Algoritmi in podatkovne strukture 1

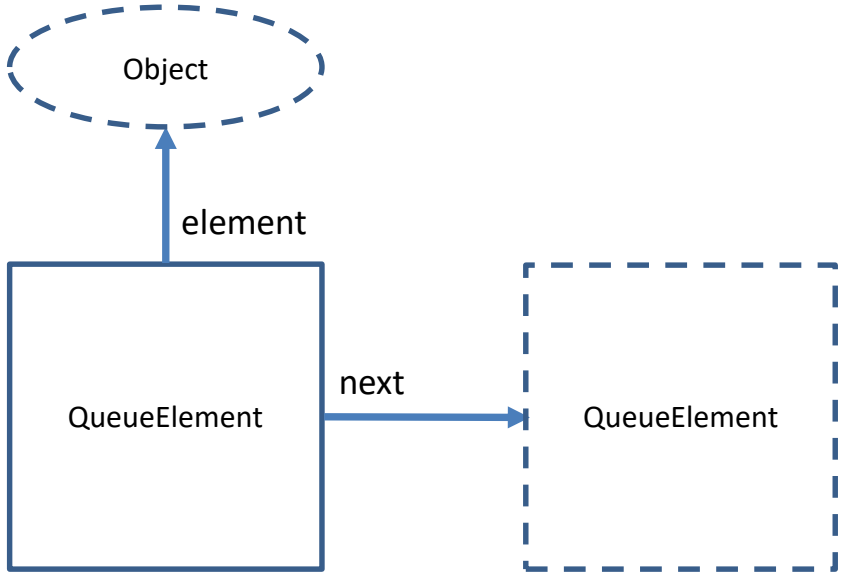
Laboratorijske vaje

Vrsta in sklad

VRSTA

```
class QueueElement
{
    Object element;
    QueueElement next;
    …
}
```

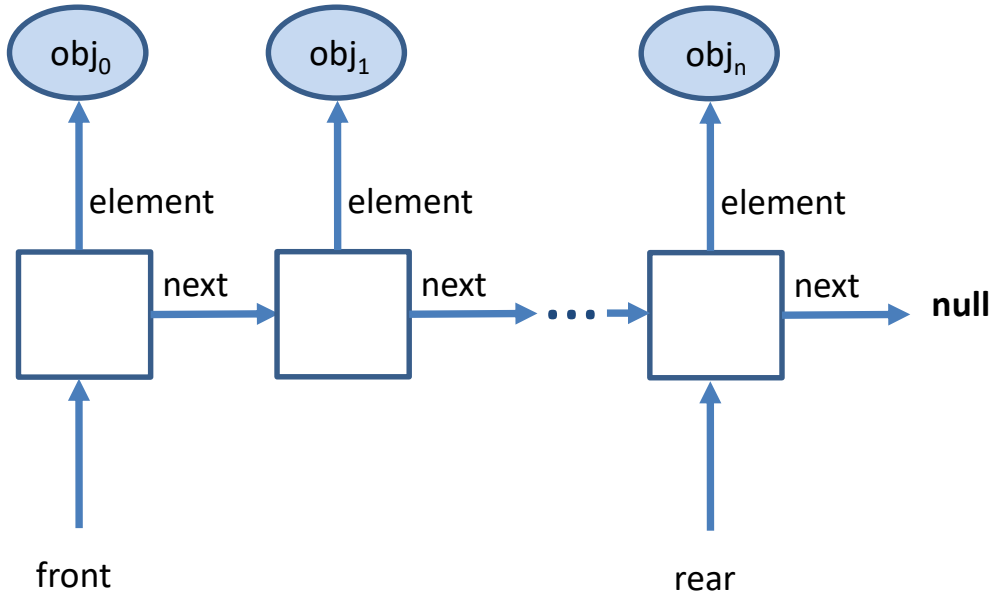
```
class Queue
{
    QueueElement front;
    QueueElement rear;
    …
}
```



VRSTA

Osnovne operacije:

- enqueue
- front
- dequeue



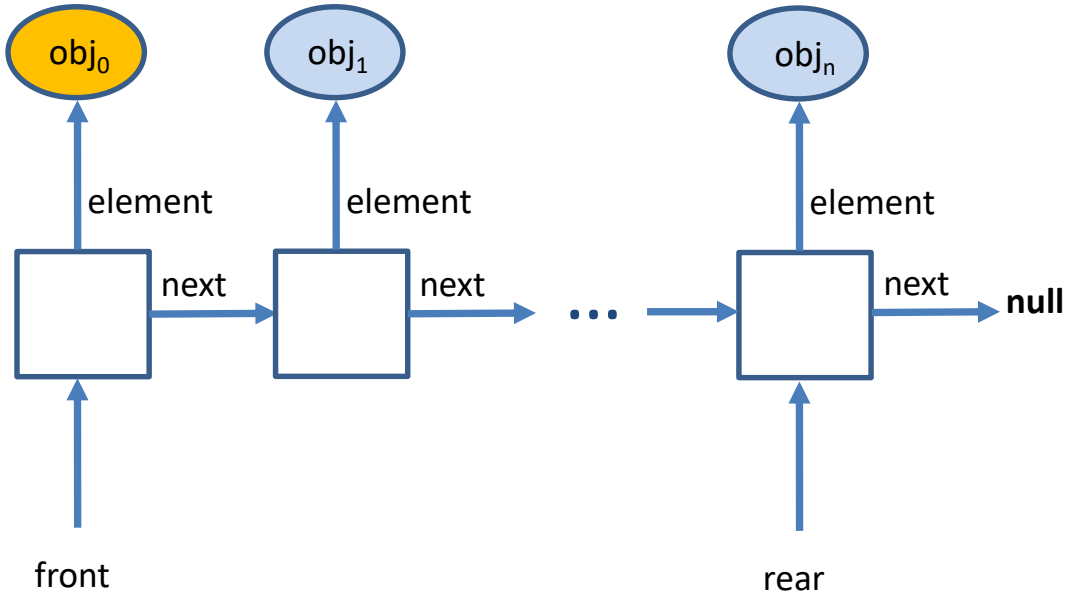
NALOGE

Implementirajte naslednje metode v razredu Queue:

- `Object front()` – vrne začetni element vrste (elementa ne odstrani!)
- `void enqueue(Object obj)` – doda element na konec vrste
- `void dequeue()` - odstrani začetni element vrste

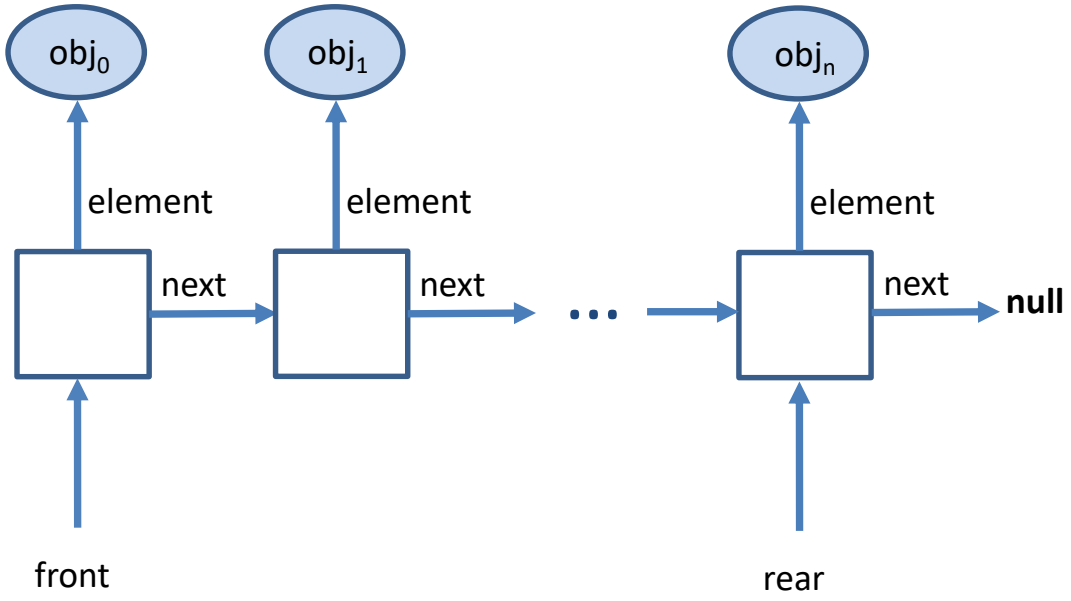
NALOGE

`Object front()` – vrne začetni element vrste (elementa ne odstrani!)



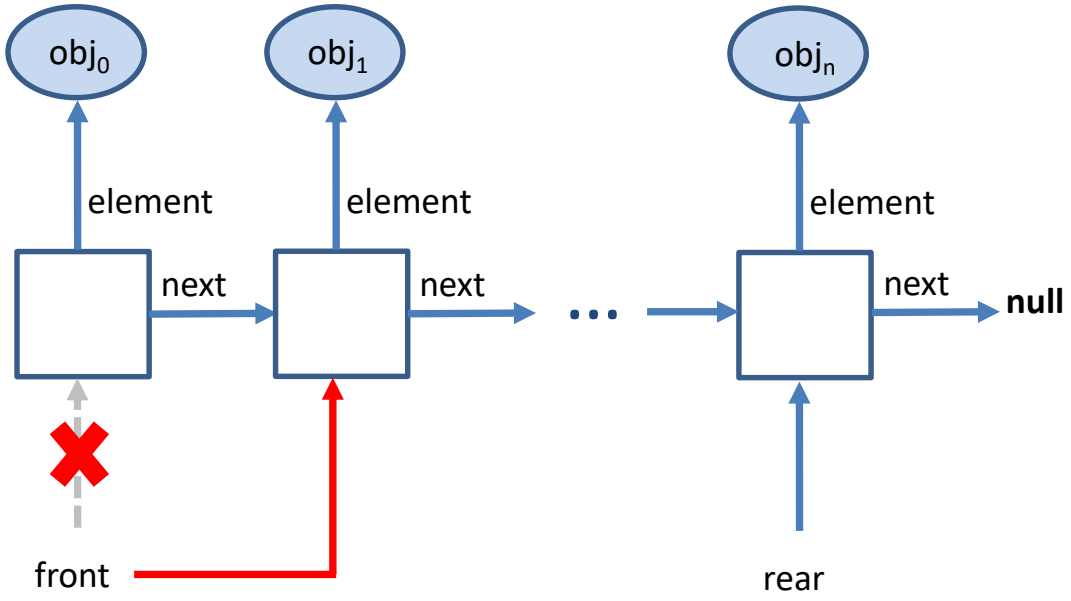
NALOGE

`void dequeue(Object obj)` – odstrani začetni element vrste



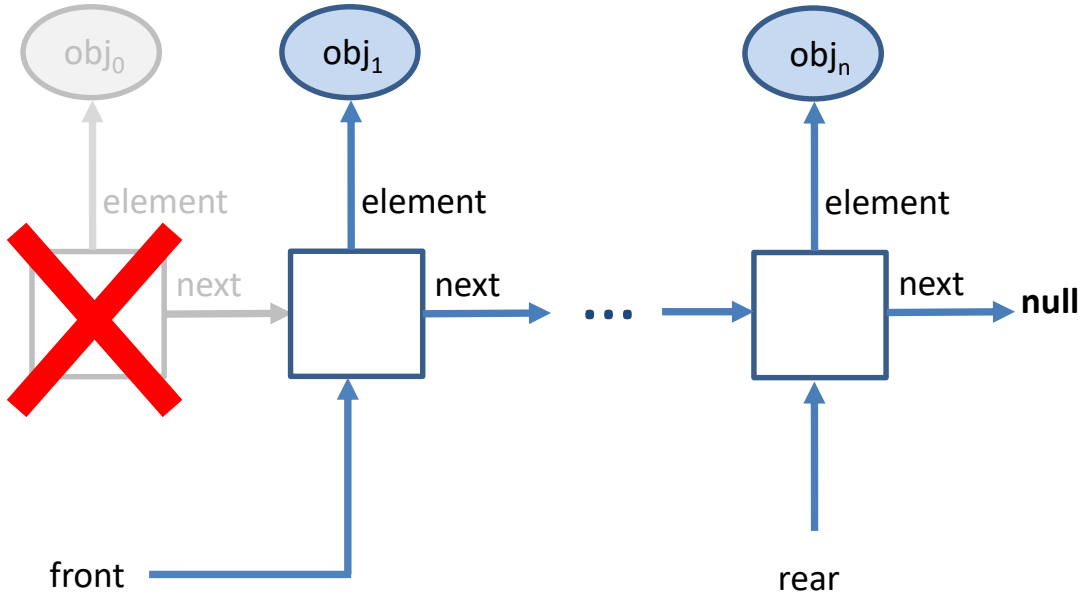
NALOGE

`void dequeue(Object obj)` – odstrani začetni element vrste



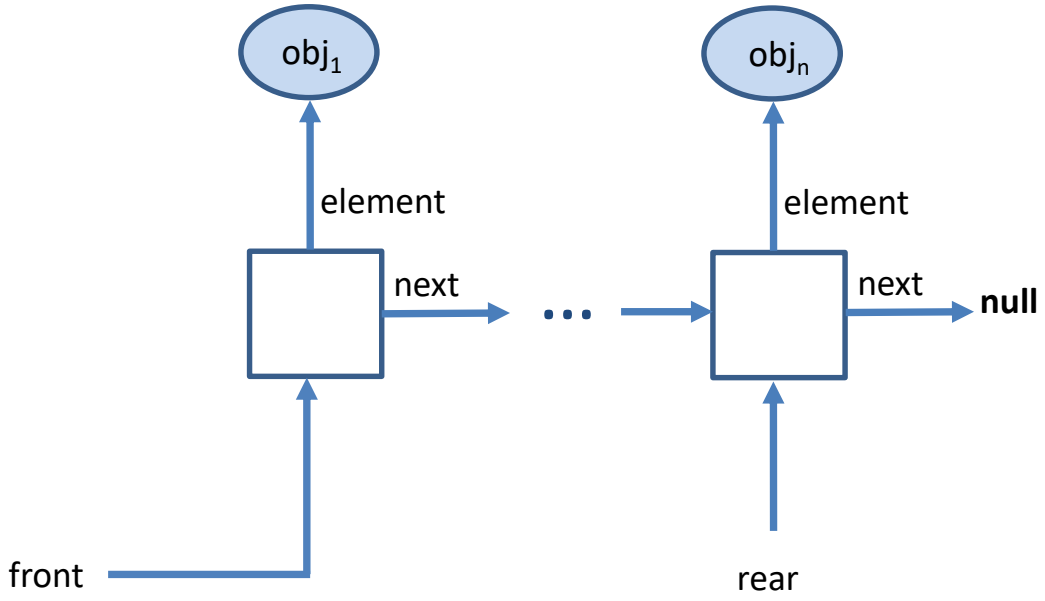
NALOGE

`void dequeue(Object obj)` – odstrani začetni element vrste



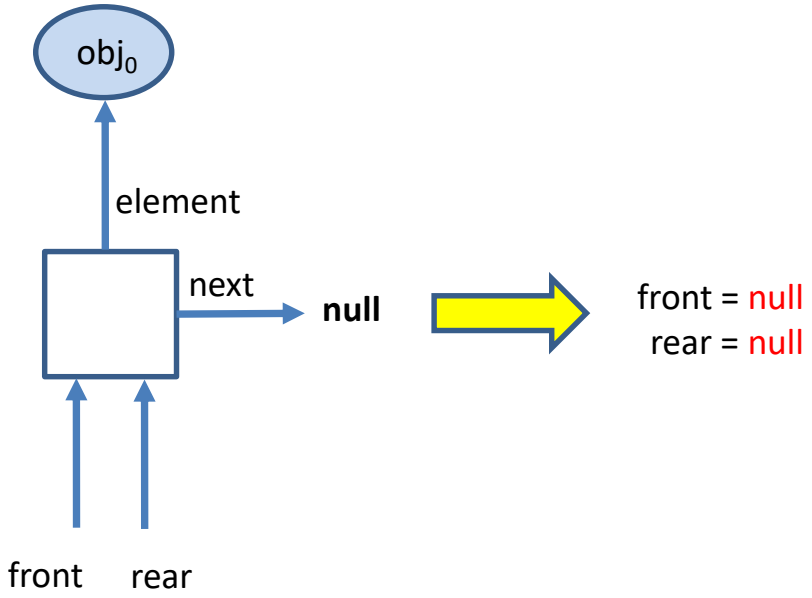
NALOGE

`void dequeue(Object obj)` – odstrani začetni element vrste



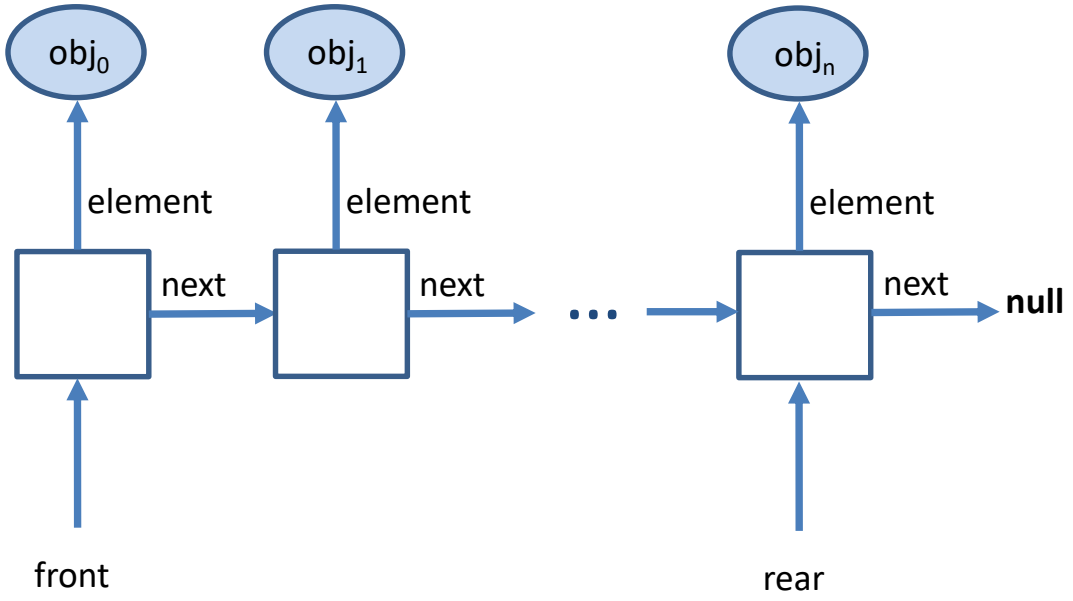
NALOGE

`void dequeue(Object obj)` – odstrani začetni element vrste



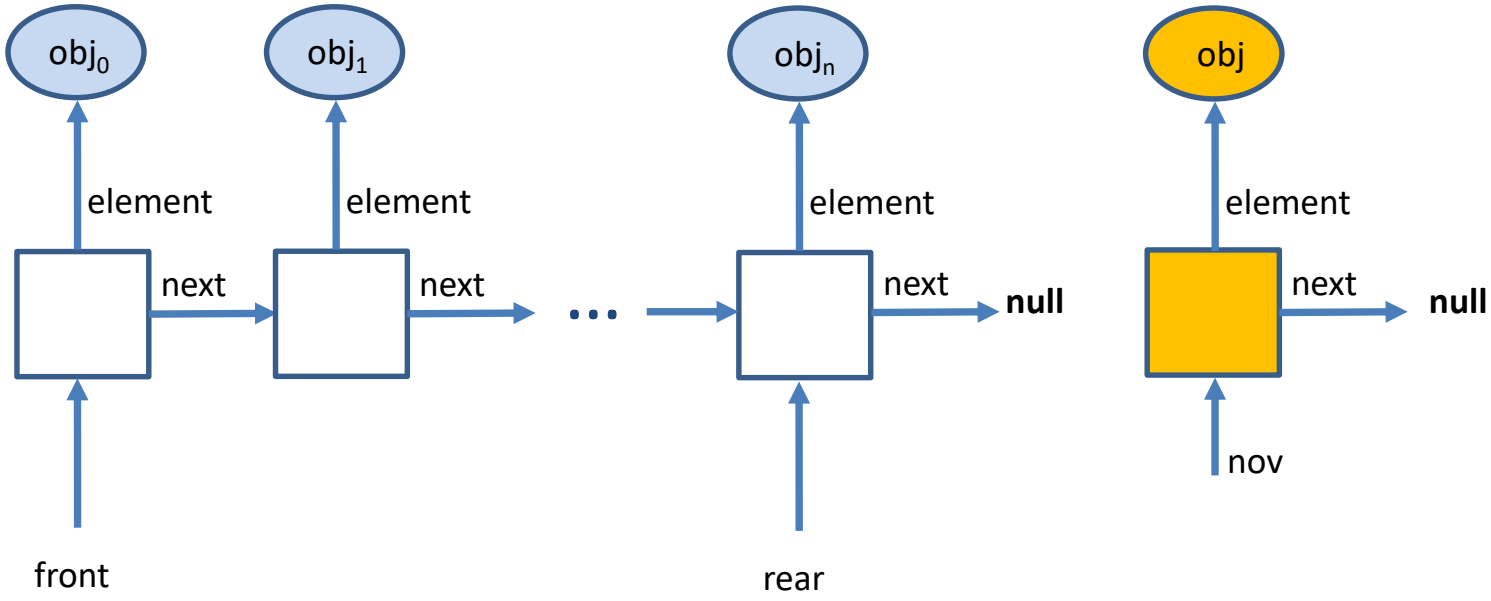
NALOGE

`void enqueue(Object obj)` – doda element na konec vrste



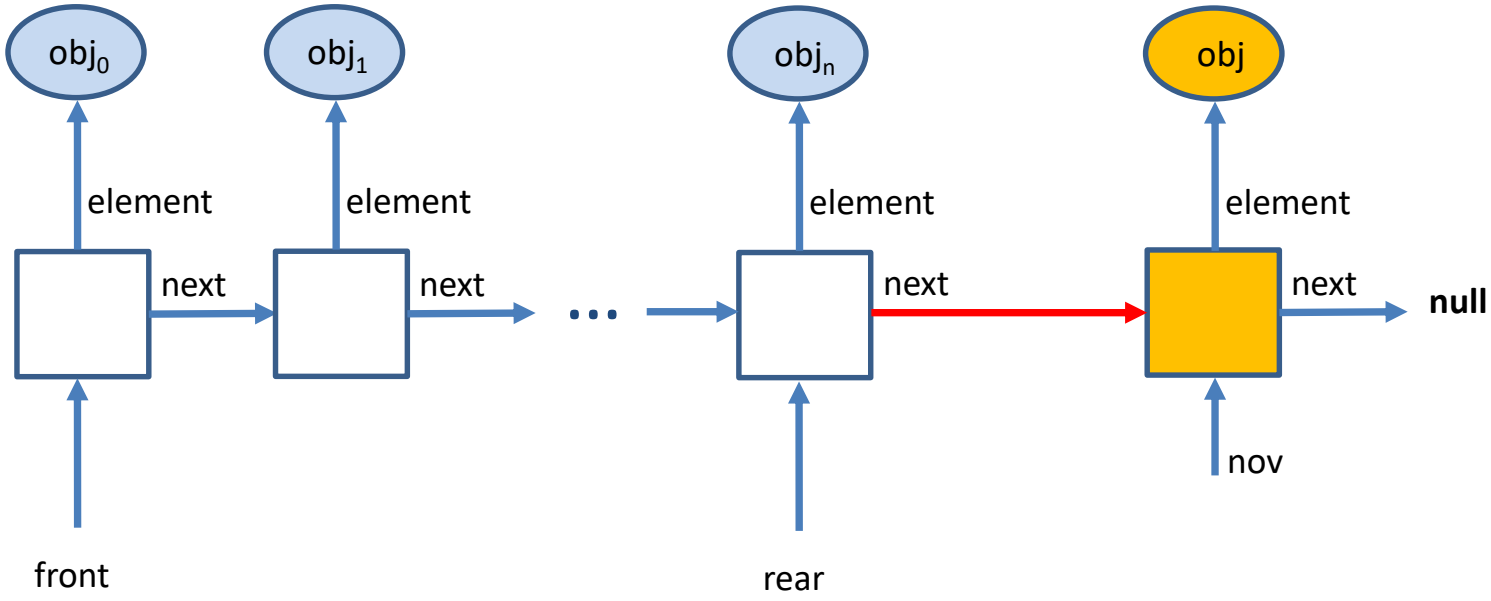
NALOGE

`void enqueue(Object obj)` – doda element na konec vrste



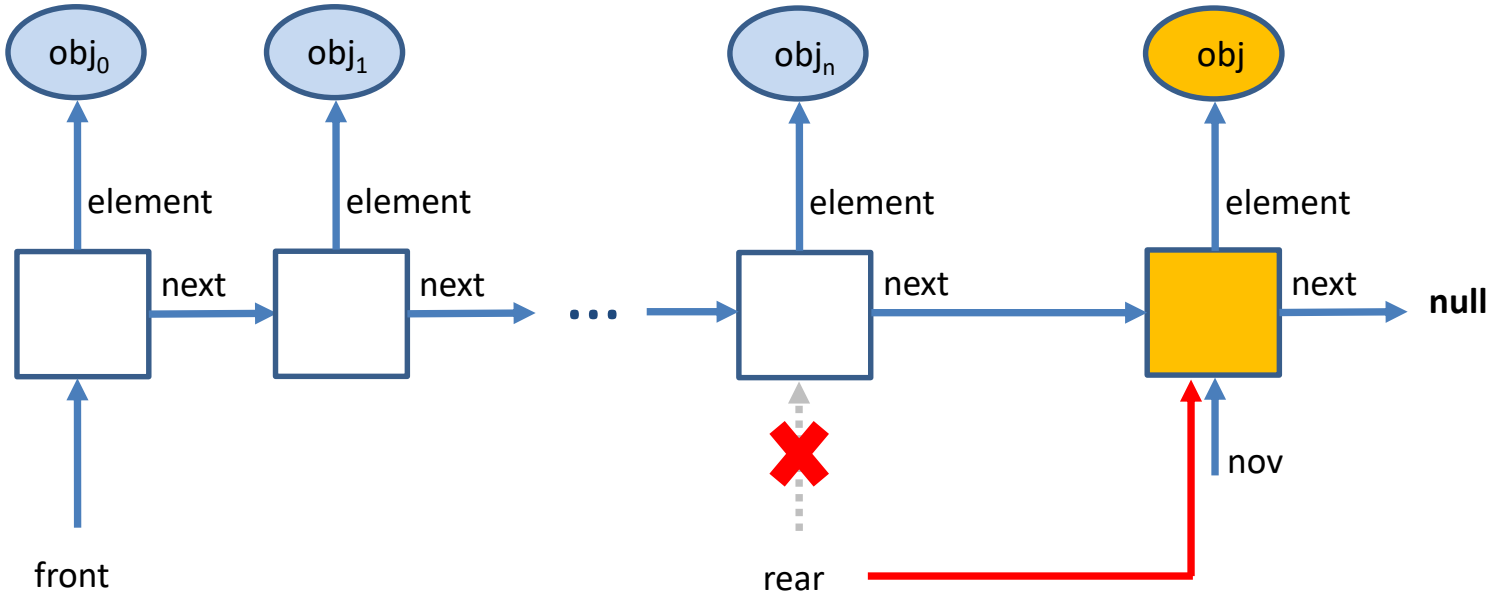
NALOGE

`void enqueue(Object obj)` – doda element na konec vrste



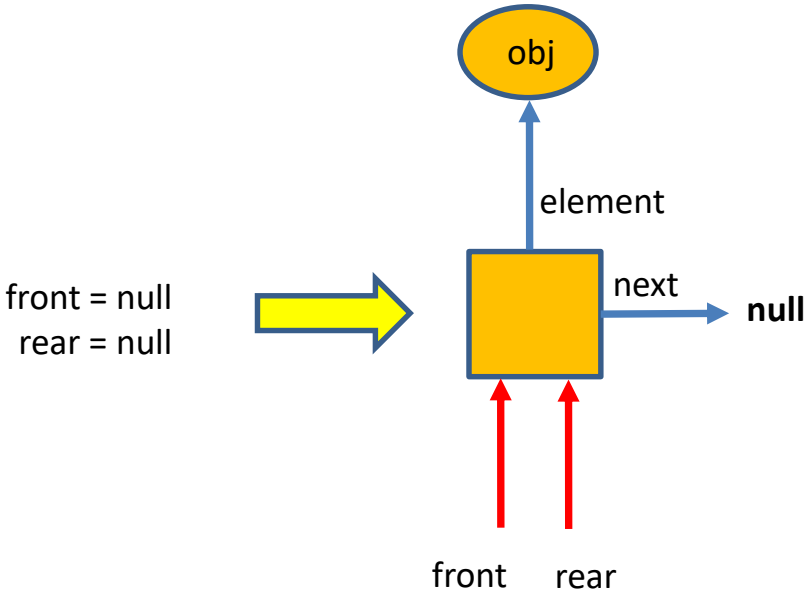
NALOGE

`void enqueue(Object obj)` – doda element na konec vrste

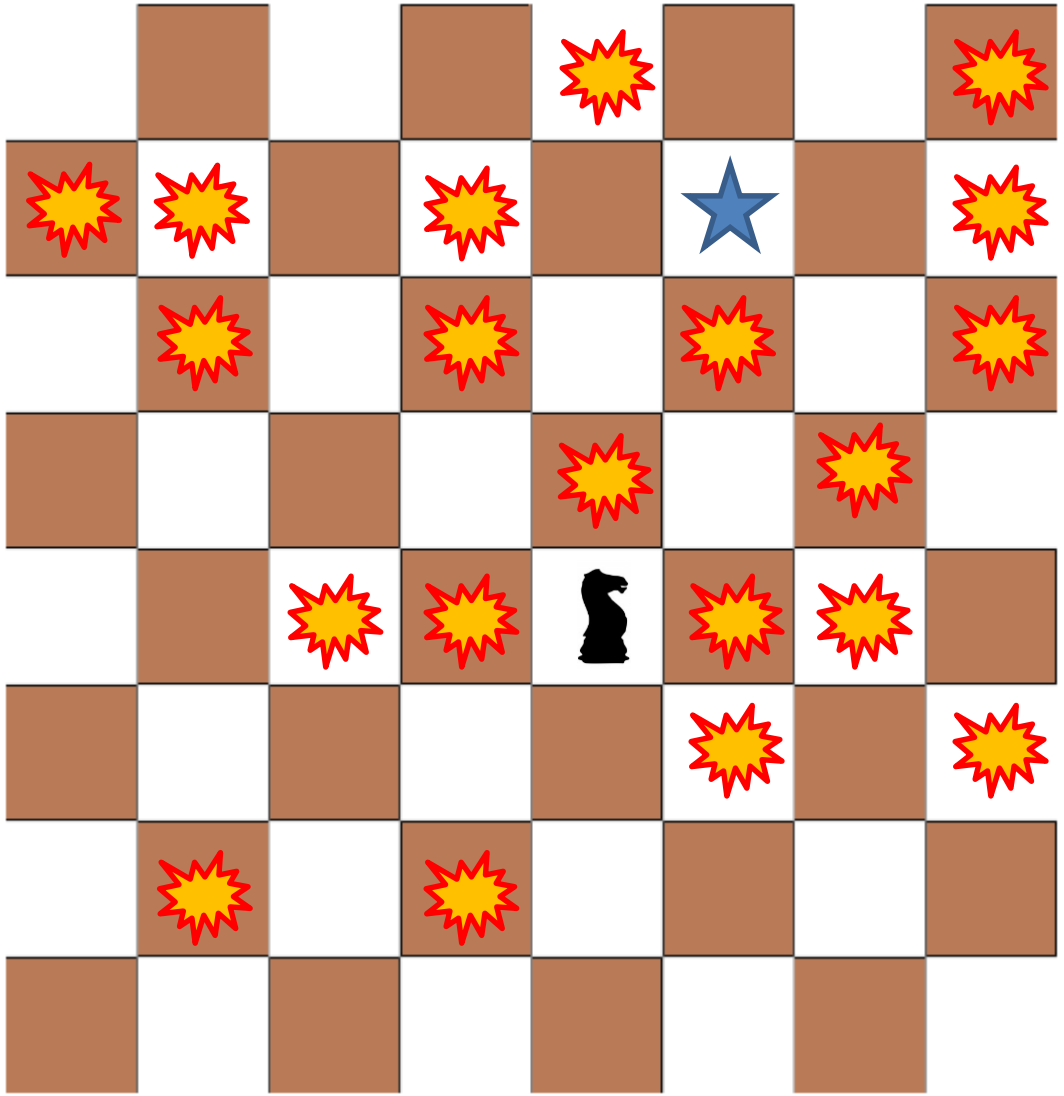


NALOGE

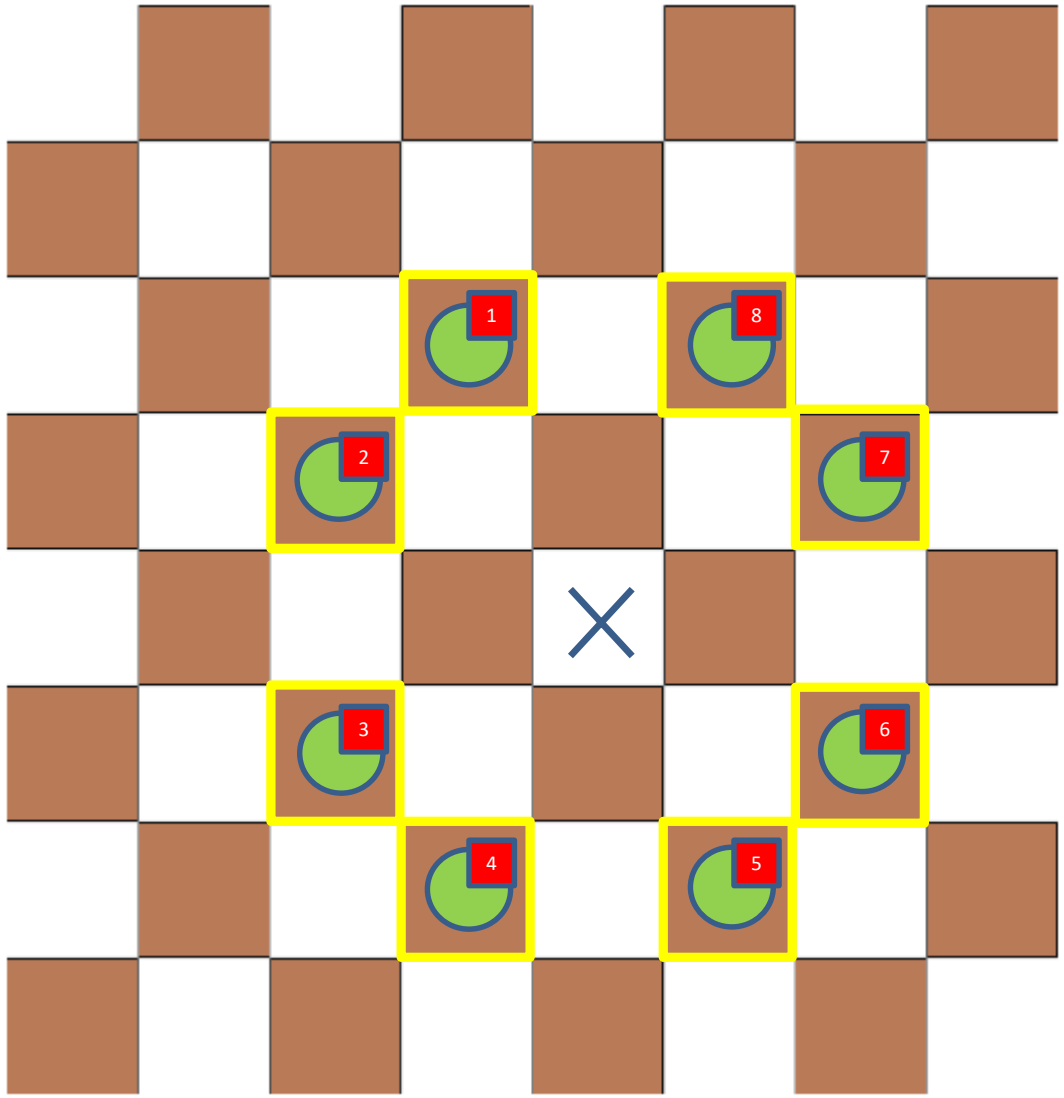
`void enqueue(Object obj)` – doda element na konec vrste

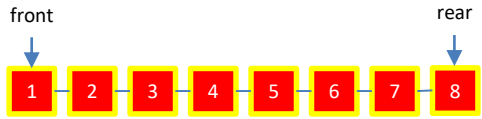


APLIKACIJA: NAJKRAJŠA POT

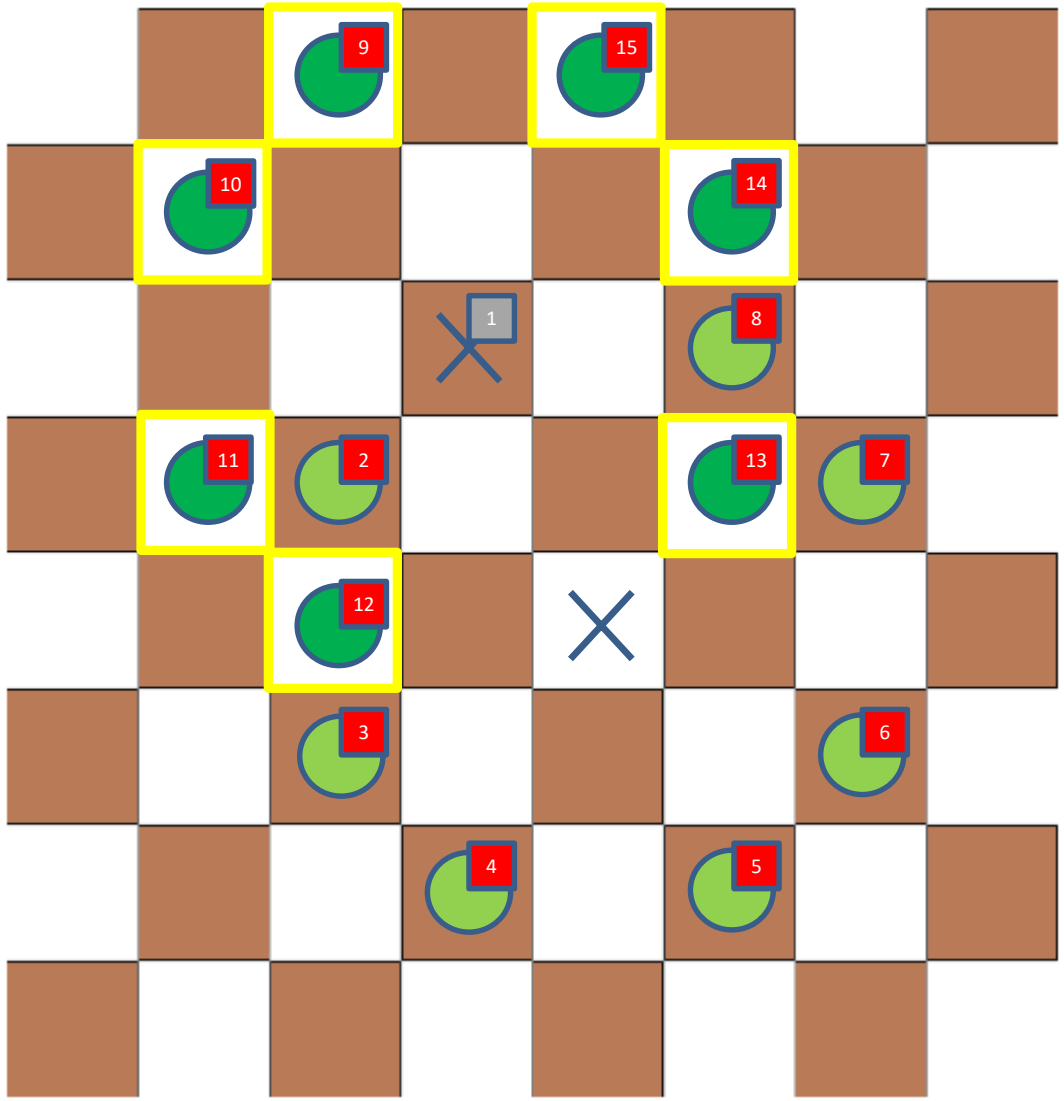


IDEJA: NAJPREJ PREGLEDAMO POLJA, KI SO DOSEGLJIVA Z ENIM SKOKOM...

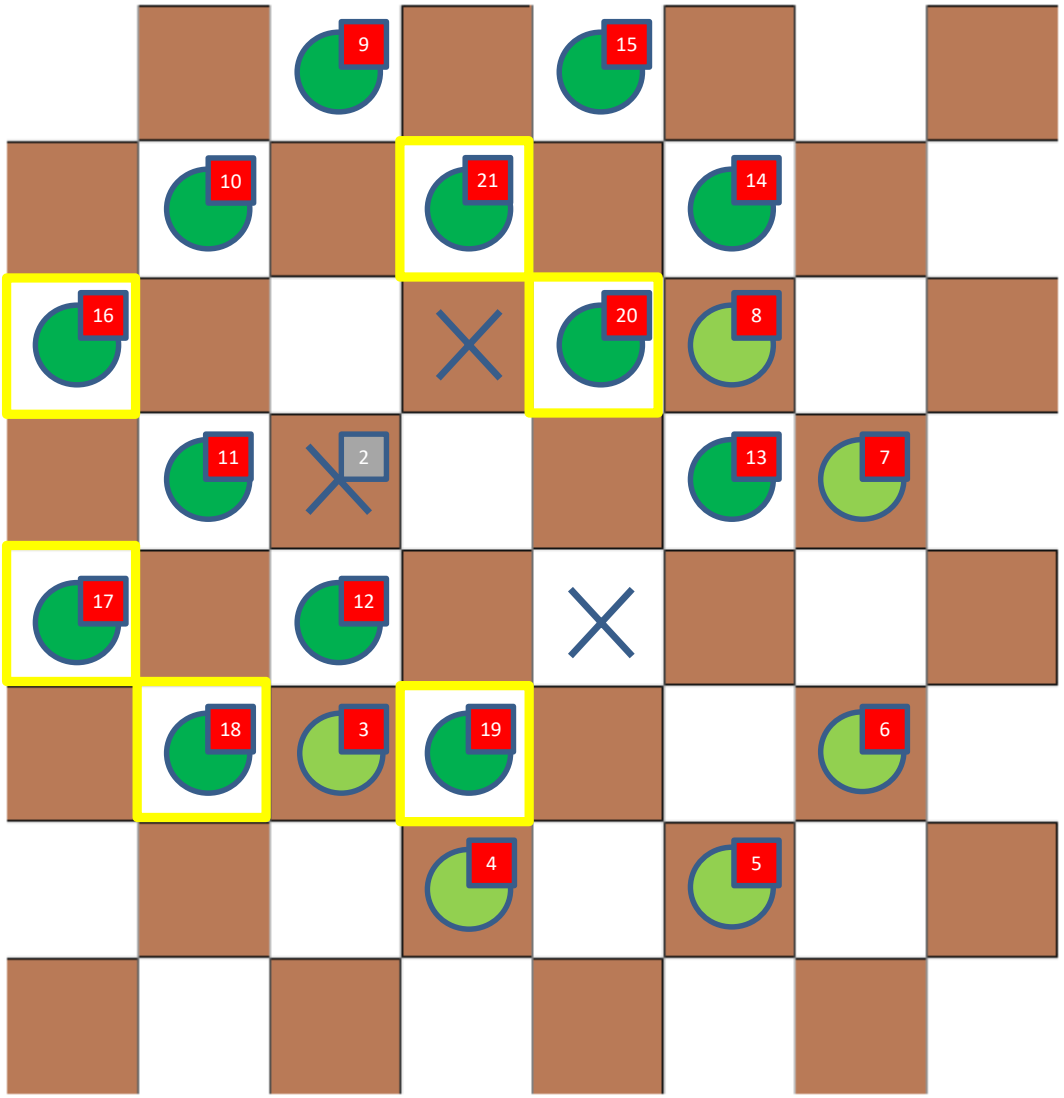


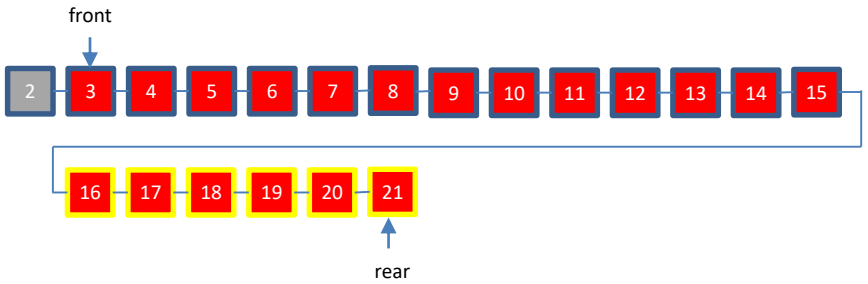


IDEJA: NATO PREGLEDUJEMO POLJA, KI SO DOSEGLJIVA Z DVEMA SKOKOMA...

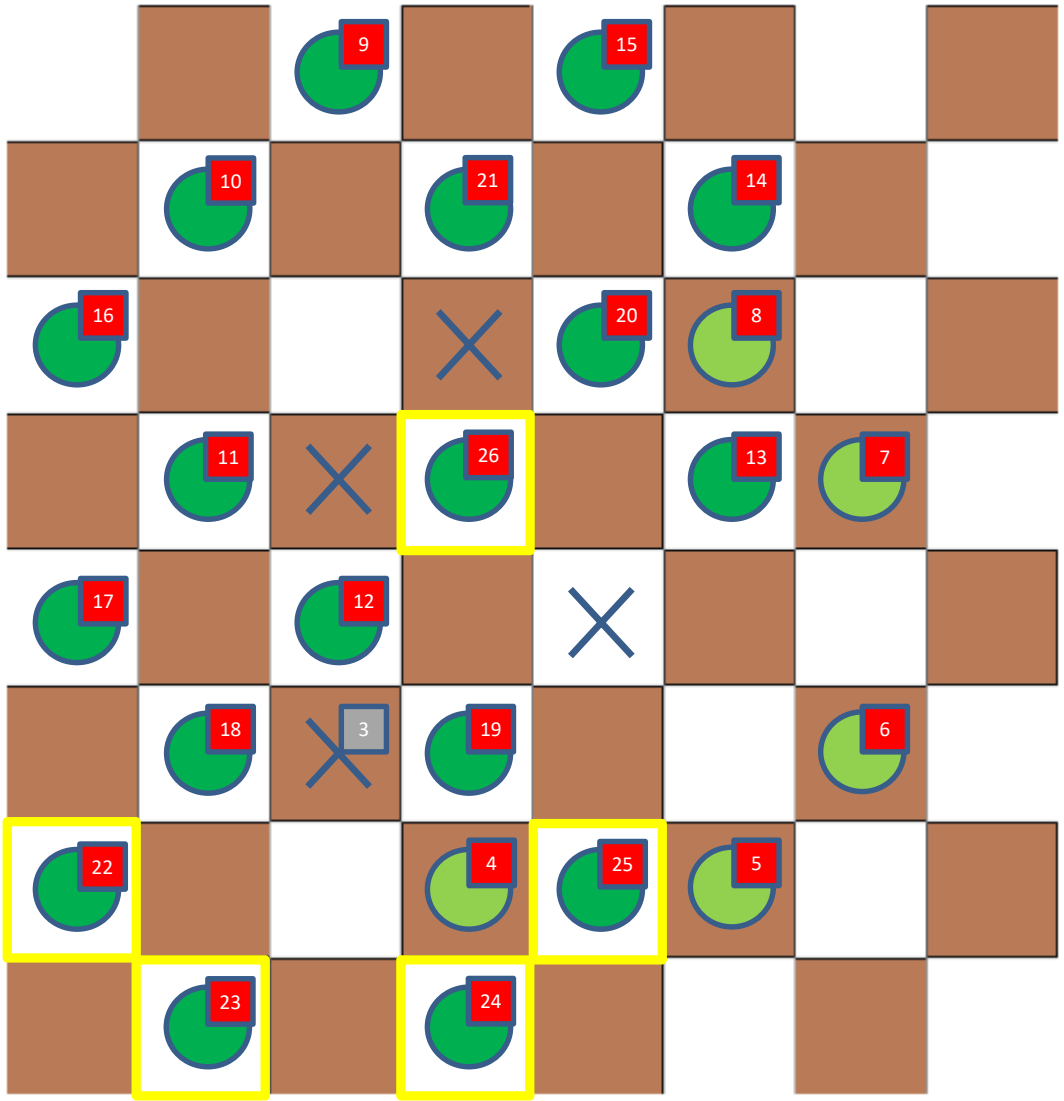


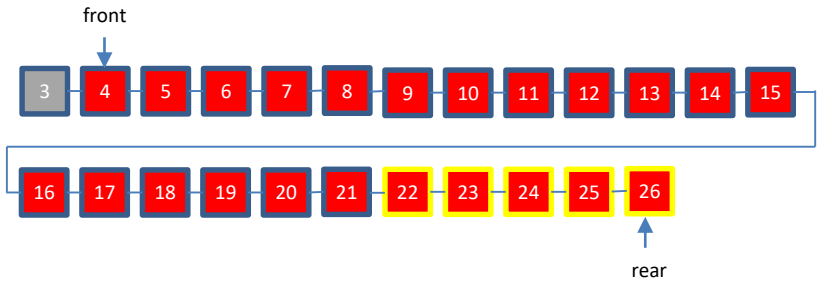
IDEJA: NATO PREGLEDUJEMO POLJA, KI SO DOSEGLJIVA Z DVEMA SKOKOMA...



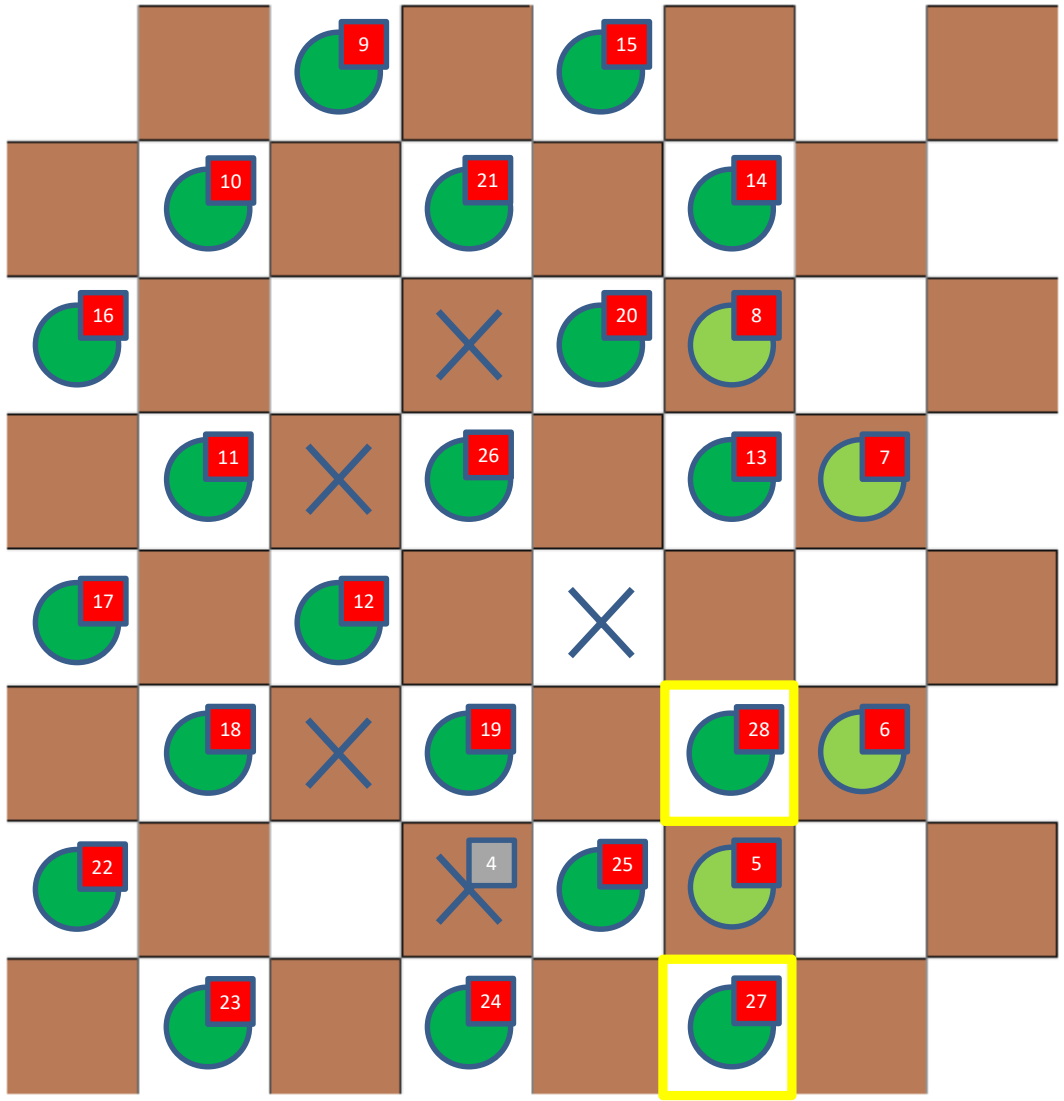


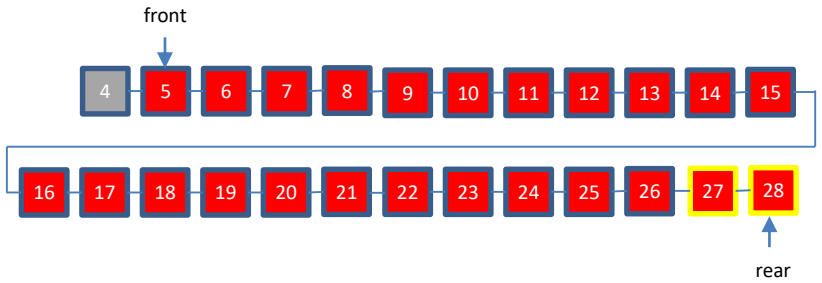
IDEJA: NATO PREGLEDUJEMO POLJA, KI SO DOSEGLJIVA Z DVEMA SKOKOMA...



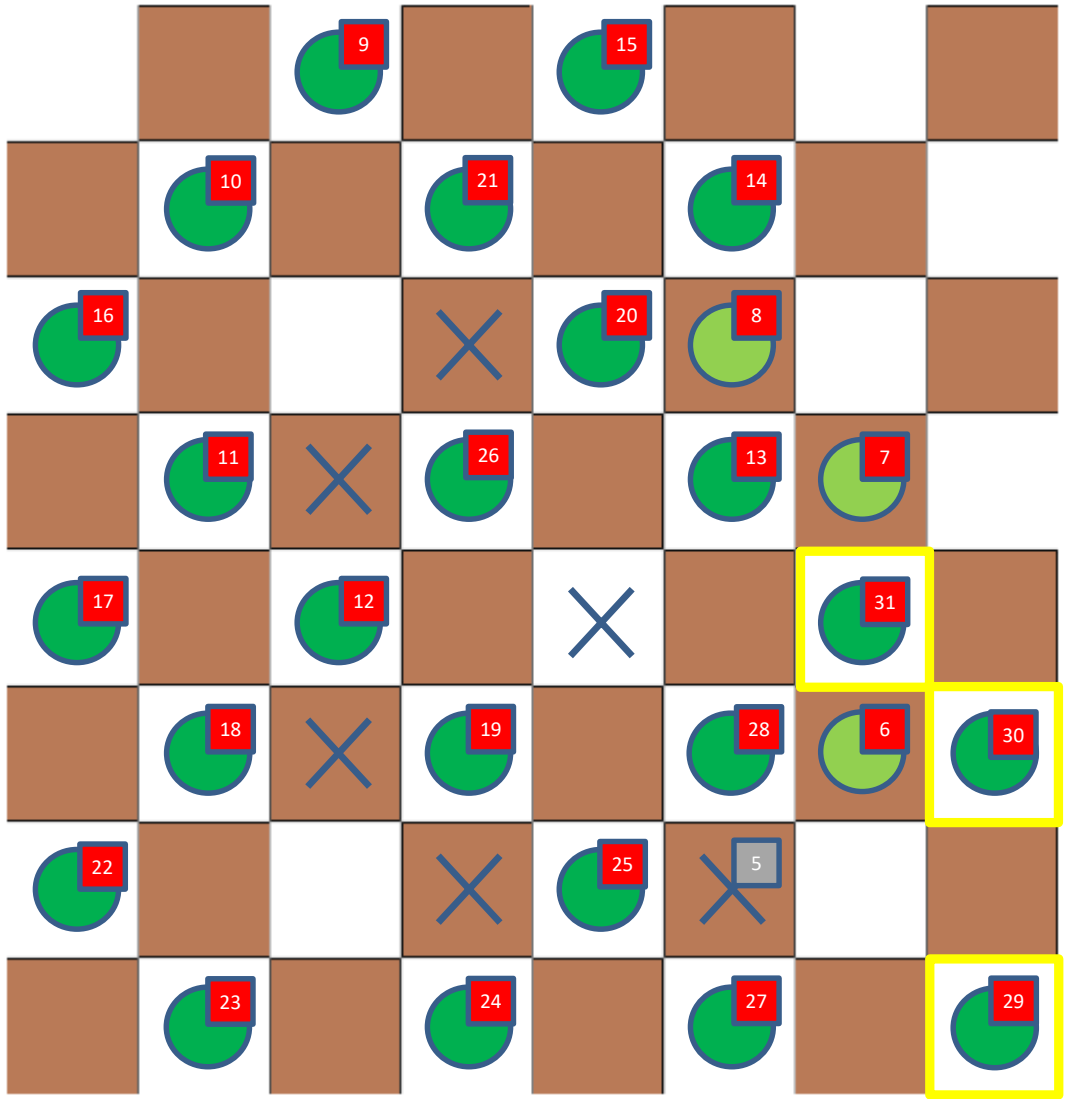


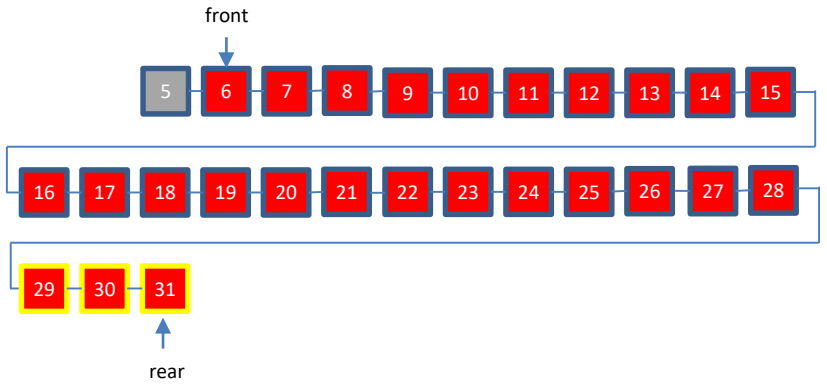
IDEJA: NATO PREGLEDUJEMO POLJA, KI SO DOSEGLJIVA Z DVEMA SKOKOMA...



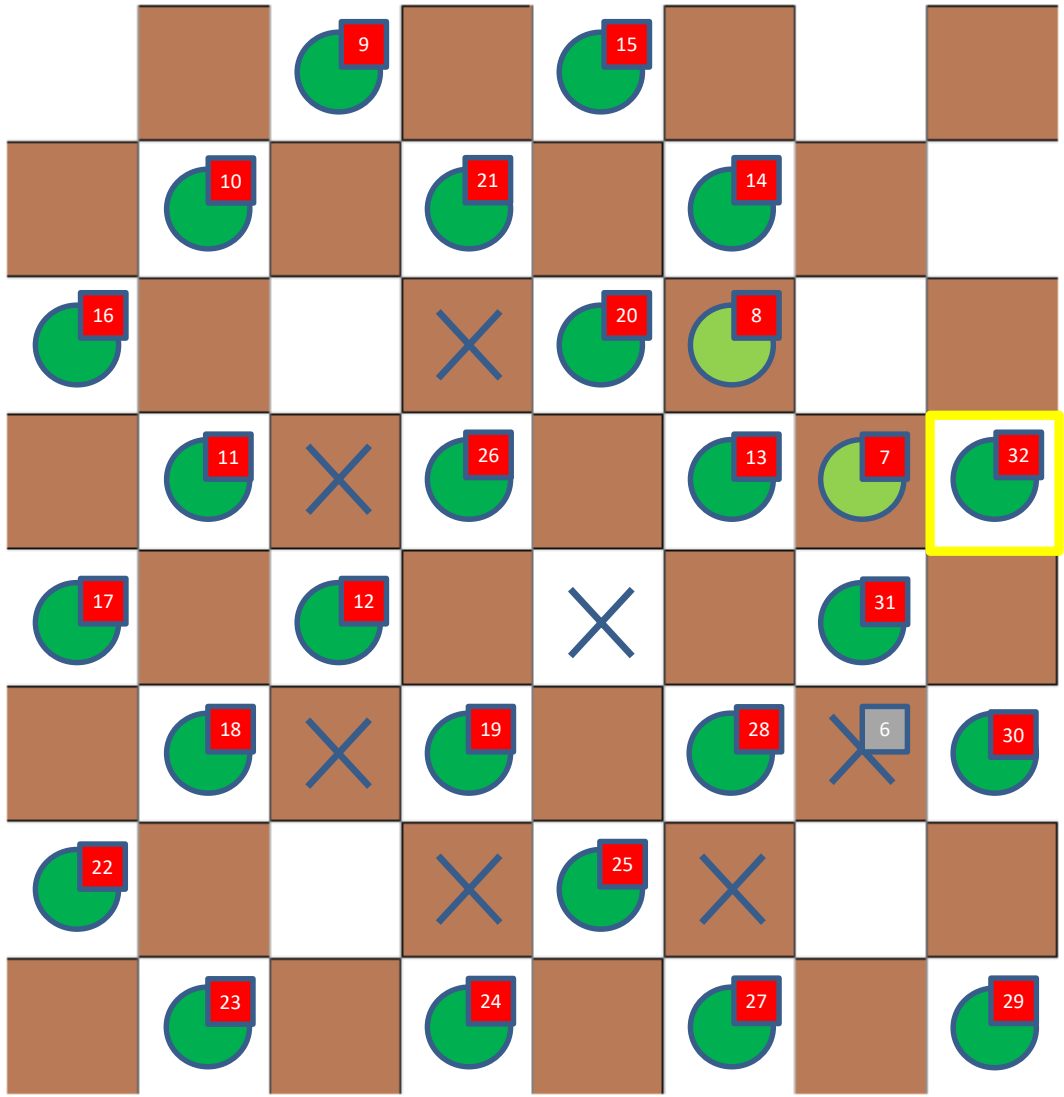


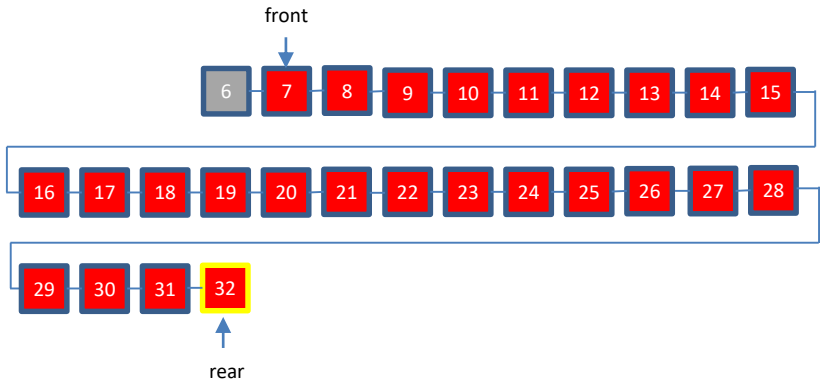
IDEJA: NATO PREGLEDUJEMO POLJA, KI SO DOSEGLJIVA Z DVEMA SKOKOMA...



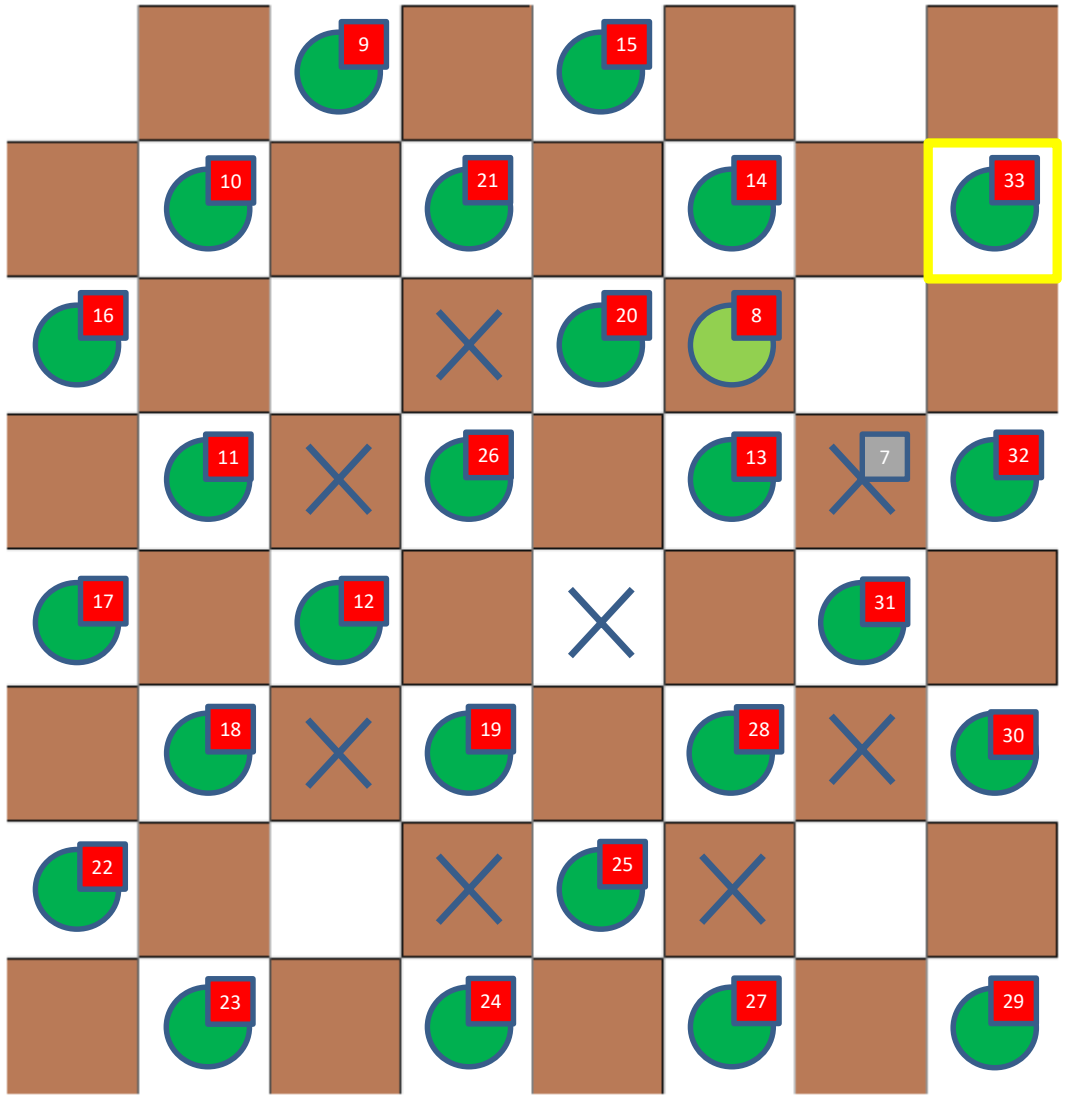


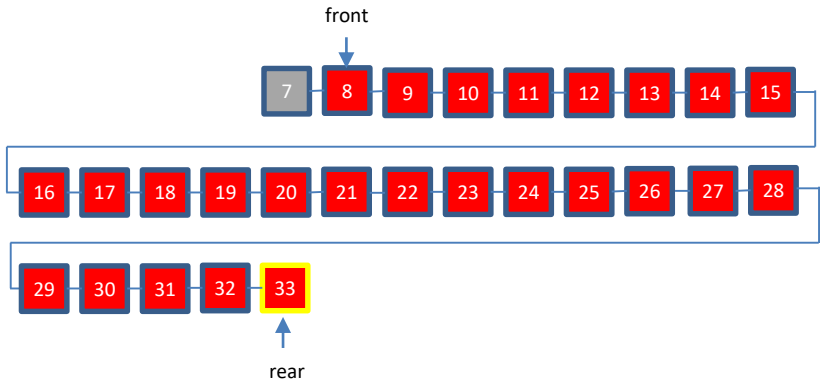
IDEJA: NATO PREGLEDUJEMO POLJA, KI SO DOSEGLJIVA Z DVEMA SKOKOMA...



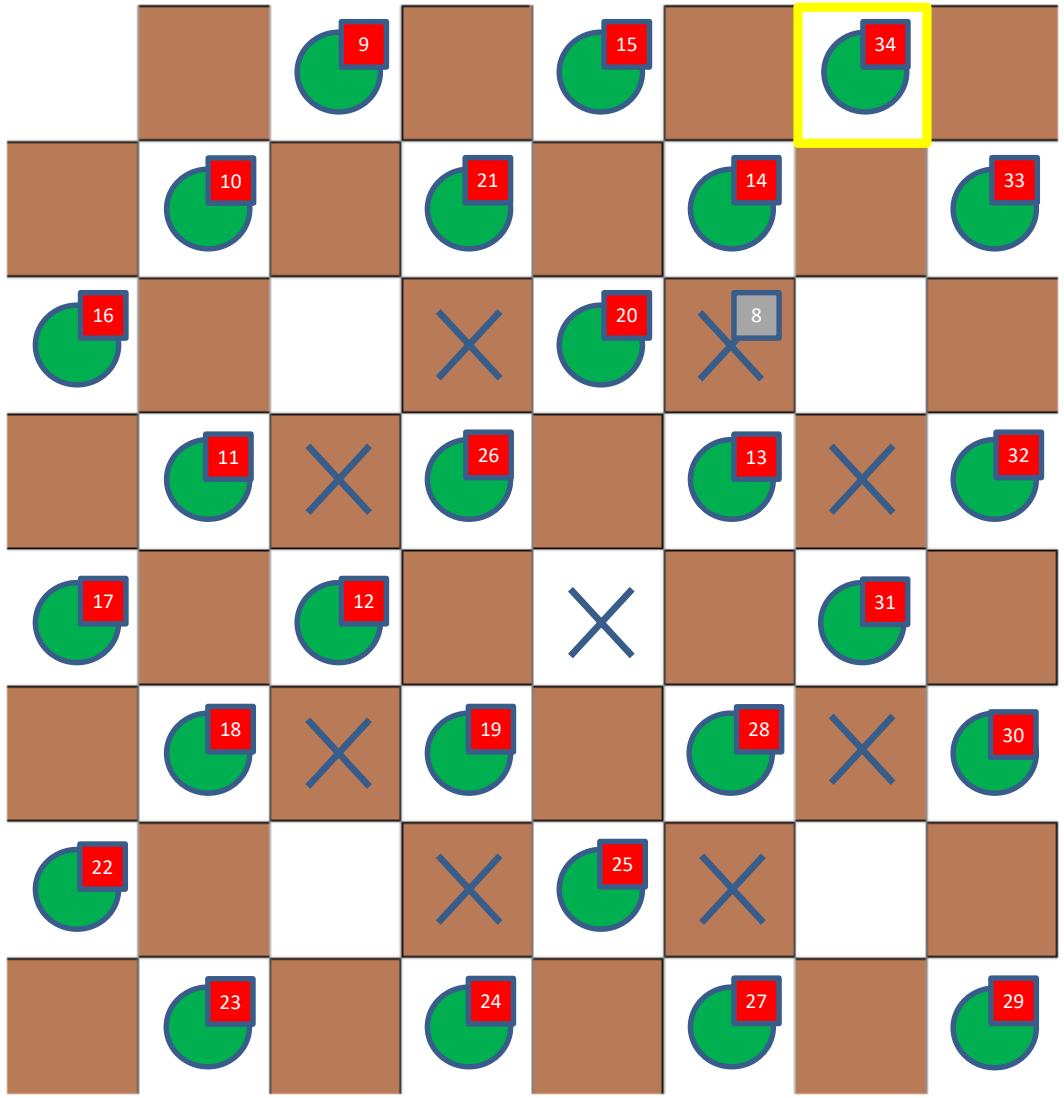


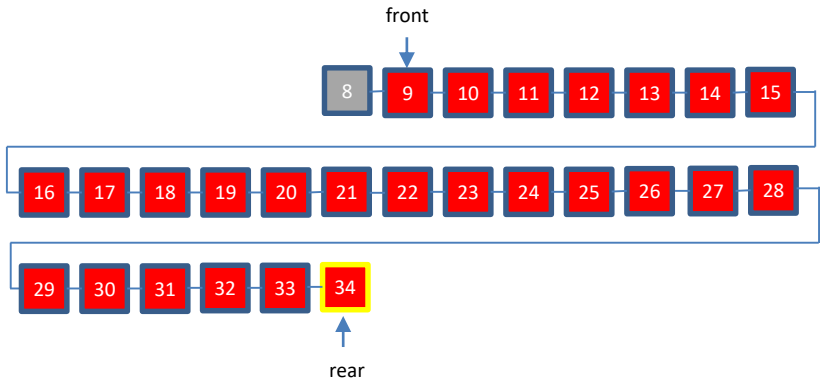
IDEJA: NATO PREGLEDUJEMO POLJA, KI SO DOSEGLJIVA Z DVEMA SKOKOMA...



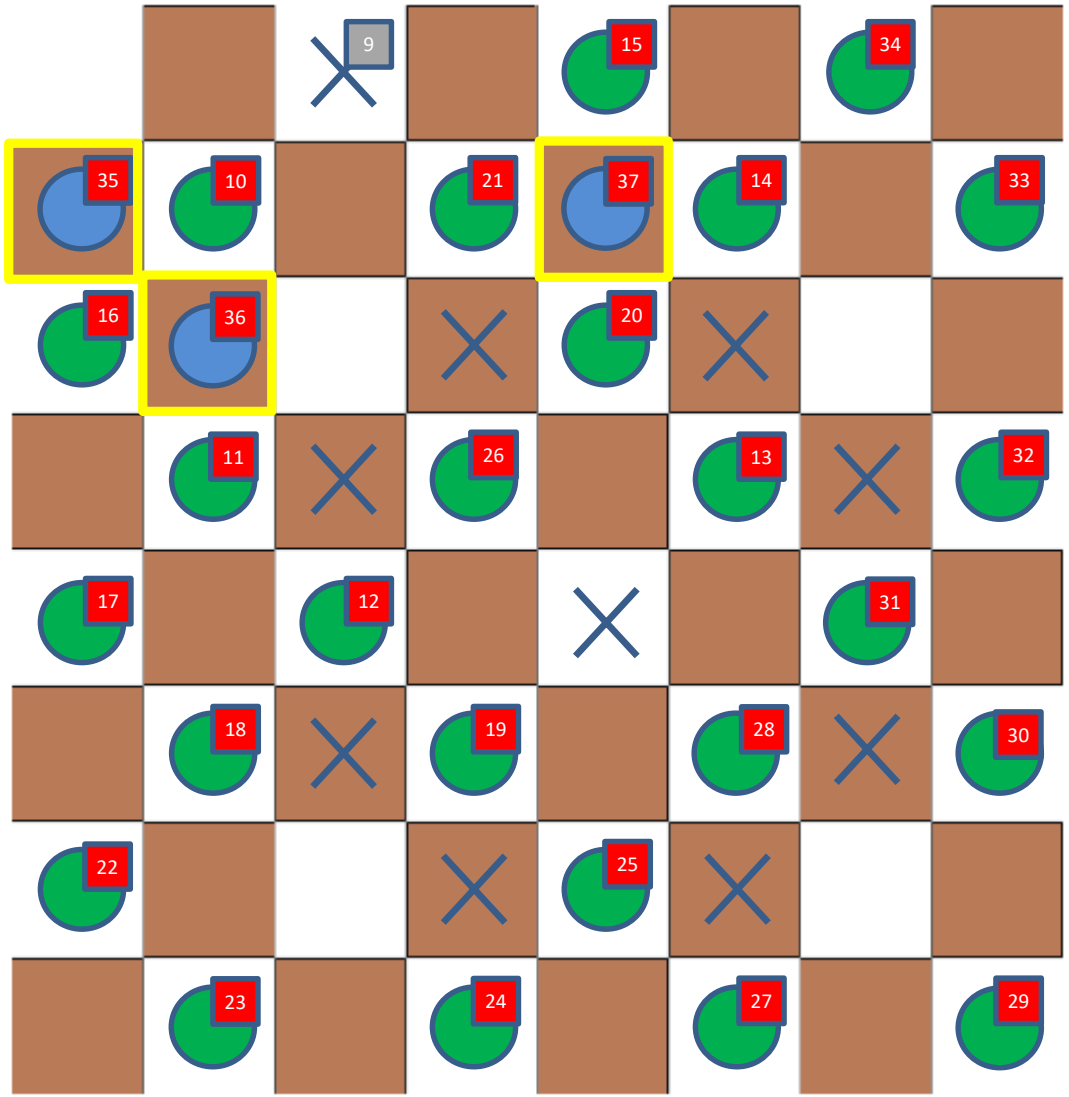


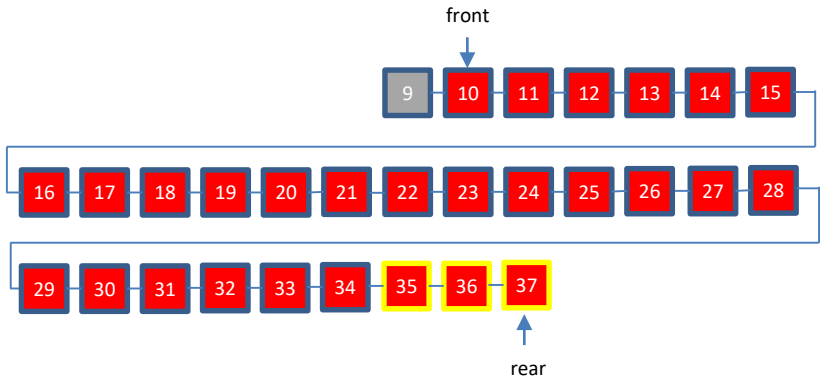
IDEJA: NATO PREGLEDUJEMO POLJA, KI SO DOSEGLJIVA Z DVEMA SKOKOMA...





IDEJA: NATO PREGLEDUJEMO POLJA, KI SO DOSEGLJIVA S TREMI SKOKI...





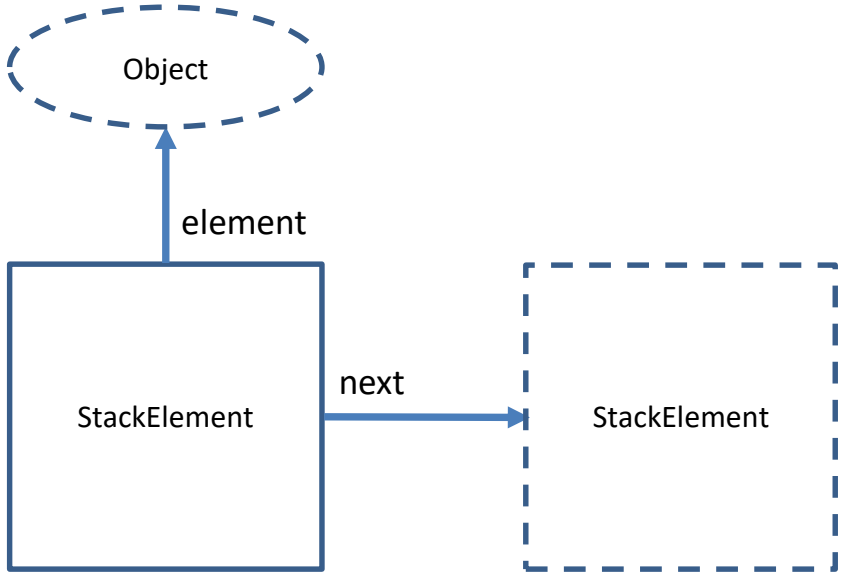
APLIKACIJA: NAJKRAJŠA POT

```
class Konj {
  int x;
  int y;
  String poteze;

  Konj(int x, int y, String poteze) {
    this.x = x;
    this.y = y;
    this.poteze = poteze + " -> [" + x + "," + y + "]";
  }
}


public class NajkrajsaPot {

  public static void main(String[] args) {
    ...
    Queue queue = new Queue();
    Konj k = new Konj(zacetni_x, zacetni_y, "");
    queue.enqueue(k);
    ...

    while (!queue.empty()) {
      k = (Konj)queue.front();
      queue.dequeue();

      ...
      queue.enqueue(new Konj(k.x + ..., k.y + ..., k.poteze));
      ...
  }
}
```

SKLAD

```
class StackElement
{
    Object element;
    StackElement next;
    …
}
```

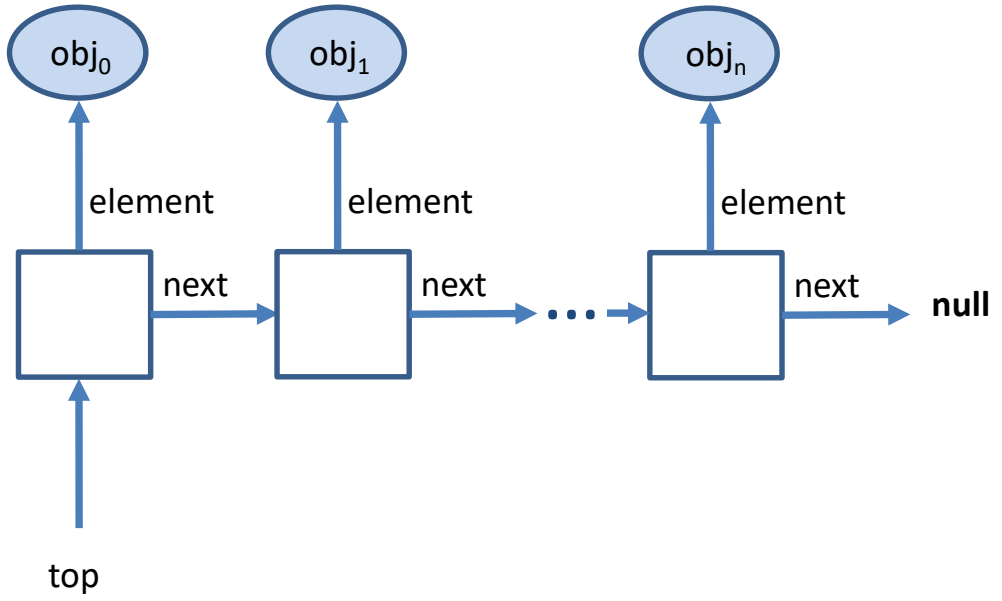
```
class Stack
{
    StackElement top;
    …
}
```



SKLAD

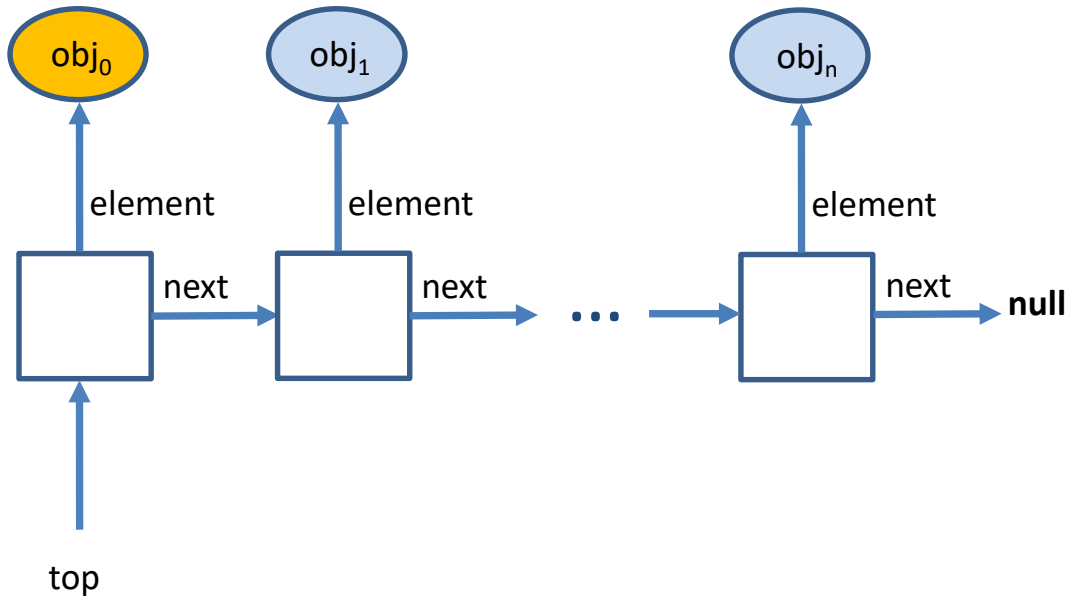
Osnovne operacije:

- push
- top
- pop



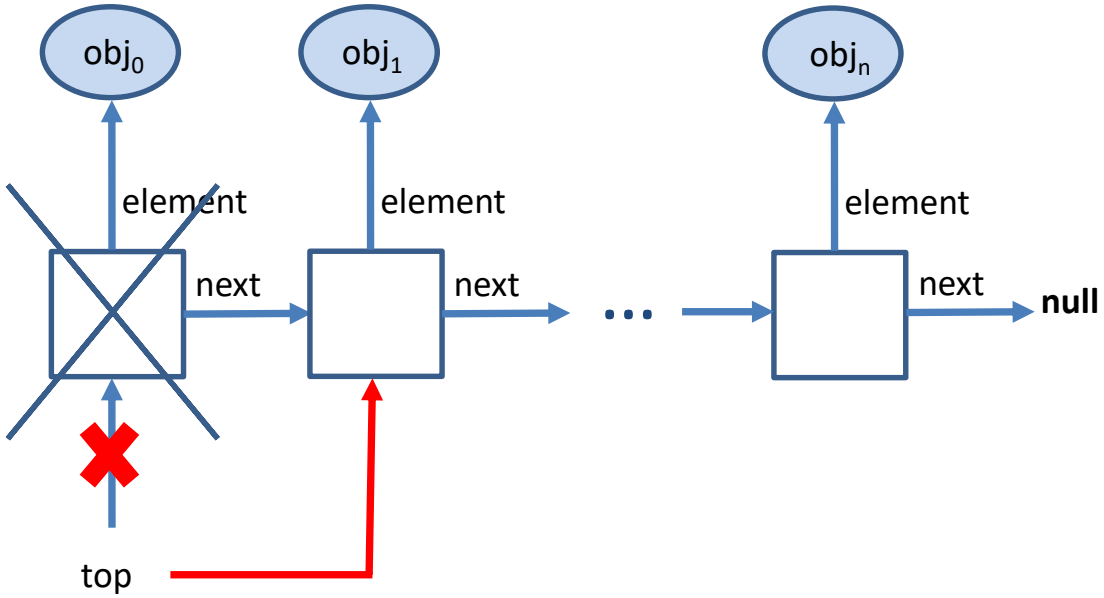
NALOGE

`Object top()` – vrne vrhnji element sklada (elementa ne odstrani!)



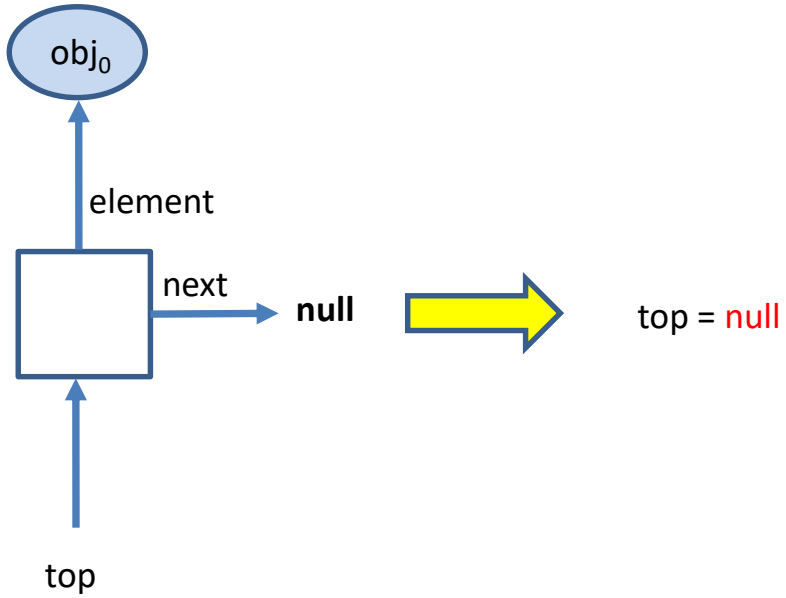
NALOGE

`void pop()` - odstrani element z vrha sklada



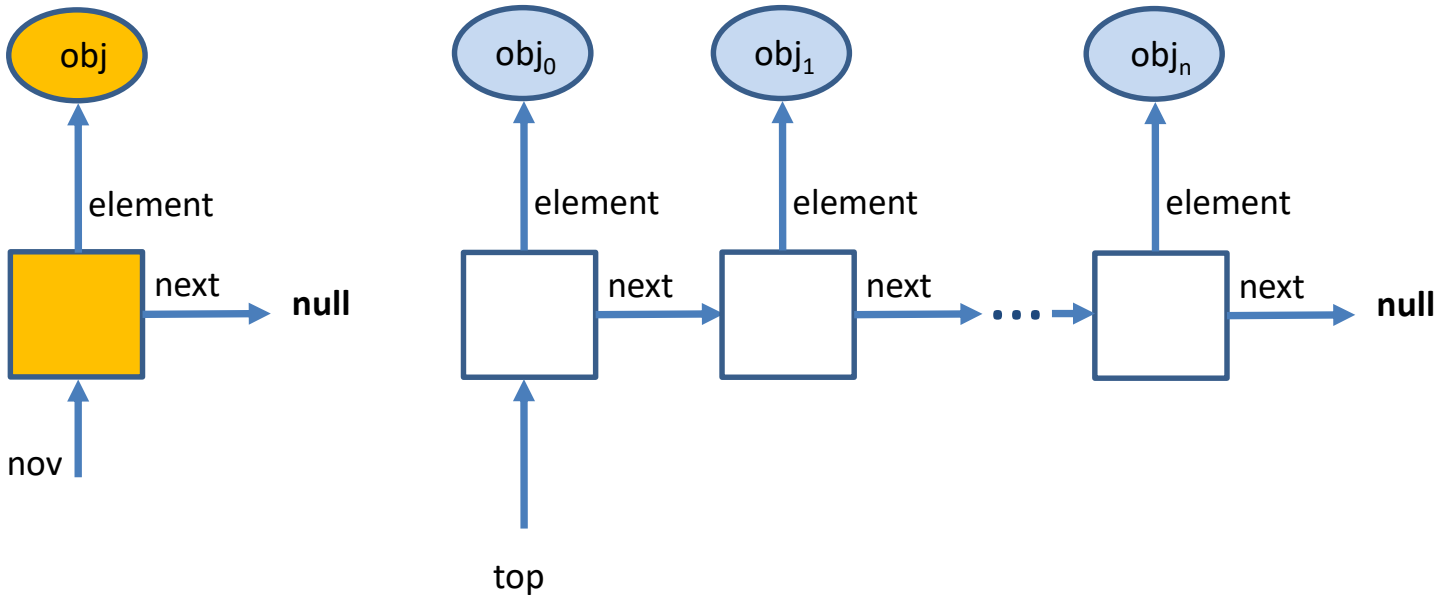
NALOGE

`void pop()` - odstrani element z vrha sklada



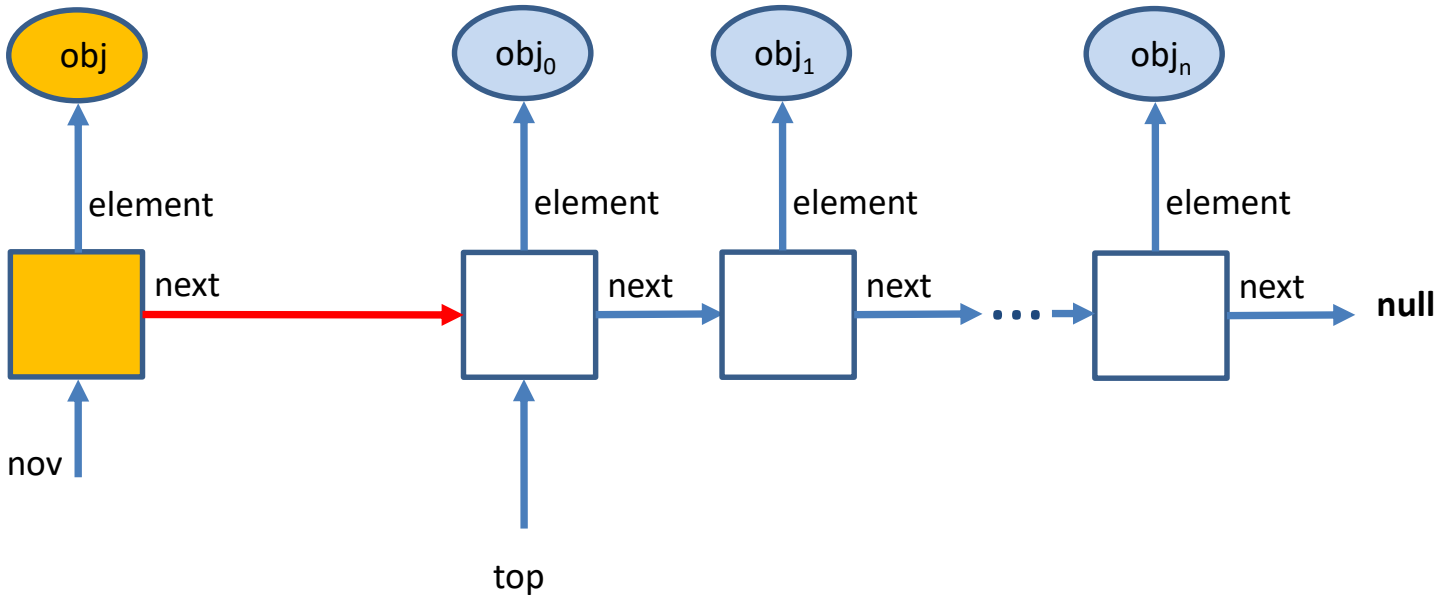
NALOGE

`void push(Object obj)` – doda element na vrh sklada



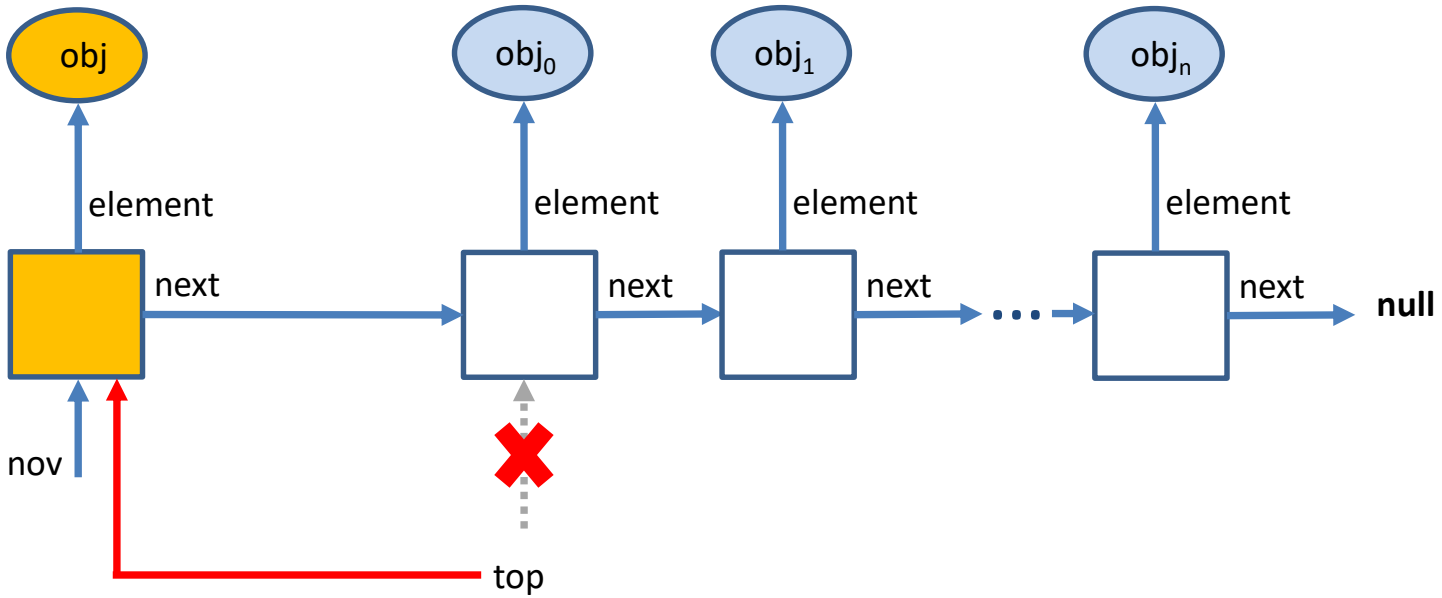
NALOGE

`void push(Object obj)` – doda element na vrh sklada



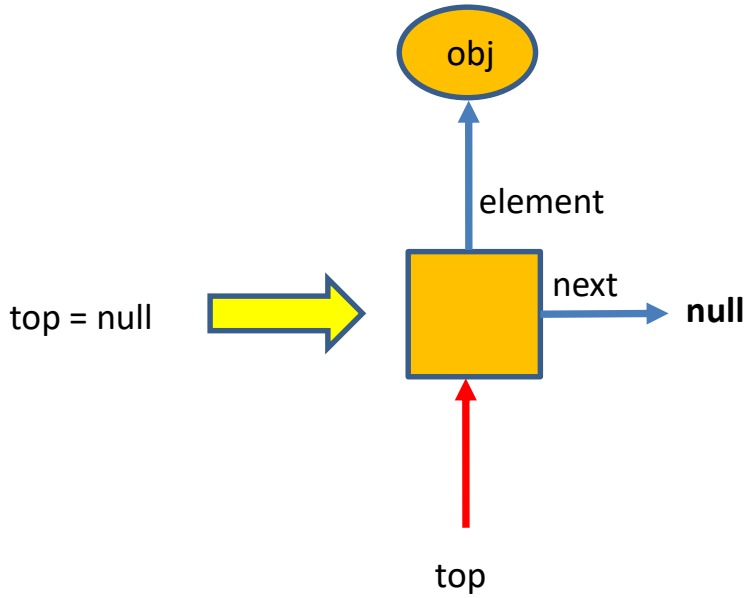
NALOGE

`void push(Object obj)` – doda element na vrh sklada



NALOGE

`void push(Object obj)` – doda element na vrh sklada



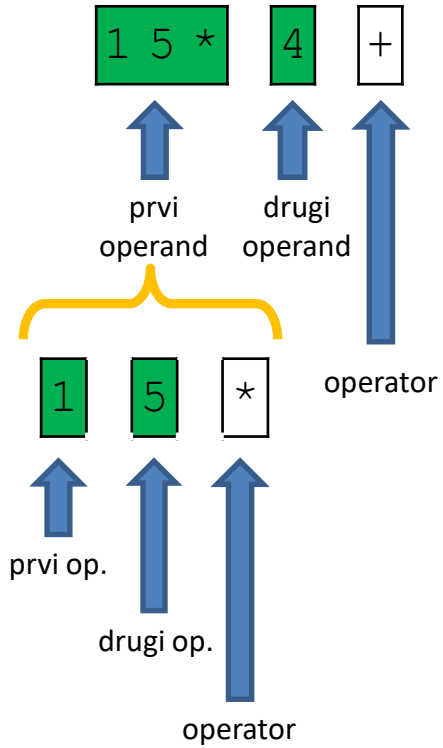
APLIKACIJA: IZRAČUN VREDNOSTI IZRAZA V POSTFIKSNI OBLIKI

Implementirajte funkcijo, ki prejme izraz v postfiksni obliki in izpiše njegovo vrednost.

Za postfiksno notacijo velja, da se operatorji pišejo za operandoma.
Operandi so lahko konstante ali so tudi sami izrazi v postfiksni obliki.

Primer izraza v postfiksni obliki:

1	5	*	4	+



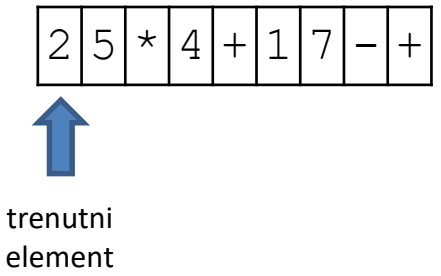
APLIKACIJA: IZRAČUN VREDNOSTI IZRAZA V POSTFIKSNI OBLIKI

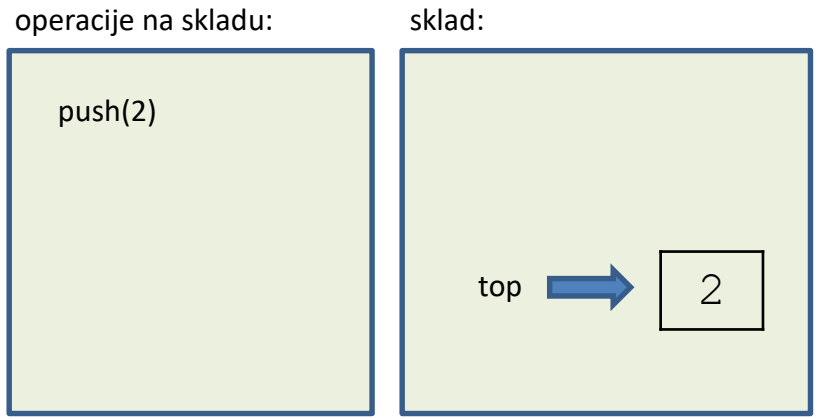
Ideja:

- pregledujemo elemente izraza z leve proti desni:
    - če je trenutni element operand, si ga zapomnimo,
    - če je trenutni element operator, ga apliciramo na nazadnje zapomnjenima operandoma in si rezultat zapomnimo (z rezultatom nadomestimo operanda).
- ko pregledamo vse elemente izraza, je nazadnje zapomnjena vrednost naš končni rezultat.

APLIKACIJA: IZRAČUN VREDNOSTI IZRAZA V POSTFIKSNI OBLIKI

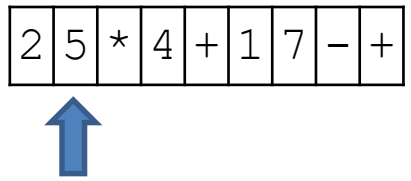
Primer:

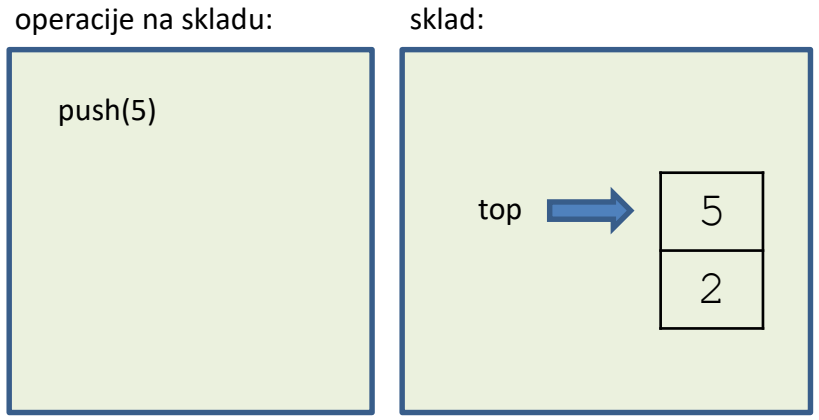




APLIKACIJA: IZRAČUN VREDNOSTI IZRAZA V POSTFIKSNI OBLIKI

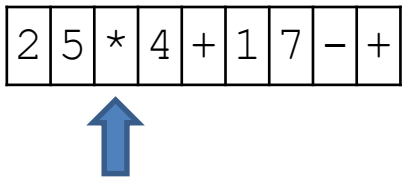
Primer:

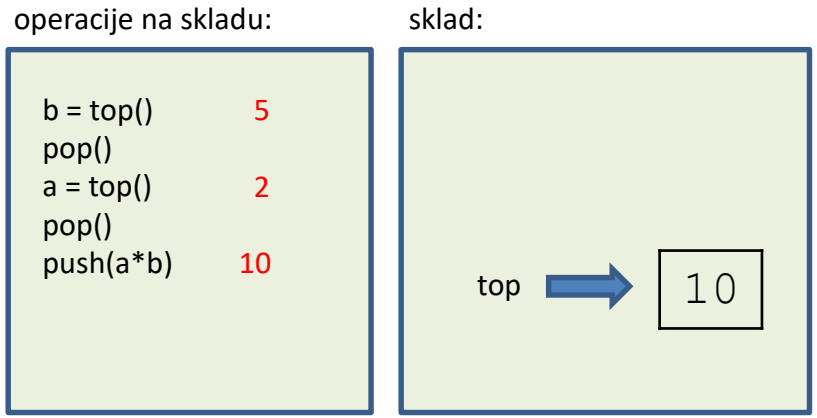




APLIKACIJA: IZRAČUN VREDNOSTI IZRAZA V POSTFIKSNI OBLIKI

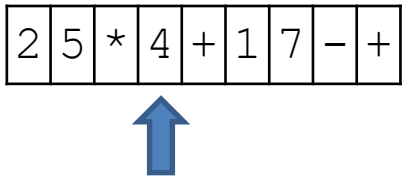
Primer:

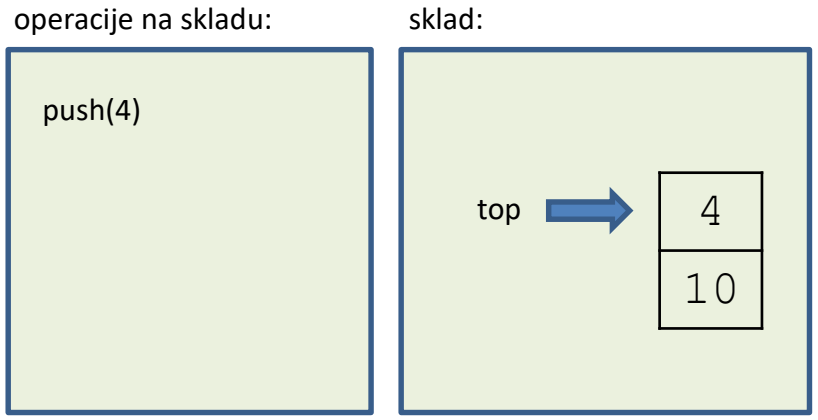




APLIKACIJA: IZRAČUN VREDNOSTI IZRAZA V POSTFIKSNI OBLIKI

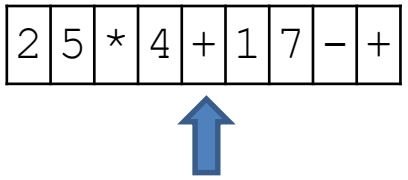
Primer:





APLIKACIJA: IZRAČUN VREDNOSTI IZRAZA V POSTFIKSNI OBLIKI

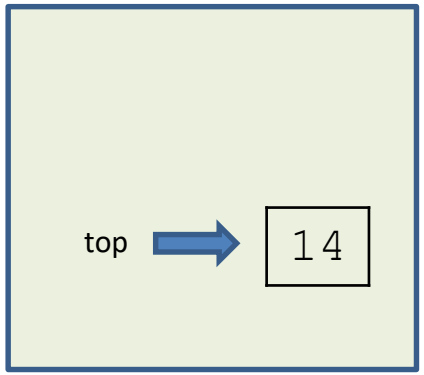
Primer:



operacije na skladu:

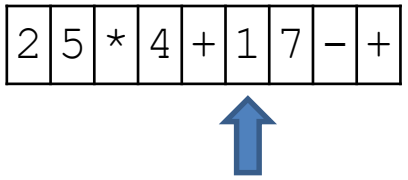
b = top() 4
pop()
a = top() 10
pop()
push(a+b) 14

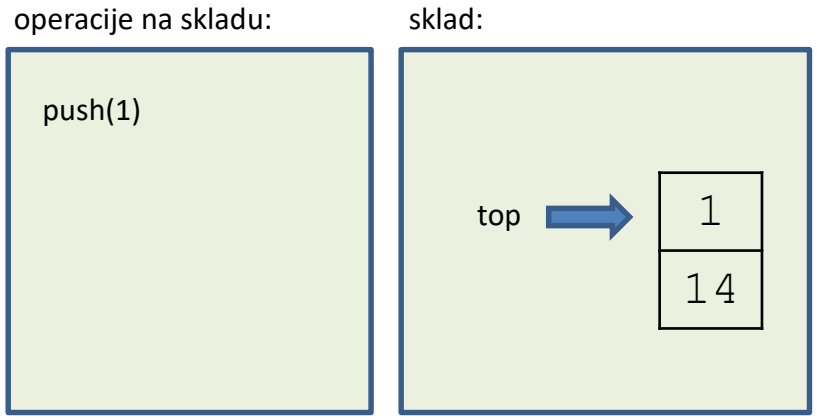
sklad:



APLIKACIJA: IZRAČUN VREDNOSTI IZRAZA V POSTFIKSNI OBLIKI

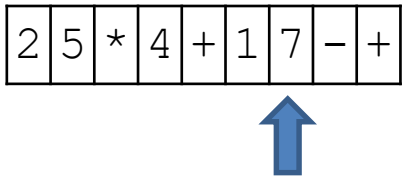
Primer:

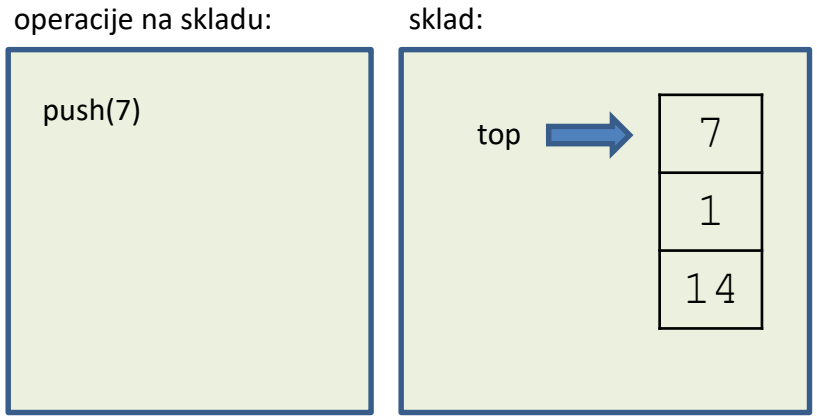




APLIKACIJA: IZRAČUN VREDNOSTI IZRAZA V POSTFIKSNI OBLIKI

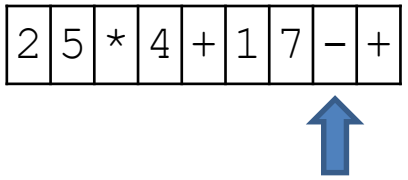
Primer:





APLIKACIJA: IZRAČUN VREDNOSTI IZRAZA V POSTFIKSNI OBLIKI

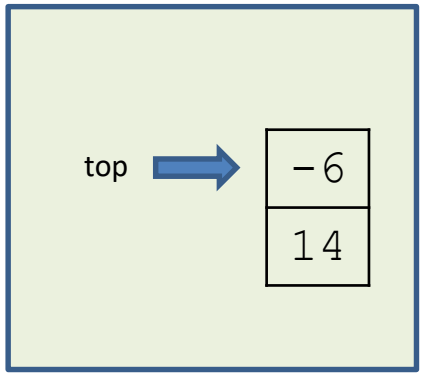
Primer:



operacije na skladu:

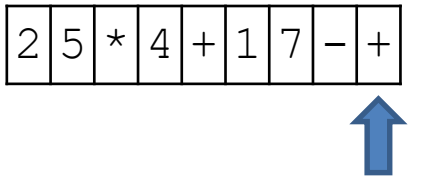
b = top()	7
pop()	
a = top()	1
pop()	
push(a-b)	-6

sklad:



APLIKACIJA: IZRAČUN VREDNOSTI IZRAZA V POSTFIKSNI OBLIKI

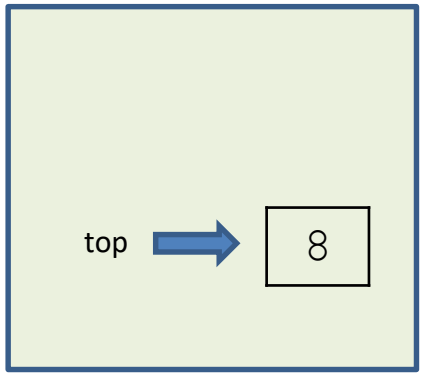
Primer:



operacije na skladu:

```
b = top()        -6
pop()
a = top()        14
pop()
push(a+b)        8
```

sklad:



APLIKACIJA: IZRAČUN VREDNOSTI IZRAZA V POSTFIKSNI OBLIKI

```
String[] izraz = {"2","3","2","*","1","+","+","4","-"};

for (int i = 0; i < izraz.length; i++) {

  String token = izraz[i];
  Double arg1;
  Double arg2;
  ...
}
```

```
// branje in odstranjevanje argumentov s sklada
arg2 = (Double)stack.top(); stack.pop();
arg1 = (Double)stack.top(); stack.pop();

// dodajanje rezultata na sklad
stack.push(arg1 + arg2);

// dodajanje argumenta na sklad
stack.push(Double.parseDouble(token));
```