STRUKTURE PODATAKA • PRIMERI U PYTHON-U

Stek i red

Magacin (LIFO) i red (FIFO): ideja, operacije, sopstvene klase, ugrađeni alati i primeri

PYTHON

```
s = DStack()
s.push(1); s.push(2)
print(s.pop())
```

IZLAZ

```
2
```

STEK I RED

Plan časa

1

Stek (magacin)

poslednji unutra, prvi napolje

2

Red

prvi unutra, prvi napolje

3

Sopstvene klase

DStack i DQueue nad lančanom listom

4

Šta ima Python

list, collections.deque

5

Lavirint

primena steka: vraćanje unazad

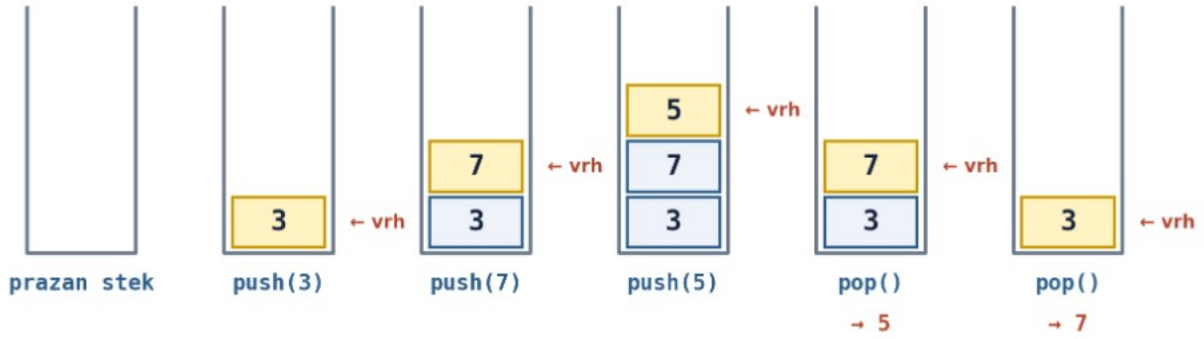
6

Izvršavanje poslova

primena reda: kvant vremena

STEK (MAGACIN)

Poslednji unutra, prvi napolje (LIFO)



Gomila tanjira

Novi tanjir ide na vrh, a uzima se uvek onaj sa vrha.

Ctrl + Z

Poništava se poslednja izmena, pa pretposlednja…

Dugme „Nazad“

Pregledač se vraća na poslednju posećenu stranu.

STEK (MAGACIN)

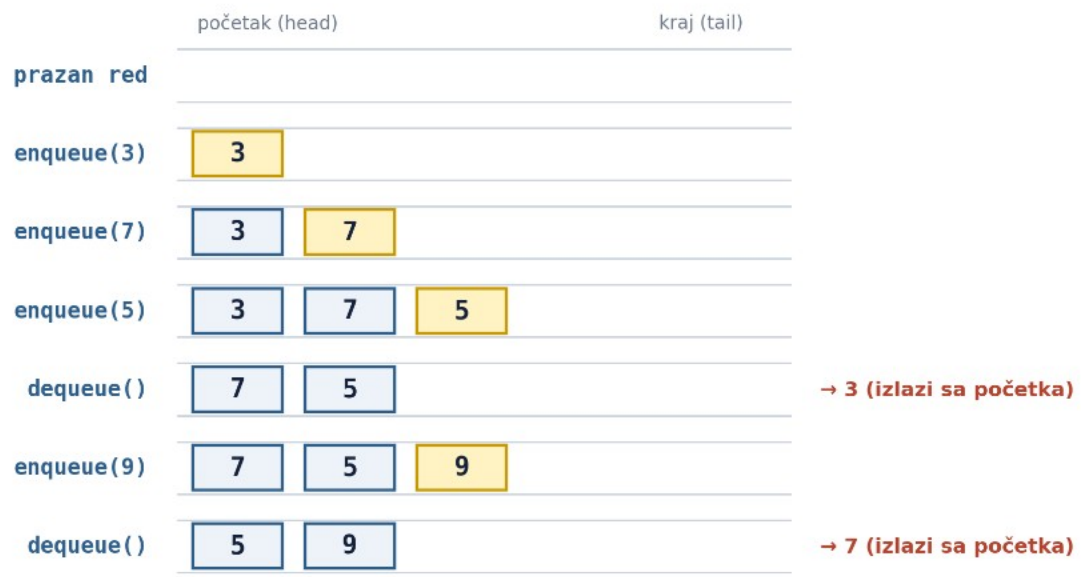
Operacije nad stekom

Operacija	Šta radi	Složenost
`push(el)`	stavlja element na vrh	O(1)
`pop()`	skida element sa vrha i vraća ga	O(1)
`get_top()`	vraća element sa vrha, ne skida ga	O(1)
`is_empty()`	proverava da li je stek prazan	O(1)

Pristup elementima ispod vrha nije dozvoljen. Do njih se stiže samo skidanjem onih iznad. Primene: pozivi funkcija, poništavanje izmena, provera zagrada, vraćanje unazad (lavirint).

RED

Prvi unutra, prvi napolje (FIFO)



Red na kasi

Ko je prvi došao, prvi je uslužen; novi staju na kraj.

Štampač

Dokumenti se štampaju redom kojim su poslati.

Procesor

Procesi čekaju u redu da dobiju vreme za izvršavanje.

RED

Operacije nad redom

Operacija	Šta radi	Složenost
`enqueue(el)`	dodaje element na kraj reda	O(1)
`dequeue()`	uzima element sa početka i vraća ga	O(1)
`front()`	vraća prvi element, ne uzima ga	O(1)
`is_empty()`	proverava da li je red prazan	O(1)

Kod steka se sve dešava na jednom kraju. Kod reda na dva: **dodaje se na kraj, uzima sa početka**. Zato red čuva dva pokazivača, head i tail.

Kako napraviti stek i red

Pomoću polja

Stek: dodaje se i uzima na kraju polja, O(1). Red: uzimanje sa početka pomera sve elemente (O(n)), osim kod kružnog bafera.

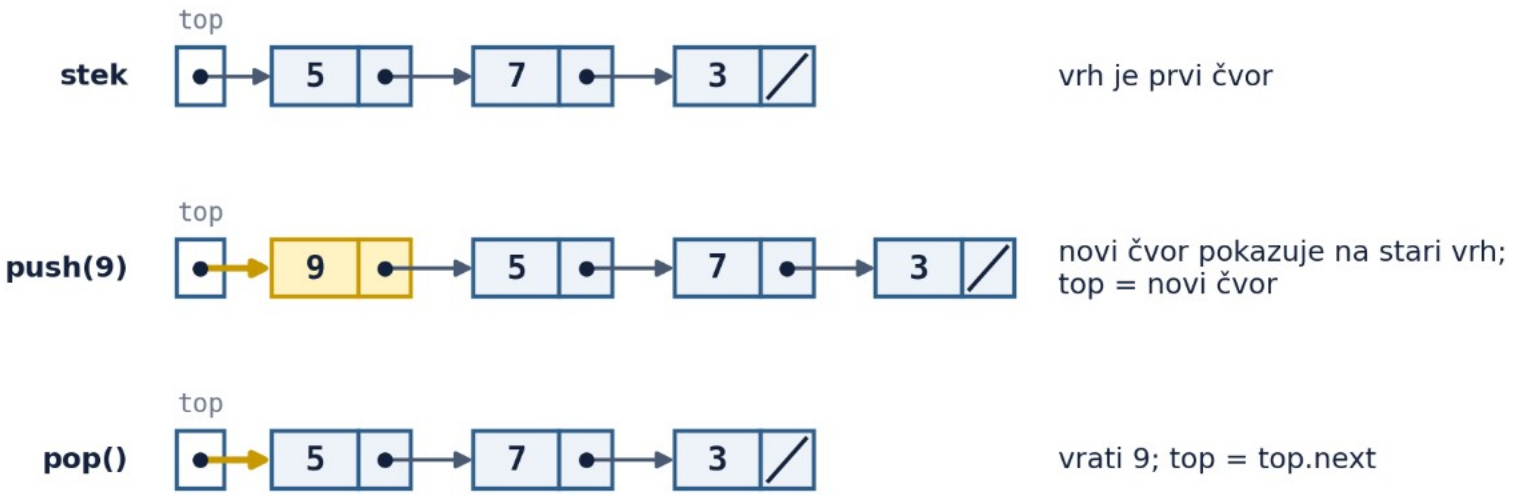
Pomoću lančane liste

Stek: čvorovi se dodaju i uzimaju na početku liste. Red: dodaje se na tail, uzima sa head. Sve operacije O(1), bez ograničenja veličine.

Klase **DStack** i **DQueue** iz originalnih vežbi su dinamičke (D), napravljene nad lančanom listom. Isto radimo i mi, koristeći ono što već znamo o čvorovima.

STEK NAD LANČANOM LISTOM

Vrh steka je prvi čvor liste



push je dodavanje na početak liste (add_to_head), a **pop** brisanje sa početka (delete_from_head). Pokazivač na kraj nije potreban.

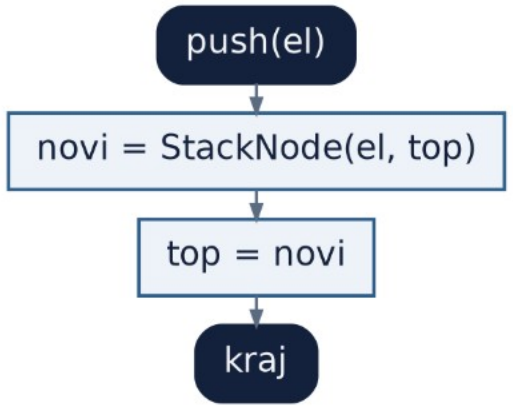
STEK NAD LANČANOM LISTOM

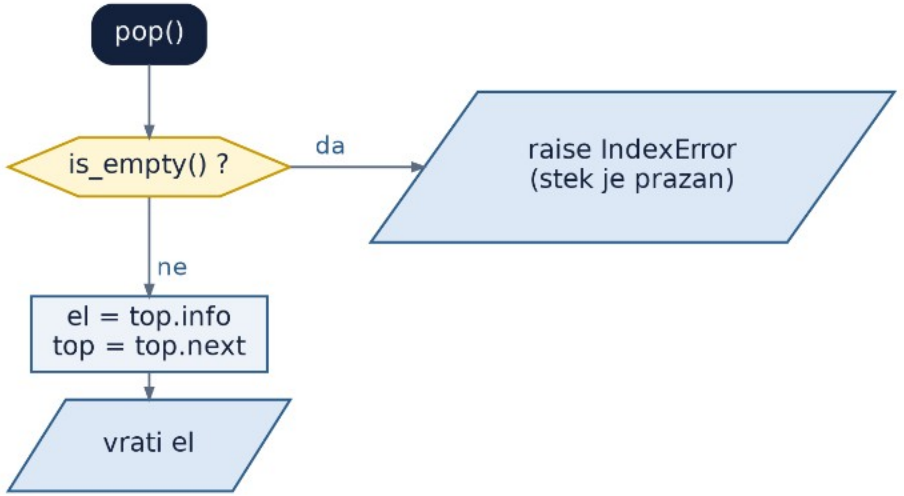
Klasa DStack

```
class StackNode:
    def __init__(self, info, next_node=None):
        self.info = info
        self.next = next_node     # čvor ispod u steku


class DStack:
    def __init__(self):
        self.top = None           # pokazivač na vrh steka

    def is_empty(self):
        return self.top is None

    def push(self, el):
        self.top = StackNode(el, self.top)
```

```
    def pop(self):
        if self.is_empty():
            raise IndexError("Stek je prazan!")
        el = self.top.info
        self.top = self.top.next
        return el

    def get_top(self):
        if self.is_empty():
            raise IndexError("Stek je prazan!")
        return self.top.info
```

Dijagrami toka: push i pop





push

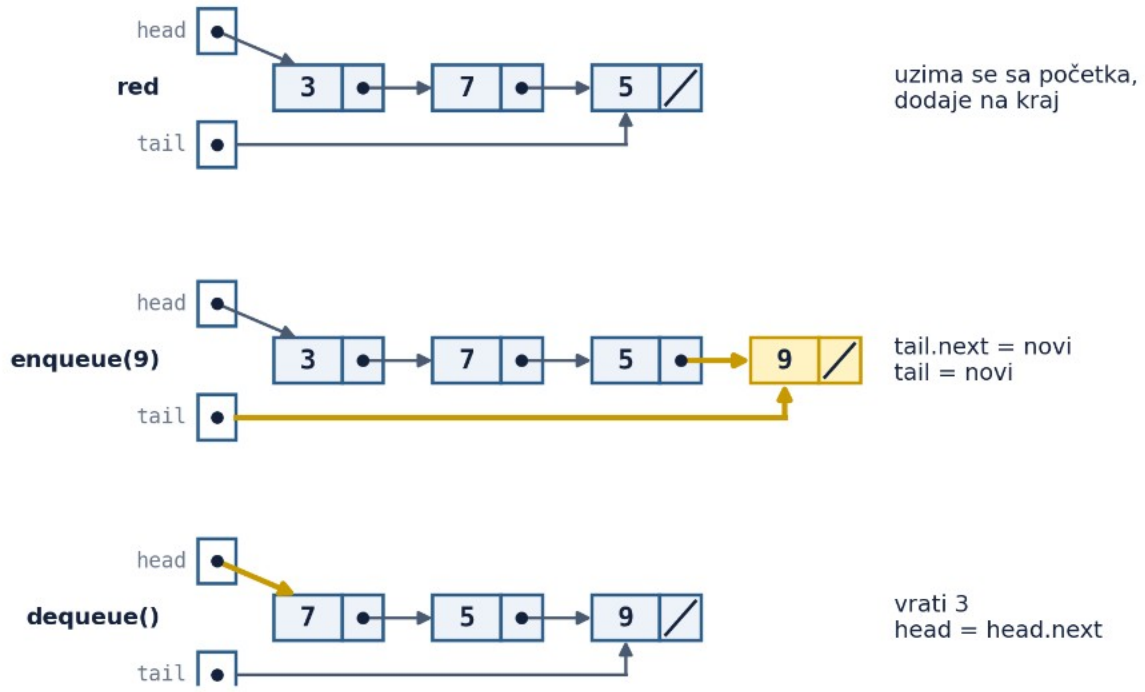
Uvek uspeva, bez provera.

pop

Nad praznim stekom podiže grešku.

RED NAD LANČANOM LISTOM

Dodaje se na tail, uzima sa head



enqueue kači novi čvor iza poslednjeg: tail.next = novi, tail = novi.

dequeue uzima prvi čvor: head = head.next.

Kada red postane prazan, i **tail** mora postati None.

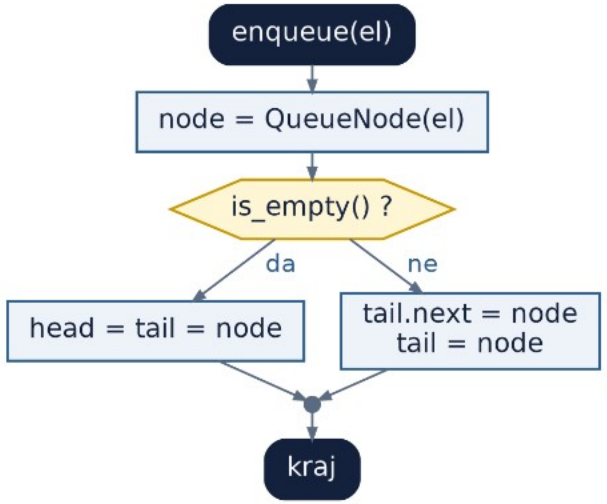
RED NAD LANČANOM LISTOM

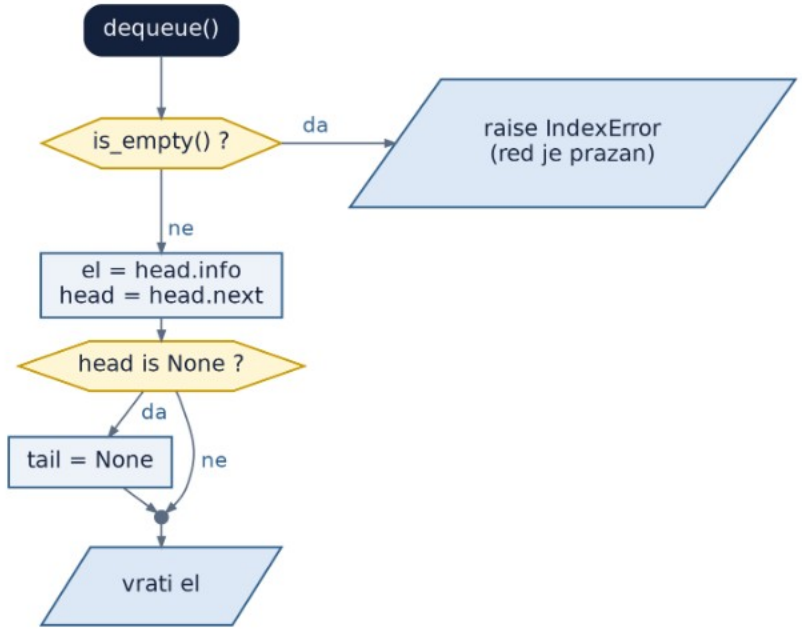
Klasa DQueue

```
class QueueNode:
    def __init__(self, info):
        self.info = info
        self.next = None          # čvor iza u redu


class DQueue:
    def __init__(self):
        self.head = None          # početak reda: odavde se uzima
        self.tail = None          # kraj reda: ovde se dodaje

    def is_empty(self):
        return self.head is None
```

```
    def enqueue(self, el):
        node = QueueNode(el)
        if self.is_empty():
            self.head = self.tail = node
        else:
            self.tail.next = node
            self.tail = node

    def dequeue(self):
        if self.is_empty():
            raise IndexError("Red je prazan!")
        el = self.head.info
        self.head = self.head.next
        if self.head is None:     # red je ostao
            self.tail = None
        return el
```

Dijagrami toka: enqueue i dequeue





Prazan red

Poseban slučaj u obe operacije: menjaju se i head i tail.

STEK I RED

Šta već ima u Python-u

Struktura	Python	Dodavanje	Uzimanje	Pogled na vrh / početak
stek	`list`	`s.append(x)`	`s.pop()`	`s[-1]`
red	`collections.deque`	`q.append(x)`	`q.popleft()`	`q[0]`
red za niti	`queue.Queue`	`q.put(x)`	`q.get()`	—

Ne list.pop(0) za red

Uzimanje sa početka liste pomera sve elemente: O(n). deque to radi u O(1).

Moduli

collections i queue su u standardnoj biblioteci: samo from collections import deque, bez instalacije.

Ugrađeni stek i red na delu

PYTHON

```
stek = []
stek.append(1)
stek.append(2)
stek.append(3)
print(stek[-1], stek.pop(), stek)

from collections import deque
red = deque()
red.append(1)
red.append(2)
red.append(3)
print(red[0], red.popleft(), list(red))
```

IZLAZ

```
3 3 [1, 2]
1 1 [2, 3]
```

Kada šta

Za učenje pišemo svoje klase da vidimo kako rade. U praksi se koriste list i deque.

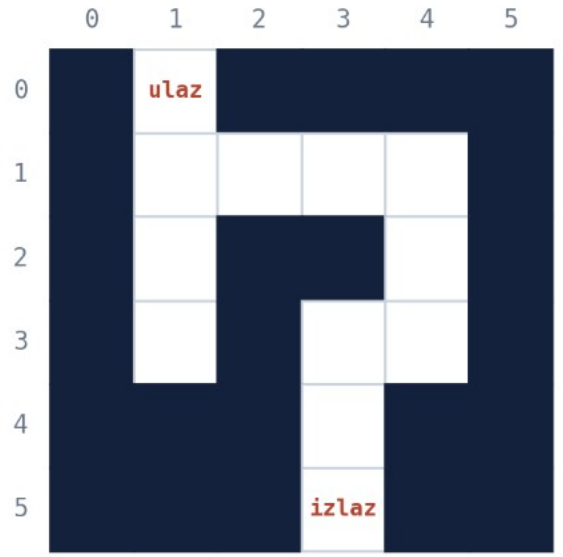
PRIMENA STEKA

Određivanje puta kroz lavirint

ZADATAK

Odrediti put kojim miš treba da prođe da bi izašao iz lavirinta. Lavirint je zadat kao matrica prepreka. Miš može da pređe na susedno polje levo, desno, gore ili dole. Ulaz je u prvoj vrsti, a izlaz je bilo koje polje u poslednjoj vrsti.

U Python-u lavirint zapisujemo kao listu stringova: **#** je zid, **.** je prolaz.



tamno = zid (#), belo = prolaz (.)

LAVIRINT

Ideja: stek pamti put kojim je miš prošao

Korak napred = push

Kada miš pređe na novo polje, to polje se stavlja na stek.

Slepa ulica = pop

Kada nema kuda dalje, skida se polje sa vrha: miš se vraća korak nazad.

Posećena polja

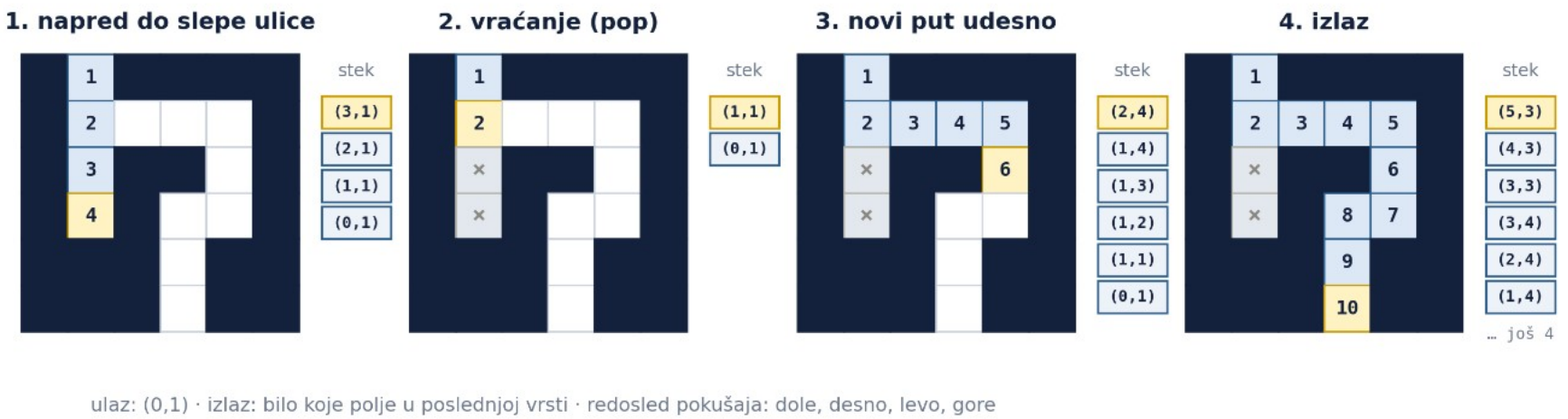
Pamte se da miš ne bi ulazio dva puta u isto polje i vrteo se u krug.

Kraj

Kada je vrh steka u poslednjoj vrsti, stek sadrži ceo put od ulaza do izlaza.

LAVIRINT

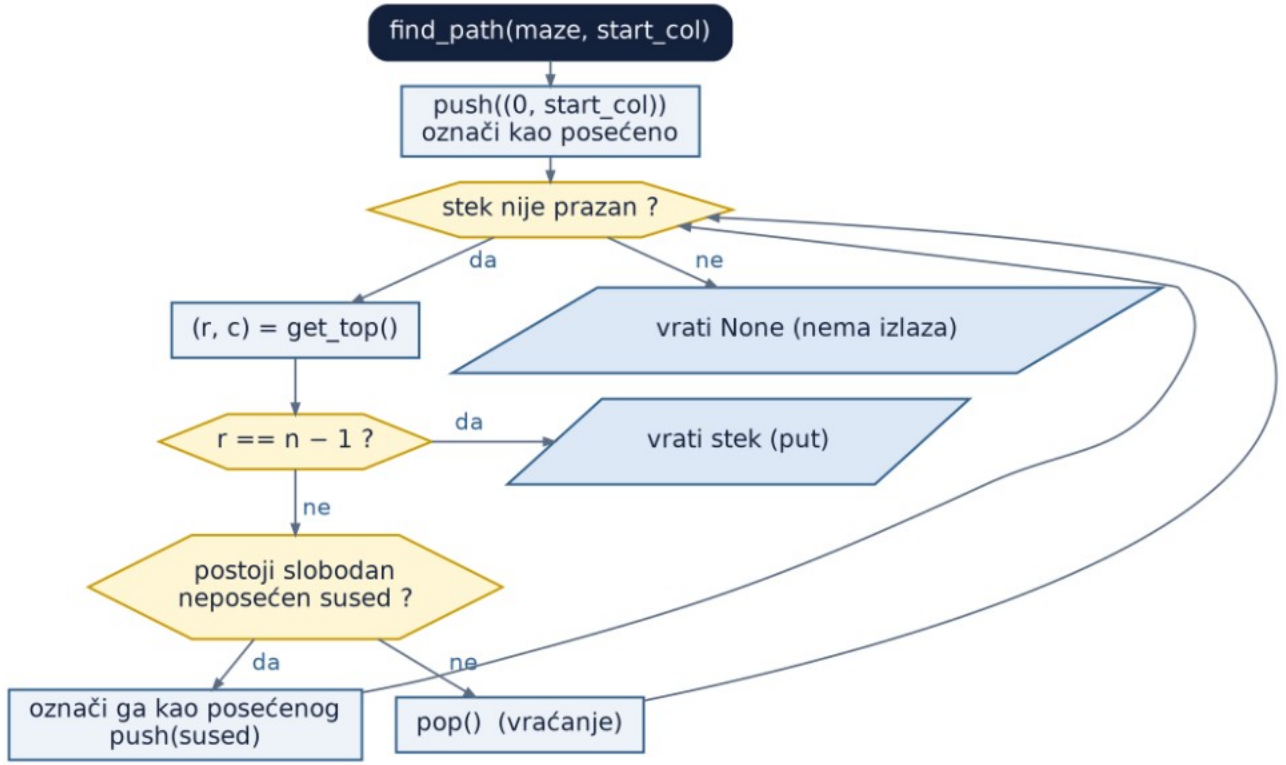
Skica rada: napred, nazad, pa novi put



Plavo je put na steku, žuto je vrh, × su posećena polja iz kojih se miš vratio. Redosled pokušaja: dole, desno, levo, gore.

LAVIRINT

Dijagram toka



LAVIRINT

Kôd: priprema

```
def is_free(maze, r, c):
    return 0 ⩽ r < len(maze) and 0 ⩽ c < len(maze[0]) and maze[r]
[c] == "."

def find_path(maze, start_col):
    n = len(maze)
    visited = [[False] * len(maze[0]) for _ in range(n)]
    moves = [(1, 0), (0, 1), (0, -1), (-1, 0)]    # dole, desno, levo, gore
    stack = DStack()
    stack.push((0, start_col))
    visited[0][start_col] = True
    while not stack.is_empty():
        r, c = stack.get_top()
```

is_free

Polje je u matrici i nije zid.

moves

Pomeraji: dole, desno, levo, gore.

LAVIRINT

Kôd: glavna petlja

```
        if r == n - 1:                          # stigli smo u poslednju vrstu
            return stack
        moved = False
        for dr, dc in moves:
            nr, nc = r + dr, c + dc
            if is_free(maze, nr, nc) and not visited[nr][nc]:
                visited[nr][nc] = True
                stack.push((nr, nc))            # korak napred
                moved = True
                break
        if not moved:
            stack.pop()                         # slepa ulica: vrati se
    return None
```

PRIMENA REDA

Izvršavanje poslova

ZADATAK

Svaki posao zahteva određeno vreme. Da bi izvršavanje teklo ravnomerno, svaki posao dobija kvant vremena, pa se zaustavlja i kreće sledeći. Poslovi dobijaju kvante redom kojim su se pojavili.

Uzmi prvi

dequeue: posao sa početka reda radi jedan kvant.

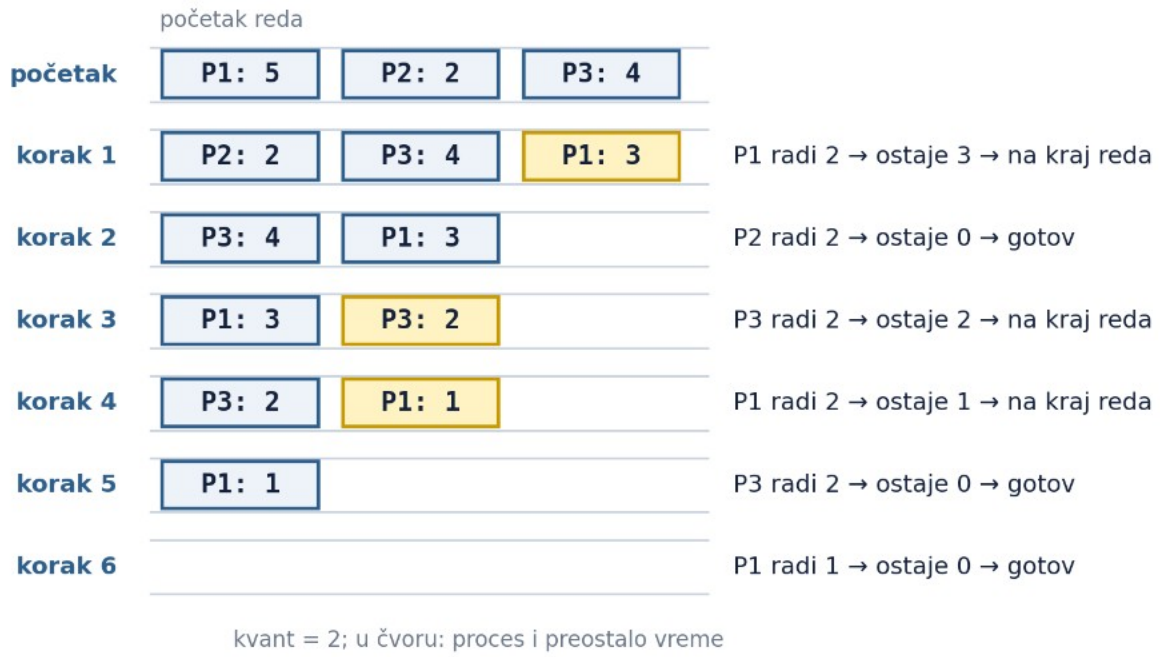
Nije gotov?

enqueue: vraća se na kraj reda sa umanjenim vremenom.

Gotov?

Ne vraća se u red.

Skica rada: kvant = 2



Na početku: P1 treba 5, P2 treba 2, P3 treba 4.

P1 radi 2, ostaje 3, pa ide na kraj reda.

P2 radi 2 i završava se.

Svi su gotovi posle 6 koraka.

IZVRŠAVANJE POSLOVA

Klasa Task

PYTHON

```
class Task:
    def __init__(self, id_task, time):
        self.id_task = id_task
        self.time = time              # preostalo vreme

    def update_time(self, tm):
        self.time -= tm

    def process(self, kvant):
        rad = kvant if self.time > kvant else self.time
        print("Proces", self.id_task, "radi", rad, "-
 ostaje", self.time - rad)
```

time

Preostalo vreme posla; smanjuje se posle svakog kvanta.

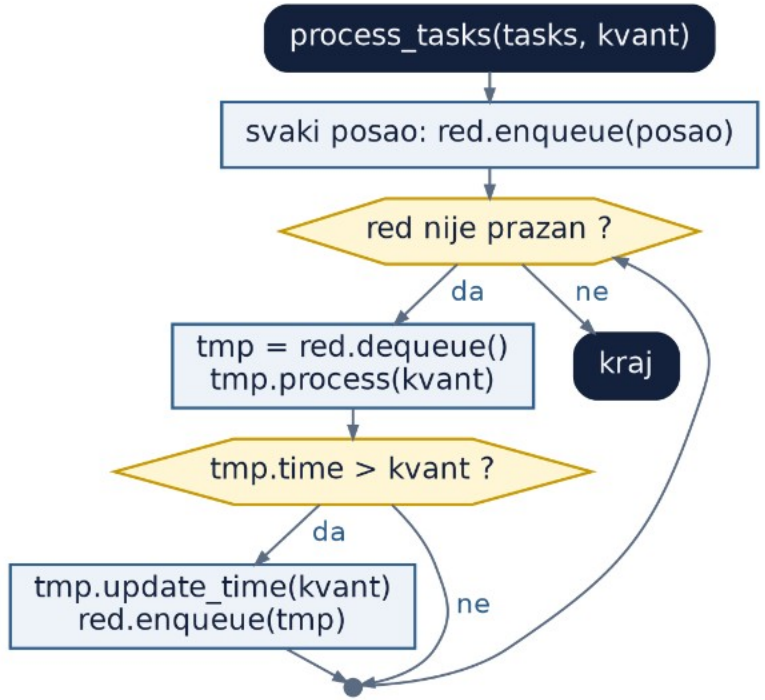
process

Posao radi kvant, ili manje ako mu treba manje.

Kôd i dijagram toka

PYTHON

```
def process_tasks(tasks, kvant):
    red = DQueue()
    for i in range(len(tasks)):
        red.enqueue(tasks[i])
    while not red.is_empty():
        tmp = red.dequeue()
        tmp.process(kvant)
        if tmp.time > kvant:
            tmp.update_time(kvant)
            red.enqueue(tmp)              # nije gotov: na kraj red
```



ZA VEŽBU

Zadaci sa potpunim rešenjima u materijalu

1

Provera uparenosti zagrada u izrazu (stek).

2

Izračunavanje izraza u postfiksnom zapisu (stek).

3

Josifov problem: ispadanje iz kruga (red).

4

Red napravljen od dva steka.

5

Najkraći put kroz lavirint (red).