Редицата на Фибоначи

Числата на Фибоначи в [математиката](https://bg.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0) образуват [редица](https://bg.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B4%D0%B8%D1%86%D0%B0), която се дефинира рекурсивно по следния начин:

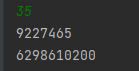
* F(0) = 0
* F(1) = 1
* F(n) = F(n-1) + F(n-2)

С увеличаването на броя на членовете на редицата всеки следваш член започва да нараства изключително бързо.

# Рекурсивна имплементация в Python

def fibonacci(n):  
 if n == 0:  
 return 0  
 elif n == 1:  
 return 1  
 else:  
 return fibonacci(n - 1) + fibonacci(n - 2)

Рекурсивното решаване на редицата е изключително трудоемко и отнема значителни периоди от време с нарастването на броя на редицата. При пресмятането на всеки следващ елемент се изчисляват отново стойностите за предходните 2 елемента, което води до загуба на изчислителен ресурс и увеличава значително времето за изчисление.  
Например за 35-я член на редицата времето за изпълнение е около 6 секунди.



# Итеративна имплементация в Python

def fibonacci\_iterative(n):  
 arr = []  
 arr.insert(0, 0)  
 arr.insert(1, 1)  
 for x in range(2, n+1):  
 arr.insert(x, arr[x - 2] + arr[x - 1])  
  
 return arr

Спрямо рекурсивното решение итеративният вариант ни дава много по-добре време на изпълнение на задачата.  
За 35-я член на редицата на Фибоначи чрез итеративно решение понижихме времето за изпълнение с 6 секунди, времето нужно за обработка е под секунда. Проблемът на това решение е, че заема големи количества памет. За 35-я член на редицата е нужен масив с 35 елемента. Този проблем ще доведе до значителна загуба на памет при големи членове на редицата.  
  
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418, 317811, 514229, 832040, 1346269, 2178309, 3524578, 5702887, 9227465]   
  
  
Оптимизация на итеративното решение

def fibonacci\_optimization(n):  
 arr = []  
 arr.insert(0, 0)  
 arr.insert(1, 1)  
 for x in range(n-1):  
 temp = arr[0] + arr[1]  
 arr[0] = arr[1]  
 arr[1] = temp  
  
 return arr

Проблемът с паметта може да бъде решен ако пазим само последните 2 члена на редицата, по този начин гарантираме, че винаги ще можем да изчислим следващият елемент, без да е нужно да заделяме големи количества памет.  
  
  
  
При решение на редицата за 1000 елемента при итеративното решение масивът от 1000 елемента е изчислен 6 пъти по бавно от оптимизирания вариант с 2 елемента. При оптимизирания вариант на алгоритъма за 1000 елемента спестяваме значително количество памет и има подобрение във времето за изпълнение.

