# Algorytmy i Struktury Danych

## Wykład 11

### Metody konstrukcji algorytmów

- dziel i zwyciężaj (QuickSort / MergeSort)
- zachłanne (alg. Kruskala)
- programowanie dynamiczne

  metoda zamiany rychłodniwego algorytmu rekurencyjnego na rodzimowy iteracyjny

### Przykład elementarny

$F_0 = 1$

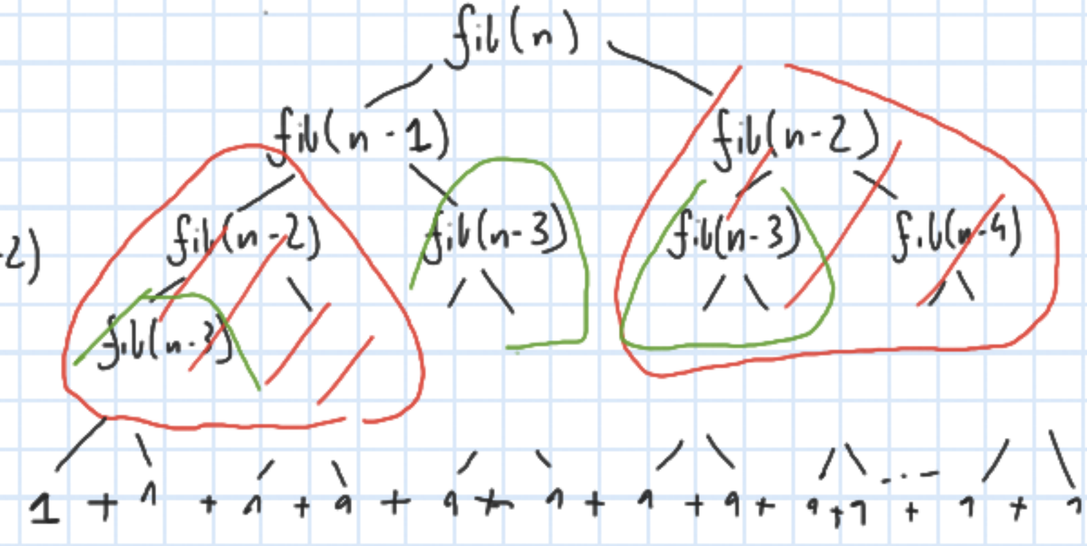
$F_1 = 1$

$F_n = F_{n-1} + F_{n-2}, n \geq 2$

1, 1, 2, 3, 5, 8, 13, 21, ...

$F_n = \Theta\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right)$

```
def fib(n):
    if n < 2: return 1
    return fib(n-1) + fib(n-2)
```




### Obliczanie n-tej liczby Fibonacciego algorytmem dynamicznym

- tworzymy tablicę, w której zapamiętujemy wyniki wywołań i z niej korzystamy zamiast rekurencji.

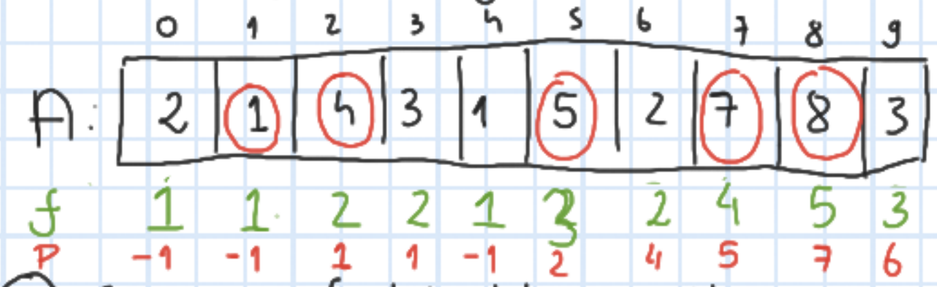
```
def fib(n):
    F = [1] * (n+1)
    for i in range(2, n+1):
        F[i] = F[i-1] + F[i-2]
    return F[n]
```

$O(n)$

# Najdłuższy rosnący podciąg

Dane: $A[0], \ldots, A[n-1]$ - tablica liczb

Zadanie: Chcemy znaleźć najdłuższy (niekoniecznie spójny) podciąg $A$




(1) Stworzenie funkcji, którą obliczamy

~~$f(i)$ = długość najdłuższego podciągu ciągu $A[0], \ldots, A[i]$~~

$f(i)$ = długość najdłuższego podciągu kończącego się na $A[i]$

```
def ps(A,P,i):
    if P[i] ≠ -1:
        ps(A,P,P[i])
    print(A[i])
```

(2) Zapisanie funkcji w postaci rekurencyjnej

$f(0) = 1$

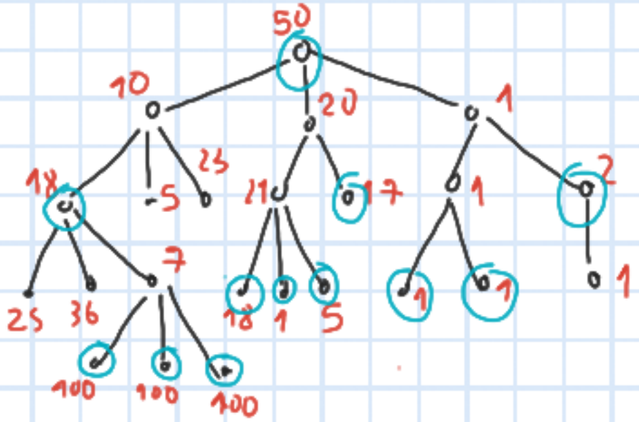
$f(i) = \max \{ f(t) + 1 \mid t < i \wedge A[t] < A[i] \}$ ($f(-x) = 0$, $A[-1] = -\infty$)

$\max_i f(i)$ — długość najdłuższego ciągu rosnącego

(3) Implementacja

```
def lis(A):
    n = len(A)
    F = [1] * n
    P = [-1] * n
    for i in range(1,n):
        for j in range(i):
            if A[j] < A[i] and F[j]+1 > F[i]:
                F[i] = F[j] + 1
                P[i] = j
    return max(F), P
```

# Problem imprezy firmowej




Chcemy wybrać niezależnych o maksymalnej sumie, takie że żadne dwa nie są połączone krawędzią

```
class Employee
    def __init__(self, fun):
        self.emp = []
        self.fun = fun
        self.f = -1
        self.g = -1
```

① Określenie funkcji, które będziemy obliczać

$f(v)$ = wartość najlepszej imprezy w poddrzewie zakorzenionym w $v$ (wyżej drzewa)

$g(v)$ = wartość najlepszej imprezy w poddrzewie zakorzenionym w $v$, o ile $v$ nie idzie na tę imprezę

② Zapis rekurencyjny

$$f(v) = \max\left( v.fun + \sum_{u_i - \text{dziecko } v} g(u_i) \;,\; g(v) \right)$$

$$g(v) = \sum_{v_i - \text{dziecko } v} f(u_i)$$

③ Implementacja

```
def f(v):
    if v.f >= 0: return v.f
    x = v.fun
    for ui in v.emp:
        x += g(ui)
    y = g(v)
    v.f = max(x,y)
    return v.f

def g(v):
    if v.g >= 0: return v.g
    x = 0
    for vi in v.emp:
        x += f(ui)
    v.g = x
    return v.g
```