный алгоритм

- Упорядочим все предметы по убыванию C_i / W_i – дорогие и лёгкие предметы идут первыми, дешёвые и тяжёлые – последними
- На каждом шаге пытаемся положить в рюкзак первый предмет из списка, если он туда помещается
- Если же он не помещается, то он удаляется из списка
- Трудоёмкость = ???

Задача о ранце – жадный алгоритм

- Упорядочим все предметы по убыванию C_i / W_i – дорогие и лёгкие предметы идут первыми, дешёвые и тяжёлые – последними
- На каждом шаге пытаемся положить в рюкзак первый предмет из списка, если он туда помещается
- Если же он не помещается, то он удаляется из списка
- Трудоёмкость = $O(M)$

Итоги

- Рассмотрели
 - Динамическое программирование
 - Числа Фибоначчи
 - Задача о разрезании стержня
 - Задача о ранце
- Далее
 - Эвристические алгоритмы поиска

This page intentionally left blank...

Шифрование с помощью задачи о ранце

- ▶ Первый появившийся алгоритм шифрования с открытым ключом
 - ▶ Ранцевая криптосистема Меркла-Хеллмана

Шифрование с открытым / закрытым КЛЮЧОМ

- ▶ Открытый ключ: известен кому угодно, т.е. его не требуется скрывать
 - ▶ При этом предполагается, что у получателя сообщения есть ещё закрытый ключ для расшифровки

Шифрование с открытым / закрытым КЛЮЧОМ

- ▶ Открытый ключ: известен кому угодно, т.е. его не требуется скрывать
 - ▶ При этом предполагается, что у получателя сообщения есть ещё закрытый ключ для расшифровки
- ▶ Закрытый ключ: известен только двоим участникам передачи (в этом случае возникает отдельный сложный вопрос о передаче закрытого ключа)

Шифрование с помощью задачи о ранце

- ▶ Пусть два человека решают скрытно общаться
 - ▶ NB: стоимость предметов в этом варианте считается одинаковой
 - ▶ Получатель сообщения выбирает набор весов предметов W_i так, чтобы вес K -го предмета был больше суммы весов предыдущих $K-1$ предметов (супер-возрастающая последовательность)
 - ▶ Набор W_i получатель никому не показывает (в том числе отправителю) – **закрытый ключ**
 - ▶ Затем он формирует *другие веса* $V_i = (R * W_i) \% Q$, где Q – целое число, большее суммы весов всех предметов, а R – другое целое число, взаимно простое с Q
 - ▶ Набор V_i получатель публикует – открытый ключ

Шифрование с помощью задачи о ранце

► Шифрование

- Берём последовательность бит X_i , которую надо отправить, разбиваем её на блоки по числу предметов
- Отправляем каждый блок: $Y(\text{block}) = \text{SUM}(B_i * X_i)$

► Дешифрование

- Вычисляем $S = (Y * R^{-1}) \% Q$
 - Т.е. такое S , что $T = (S * R) \% Q$
- Решаем задачу о рюкзаке для S и супер-возрастающей последовательности W_i