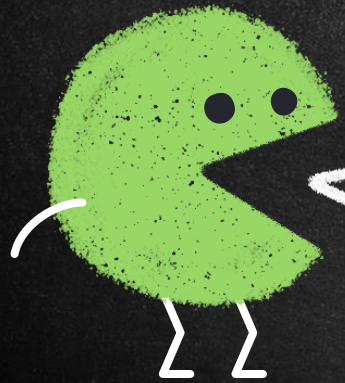
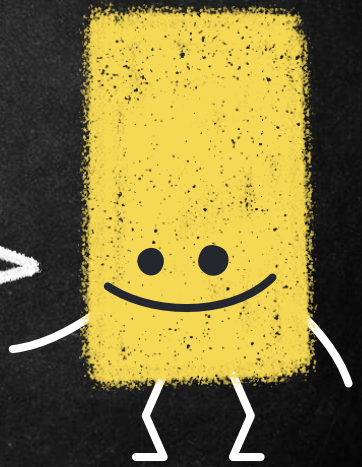
Vežbe 6

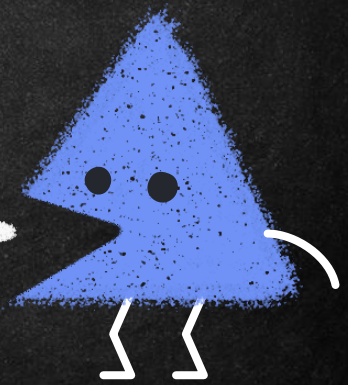
Dinamičko programiranje





Šta je dinamičko programiranje?

- Vrsta optimizacione metode
- Tehnika u programiranju kojom se može drastično smanjiti složenost nekog algoritma
- Svodi rešavanje velikog problema na rešavanje malih problema, uz mogućnost ponovnog korišćenja rešenja određenog potproblema
  - Slično paradigmi „zavadi pa vladaj“
  - Primer koji smo već videli: Merge sort



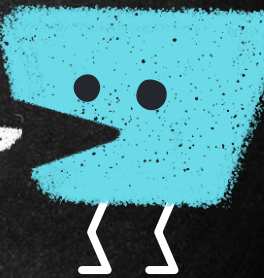
Tehnike dinamičkog programiranja

Tabulacija

- “Bottom-up” pristup
- Rezultati potproblema se čuvaju u tabeli, pa se potom „spajaju“ kako bi se rešio početni problem
- Implementira se korišćenjem iteracija

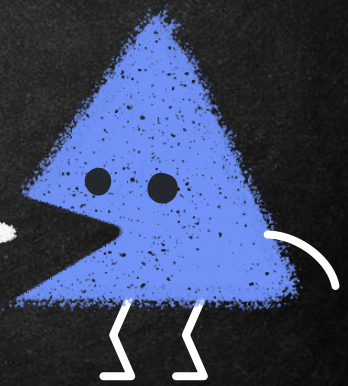
Memoizacija

- “Top-down” pristup
- Čuvaju se rezultati poziva funkcija i ponovo koriste po potrebi (ako se opet pozove ista funkcija sa istim ulazima)
- Implementira se korišćenjem rekurzije



Zadatak 1

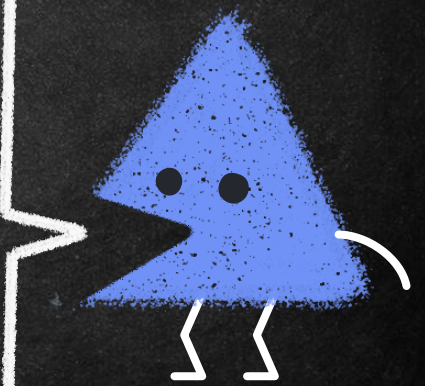
- Optimizovati algoritam koji vrši računanje *n*-tog Fibonačijevog broja:
  - korišćenjem tabulacije
  - korišćenjem memoizacije



Zadatak 1 – Rešenje

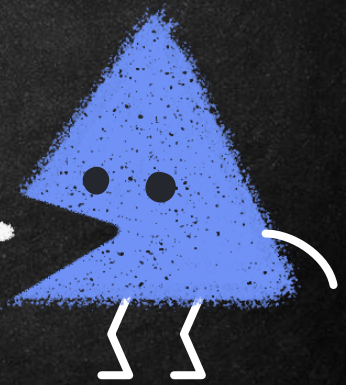
```
def fibonacci_tabulation(n):
    if n == 0:
        return 0
    if n == 1:
        return 1

    table = [None] * (n + 1)
    table[0] = 0
    table[1] = 1

    for i in range(2, n + 1):
        table[i] = table[i - 1] + table[i - 2]

    return table[n]
```

```
memo = {}

def fibonacci_memoization(n):
    if n == 0:
        return 0
    if n == 1:
        return 1
    elif n in memo:
        return memo[n]
    else:
        memo[n] = 
fibonacci_memoization(n - 1) +
fibonacci_memoization(n - 2)
        return memo[n]
```



Zadatak 2

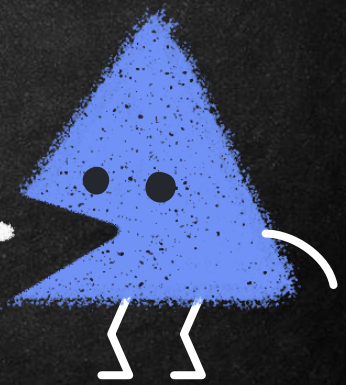
- Osoba se penje uz stepenice. U jednom skoku može da se preskoči 1, 2 ili 3 stepenika. Napisati algoritam koji određuje na koliko načina osoba može da pređe put koji se sastoji od *n* stepenika:
  - korišćenjem tabulacije
  - korišćenjem memoizacije



Zadatak 2 - Rešenje

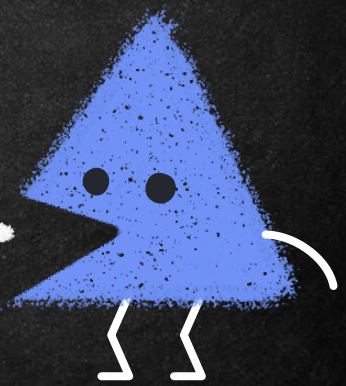
```python
def stepenice_tabulation(n):
    res = [0] * (n + 2)
    res[0] = 1
    res[1] = 1
    res[2] = 2

    for i in range(3, n + 1):
        res[i] = res[i - 1] \
                + res[i - 2] \
                + res[i - 3]

    return res[n]
```

```python
def findStepHelper(n, dp):
    if n == 0:
        return 1
    elif n < 0:
        return 0

    if dp[n] != -1:
        return dp[n]

    dp[n] = findStepHelper(n - 3, dp) \
          + findStepHelper(n - 2, dp) \
          + findStepHelper(n - 1, dp)
    return dp[n]


def stepenice_memoization(n):
    dp = [-1 for i in range(n + 1)]
    return findStepHelper(n, dp)
```



Zadatak 3

- Napisati algoritam koji određuje na koliko načina se može stići od gornjeg levog polja do donjeg desnog polja na nekoj tabli, dimenzija M_xN, pri čemu se može pomerati jedno polje u desnu stranu ili jedno polje na dole:
  - korišćenjem tabulacije
  - korišćenjem memoizacije



Zadatak 3 - Rešenje

```
def tabla_tabulation(m, n):
    count = [[0 for x in range(n)] for y in range(m)]

    for i in range(m):
        count[i][0] = 1

    for j in range(n):
        count[0][j] = 1

    for i in range(1, m):
        for j in range(1, n):
            count[i][j] = count[i - 1][j] + count[i][j - 1]

    return count[m - 1][n - 1]
```

```
def tabla_memoizacija(m, n):
    matrica = [[0 for i in range(m)] for j in range(n)]
    return tabla_helper(m, n, matrica)


def tabla_helper(m, n, matrica):
    if m == 1 or n == 1:
        matrica[m][n] = 1
        return 1

    if matrica[m][n] != 0:
        return matrica[m][n]

    matrica[m][n] = tabla_helper(m-1, n, matrica) \
                + tabla_helper(m, n-1, matrica)
    return matrica[m][n]
```