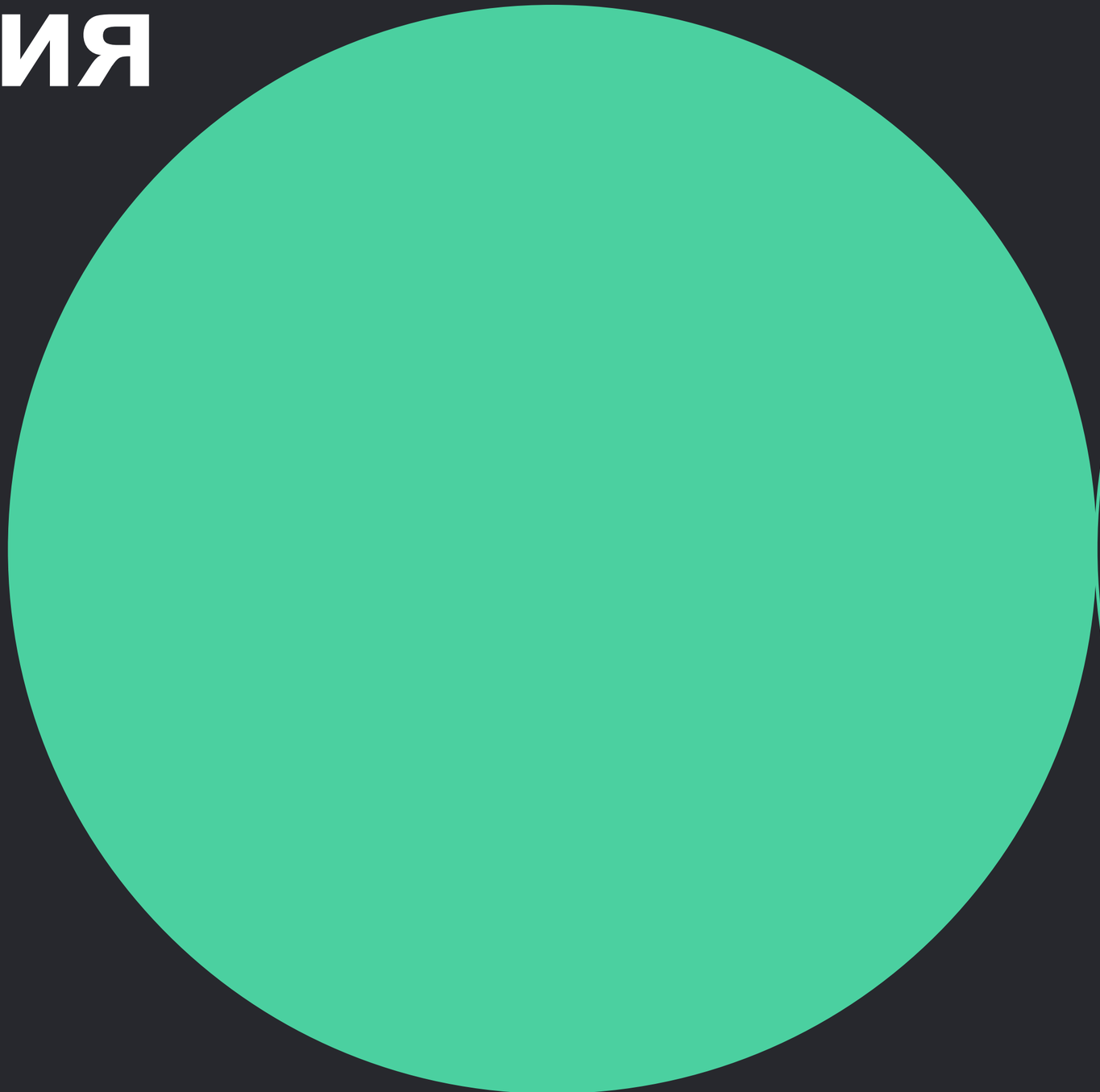
Динамическое программирование и “жадность”

Филипп Воронов

Teamlead, Поиск Mail.ru

Рекурсия



Что такое рекурсия?

Рекурсия — вызов внутри одной функции себя самой,
чаще всего с другими параметрами.

```
1 | mult(a, b):
2 |   if b == 1
3 |     return a
4 |   else
5 |     return a + mult(a, b - 1)
6 |
7 | print “Ответ: ” mult(n)
```

Рекурсивная функция подсчёта перемножения двух положительных целых чисел.

- Решение одного случая легко выразить через решение для другого случая
- Рекурсивные вызовы когда-то точно закончатся

Что такое рекурсия?

Рекурсия — вызов внутри одной функции себя самой,
чаще всего с другими параметрами.

```
1 | mult(a, b):
2 |   if b == 1
3 |     return a
4 |   else
5 |     return mult(a, b + 1) + a
6 |
7 | print “Ответ: ” mult(n)
```

Рекурсивная функция не сработает, т. к. рекурсия
не закончится

Разворачивание рекурсии

Превращение рекурсивного алгоритма
в нерекурсивный

```
1 | mult = 1
2 |
3 | while b > 1
4 |   mult = mult * a
5 |   b = b - 1
```

Рекурсии обычно медленнее работают, чем циклы, также глубина цепочки рекурсивных вызовов не может быть слишком длинной.

Для этого используют разворачивание рекурсии,
но не всегда это легко сделать и не во всех случаях нужно

Задача о монетках

Задача о монетках

Дано: число — денежная сумма, которую надо собрать из монет номиналом 3 или 5

Надо: определить, можно ли собрать эту сумму

Пример:

7 — нельзя
17 — можно (3, 3, 3, 3, 5)

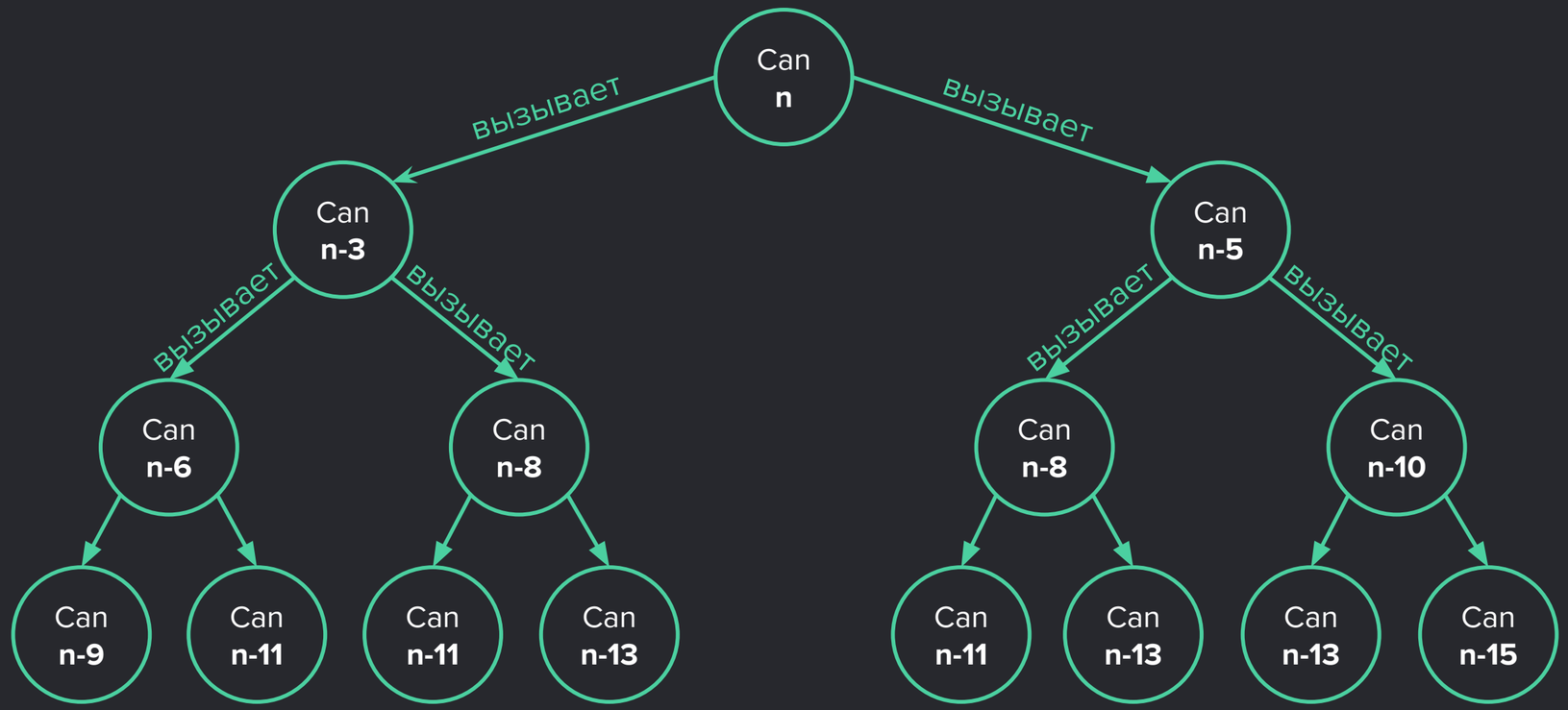
Рекурсивный метод решения

Эта задача легко решается с помощью рекурсивного алгоритма. Мы можем убедиться, что алгоритм корректный и рекурсия остановится

```
1  can(sum):
2    if sum = 3 или 5
3      return ДА
4    if sum >= 3 И can(sum - 3)
5      return ДА
6    if sum >= 5 и can(sum - 5)
7      return ДА
8    return НЕТ
```

Рекурсивный метод решения

Заметьте: с глубиной количество рекурсивных вызовов быстро растёт, но сами вызовы начинают повторяться. Глубина — от n/5 до n/3



Рекурсивный метод решения

Алгосложность

Глубина в лучшем случае n/5. У каждого рекурсивного вызова (кроме самых глубоких) есть по два своих рекурсивных вызова.

В итоге имеем $O(2^{n/5})$ времени работы, т. е. при увеличении суммы всего на 5 рублей время работы увеличится в 2 раза

Решение рекурсией с запоминанием

Рассмотрим решение задачи методом **рекурсии с запоминанием**: это применение динамического программирования к рекурсии

```
1   memory = [sum нулей]
2
3   can(sum):
4     if что-то лежит в memory[sum]:
5         return memory[sum]
6     if sum >= 3 И can(sum - 3)
7       memory[sum] = ДА
8       return ДА
9     if sum >= 5 и can(sum - 5)
10        memory[sum] = ДА
11      return ДА
12    memory[sum] = НЕТ
13    return НЕТ
```

Тут будем хранить все посчитанные промежуточные ответы

Решение рекурсией с запоминанием

Алгосложность

Вызовов теперь $O(n)$, стало быть, времени тоже $O(n)$,
что значительно лучше $O(2^{n/5})$. Памяти тоже $O(n)$
(массив + последовательность рекурсивных вызовов
в худшем случае)

Range minimum requests

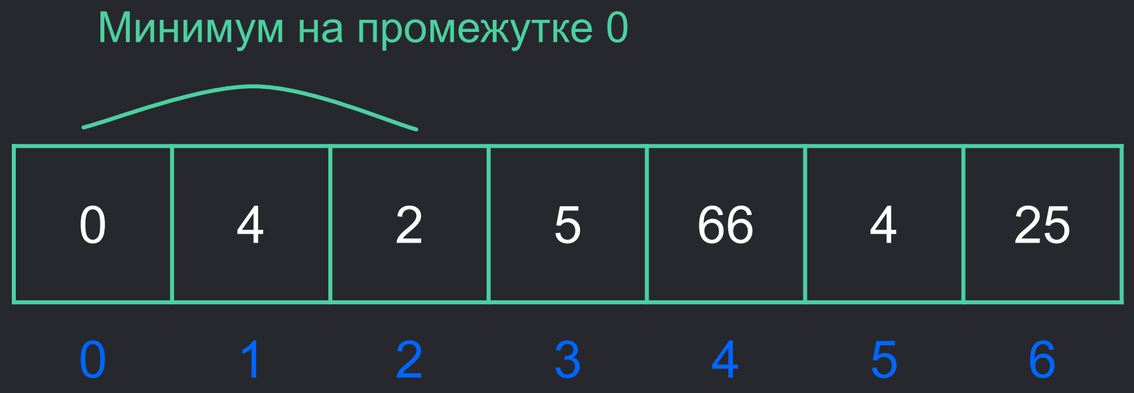
Range minimum requests

*запросы минимумов на отрезках

Задача: у нас есть массив. Нужно написать программу, которая будет принимать запросы в виде двух индексов и искать минимальное значение на промежутке в массиве от первого индекса включительно до второго невключительно

Range minimum requests

Массив:

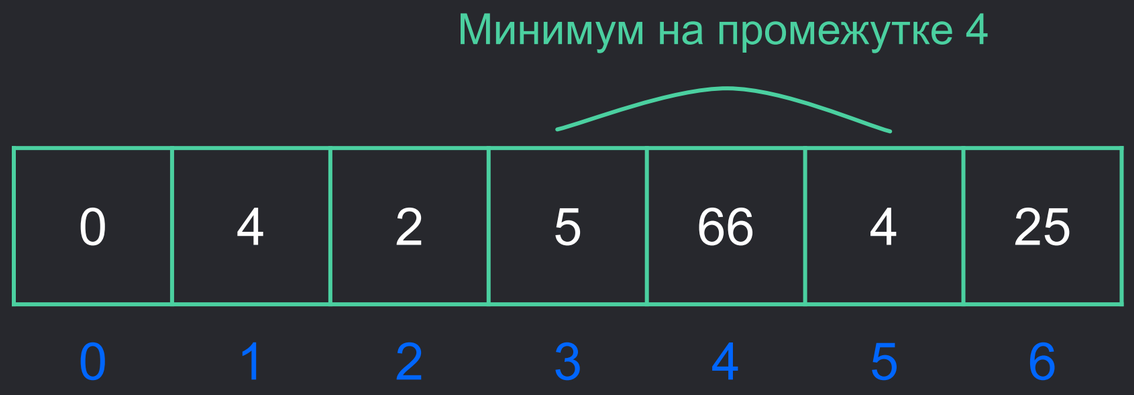


Запрос: 0 3 → **0**

[**0**, **4**, **2**, 5, 66, 3, 25], ответ 0

Range minimum requests

Массив:



Запрос: 3 6 → **4**

[0, 4, 2, **5, 66,** **4**, 25], ответ 4

Range minimum requests

```
1  arr = …
2
3  min(left, right):
4    ans = arr[left]
5    for i от left до right
6      if arr[i] < ans
7        ans = arr[i]
8    return arr[i]
```

Обычный поиск минимума,
но не на всём массиве,
а от left до right

Range minimum requests

```
1  arr = …
2
3  min(left, right):
4    ans = arr[left]
5    for i от left до right
6      if arr[i] < ans
7        ans = arr[i]
8    return arr[i]
```

Обычный поиск минимума,
но не на всём массиве,
а от left до right

Характеристики этого решения,
которое мы хотим улучшить:

- Время O(n)
- Доппамять O(1)

Range minimum requests

```
1   arr = …
2
3   memory = [n x n нулей]
4
5   for left от 0 до n
6     min = arr[left]
7     for right от left+1 до n
8       if arr[right] < min
9         min = arr[right]
10      memory[left][right] = min
11
12  min(left, right):
13    return memory[left][right]
```

Здесь будем сохранять ответы

Range minimum requests

```
1   arr = …
2
3   memory = [n x n нулей]
4
5   for left от 0 до n
6     min = arr[left]
7     for right от left+1 до n
8       if arr[right] < min
9         min = arr[right]
10      memory[left][right] = min
11
12  min(left, right):
13    return memory[left][right]
```

Здесь будем сохранять ответы

Заранее предпосчитаем все ответы

Range minimum requests

```
1   arr = …
2
3   memory = [n x n нулей]
4
5   for left от 0 до n
6     min = arr[left]
7     for right от left+1 до n
8       if arr[right] < min
9         min = arr[right]
10      memory[left][right] = min
11
12  min(left, right):
13    return memory[left][right]
```

Здесь будем сохранять ответы

Заранее предпосчитаем все ответы

Заметим, что у нас теперь две фазы работы алгоритма. Помимо фазы ответов на запросы, у нас есть подготовительная фаза, в которой мы заранее считаем все суммы. Такая фаза называется **препроцессингом (предподсчётом)**

Range minimum requests

```
1   arr = …
2
3   memory = [n x n нулей]
4
5   for left от 0 до n
6     min = arr[left]
7     for right от left+1 до n
8       if arr[right] < min
9         min = arr[right]
10      memory[left][right] = min
11
12  min(left, right):
13    return memory[left][right]
```

Здесь будем сохранять ответы

Заранее предпосчитаем все ответы

Теперь не надо ничего считать, всё предпосчитано

Заметим, что у нас теперь две фазы работы алгоритма. Помимо фазы ответов на запросы, у нас есть подготовительная фаза, в которой мы заранее считаем все суммы. Такая фаза называется **препроцессингом (предподсчётом)**

Range minimum requests

Алгосложность

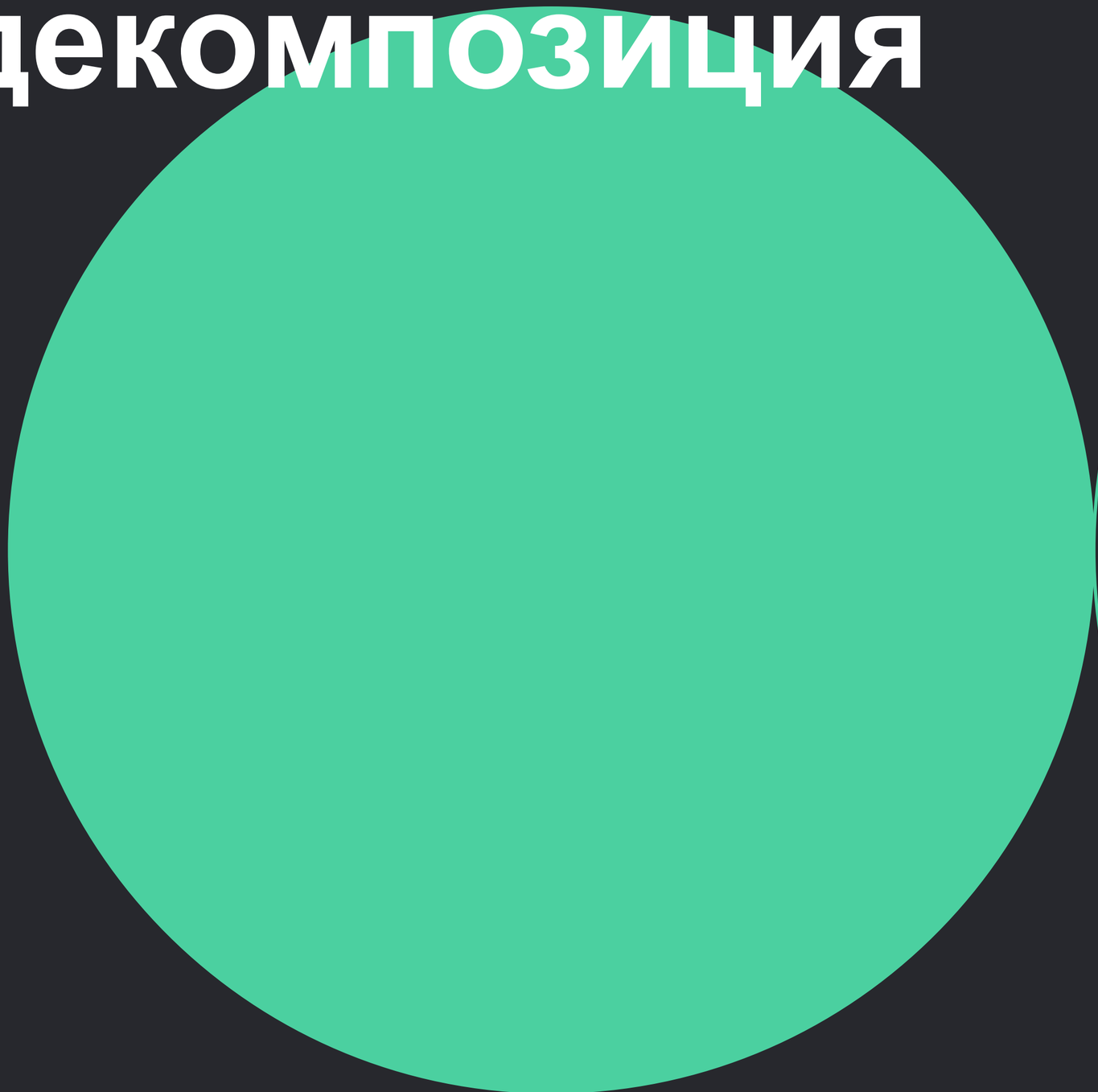
Препроцессинг тратит времени $O(n^2)$ и памяти $O(n^2)$
из-за двумерного массива, в котором запоминает результат

Range minimum requests

Хватит ли памяти?

Работа фазы запросов **ускорилась**. Для массива длиной в 1 млрд алгоритм займёт 930 тыс. терабайт памяти, а квадратичное время предподсчёта — больше трёх лет. Это очень много

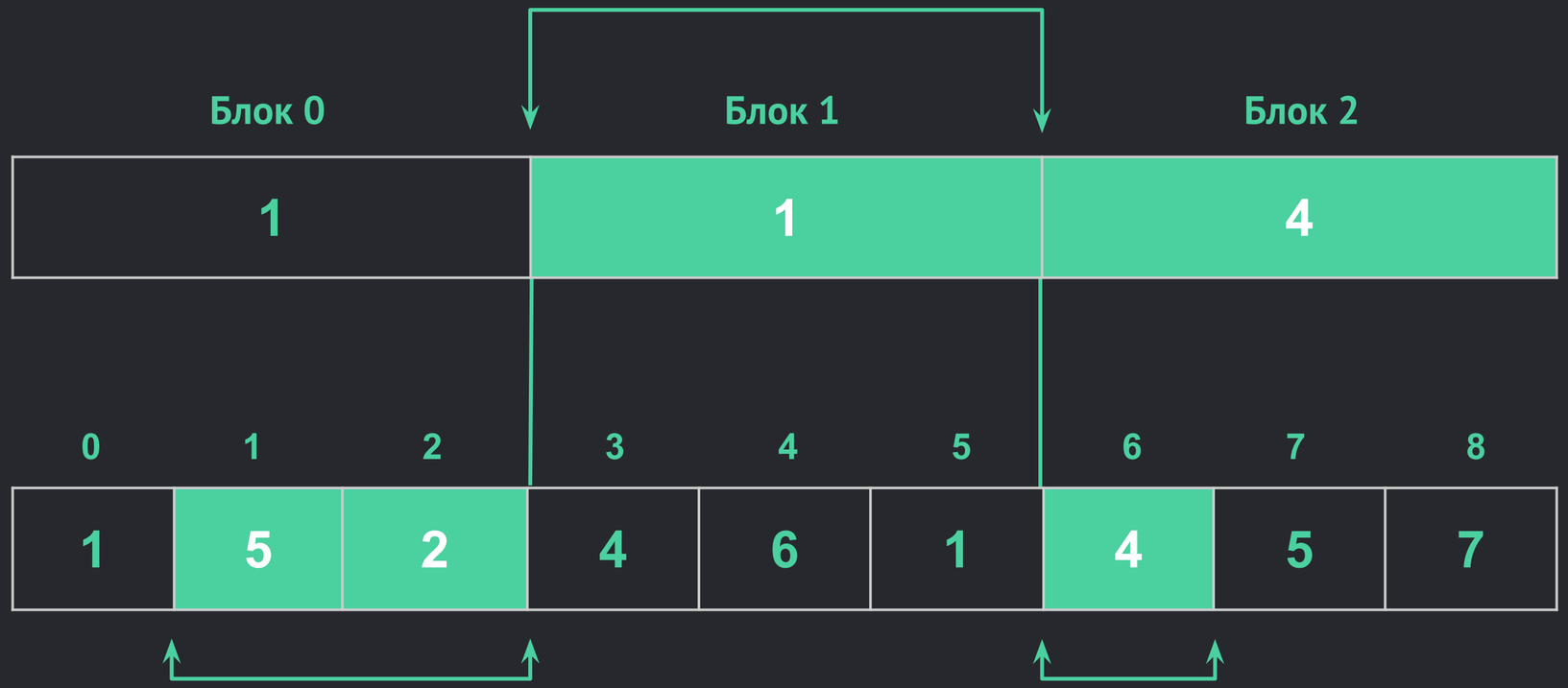
SQRT-декомпозиция



SQRT-декомпозиция

Чтобы **сократить расход памяти и времени**, разобьём весь массив на блоки длиной в корень из его длины. Количество таких блоков будет порядка корня этого числа. Это называется SQRT-декомпозицией (sqrt — англ. «квадратный корень»)

SQRT-декомпозиция



Запрос: left = 1, right = 7

SQRT-декомпозиция

Решение (простая динамика)

```
1   arr = ...
2   memory = [√n нулей ]
3
4
5   for left от 0 до n с шагом √n
6    min = arr[left]
7    for right от left+1 до left+ √n
8     if arr[right] < min
9       min = arr[right]
10   memory[номер шага] = min
```

Будем хранить ответы для подзадач

SQRT-декомпозиция

Решение (простая динамика)

```
1   arr = ...
2   memory = [√n нулей  ]
3
4
5   for left от 0 до n с шагом √n
6    min = arr[left]
7    for right от left+1 до left+ √n
8     if arr[right] < min
9       min = arr[right]
10   memory[номер шага] = min
```

Будем хранить ответы для подзадач

Предпосчитаем для каждого блока массива минимум на нём, сохраним в memory

SQRT-декомпозиция

Решение (простая динамика)

```
1   min(left, right):
2
3     left_up = left √n, округлить вверх
4     right_down = right / √n , округлить вниз
5     ans = arr[left]
6     for i от left до √n  * left_up
7       if arr[i] < ans: ans = arr[i]
8     for b от left_up до right_down
9       if memory[b] < ans
10        ans = memory[b]
11    for i от √n  * right_down до right
12      if arr[i] < ans: ans = arr[i]
13
14  return ans
```

Номер блока с левым концом интервала

SQRT-декомпозиция

Решение (простая динамика)

```
1   min(left, right):
2
3     left_up = left √n, округлить вверх
4     right_down = right / √n , округлить вниз
5     ans = arr[left]
6     for i от left до √n  * left_up
7       if arr[i] < ans: ans = arr[i]
8     for b от left_up до right_down
9       if memory[b] < ans
10        ans = memory[b]
11    for i от √n  * right_down до right
12      if arr[i] < ans: ans = arr[i]
13
14  return ans
```

Номер блока с левым концом интервала

Номер блока с правым концом интервала

SQRT-декомпозиция

Решение (простая динамика)

```
1   min(left, right):
2
3    left_up = left √n, округлить вверх
4    right_down = right / √n , округлить вниз
5    ans = arr[left]
6    for i от left до √n  * left_up
7     if arr[i] < ans: ans = arr[i]
8    for b от left_up до right_down
9     if memory[b] < ans
10      ans = memory[b]
11   for i от √n  * right_down до right
12    if arr[i] < ans: ans = arr[i]
13
14  return ans
```

Номер блока с левым концом интервала

Номер блока с правым концом интервала

Промежуточный ответ

SQRT-декомпозиция

Решение (простая динамика)

```
1   min(left, right):
2
3    left_up = left √n, округлить вверх
4    right_down = right / √n , округлить вниз
5    ans = arr[left]
6    for i от left до √n  * left_up
7     if arr[i] < ans: ans = arr[i]
8    for b от left_up до right_down
9     if memory[b] < ans
10      ans = memory[b]
11   for i от √n  * right_down до right
12    if arr[i] < ans: ans = arr[i]
13
14  return ans
```

Номер блока с левым концом интервала

Номер блока с правым концом интервала

Промежуточный ответ

Считаем минимум на пересечении
с самым левым блоком

SQRT-декомпозиция

Решение (простая динамика)

```
1   min(left, right):
2
3    left_up = left √n, округлить вверх
4    right_down = right / √n , округлить вниз
5    ans = arr[left]
6    for i от left до √n  * left_up
7     if arr[i] < ans: ans = arr[i]
8    for b от left_up до right_down
9     if memory[b] < ans
10      ans = memory[b]
11   for i от √n  * right_down до right
12    if arr[i] < ans: ans = arr[i]
13
14  return ans
```

Номер блока с левым концом интервала

Номер блока с правым концом интервала

Промежуточный ответ

Считаем минимум на пересечении
с самым левым блоком

Блоки не с краю покрыты полностью,
берём предподсчёт

SQRT-декомпозиция

Решение (простая динамика)

```
1   min(left, right):
2
3    left_up = left √n, округлить вверх
4    right_down = right / √n , округлить вниз
5    ans = arr[left]
6    for i от left до √n  * left_up
7     if arr[i] < ans: ans = arr[i]
8    for b от left_up до right_down
9     if memory[b] < ans
10      ans = memory[b]
11   for i от √n  * right_down до right
12    if arr[i] < ans: ans = arr[i]
13
14  return ans
```

Номер блока с левым концом интервала

Номер блока с правым концом интервала

Промежуточный ответ

Считаем минимум на пересечении с самым левым блоком

Блоки не с краю покрыты полностью, берём предподсчёт

Считаем минимум на пересечении с самым правым блоком

SQRT-декомпозиция

Алгосложность

Подсчёт минимумов займёт O($\sqrt{n}$ x $\sqrt{n}$) = O(n) времени
и O(n) памяти.

Для всех частей массива, входящих в интервал целиком
(max $\sqrt{n}$ штук), возьмём готовый препроцессинг;
на крайних неполных частях посчитаем сами (длина таких
не больше $\sqrt{n}$).
Таким образом, времени O($\sqrt{n}$).

Итого 30 килобайт дополнительной памяти и ускорение
в ~10 тыс. раз

Жадные алгоритмы

Алгоритм поиска максимума в массиве

Жадные алгоритмы

Пусть нам известно решение задачи на части данных. Если для того, чтобы узнать решение на расширенных данных, нам достаточно знать лишь готовое решение на старых данных (но не сами старые данные) и новую часть данных, то такое решение называется жадным

Жадные алгоритмы

Пример, когда работает. Вспомним задачу,
где мы искали максимум на массиве. Для определения
максимума при рассмотрении нового элемента нам
достаточно было только этого элемента (=новые данные)
и максимума среди предыдущих элементов (=решение
на старых данных, сами данные уже не нужны).

Задача про монетки — тоже пример жадного
алгоритма. Мы ещё не раз встретимся с ними

Жадные алгоритмы

Пример, когда не работает. При поиске самого частотного элемента в массиве (который встретился чаще всего) такой трюк уже не пройдёт: нам недостаточно знать самый частотный среди просмотренных, надо помнить обо всех элементах

9 1 5 6 7 8 7 ? ? ?

Просмотрено