Stekovi

Katedra za informatiku, Fakultet tehničkih nauka, Univerzitet u Novom Sadu

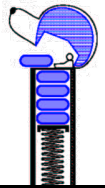
2021.

Apstraktni tipovi podataka

- ATP je apstrakcija strukture podataka
- ATP definiše
  - podatke koji se čuvaju
  - operacije nad podacima
  - uslovi kada dolazi do greške
- primer: ADT koji modeluje sistem za trgovinu akcijama
  - podaci: kupi/prodaj narudžbe
  - operacije:
    - order buy(stock, shares, price)
    - order sell(stock, shares, price)
    - void cancel(order)
  - greške:
    - nepostojeće akcije
    - otkazivanje narudžbe

Stek ATP

- stek (stack) ATP čuva proizvoljne objekte
- ubacivanje i uklanjanje poštuje LIFO (last-in-first-out) princip
- primer: PEZ bombone :)
- glavne operacije:
  - push(object): dodaje element na vrh steka
  - object pop(): skida element sa vrha steka
- dodatne operacije:
  - object top(): vraća najviši element bez skidanja
  - integer len(): vraća broj elemenata na steku
  - boolean is_empty(): vraća True ako je stek prazan



Primer operacija nad stekom

operacija	rezultat	sadržaj steka
`S.push(5)`	–	[5]
`S.push(3)`	–	[5,3]
`len(S)`	2	[5,3]
`S.pop()`	3	[5]
`S.is_empty()`	False	[5]
`S.pop()`	5	[]
`S.is_empty()`	True	[]
`S.pop()`	greška	[]
`S.push(7)`	–	[7]
`S.push(9)`	–	[7,9]
`S.top()`	9	[7,9]
`S.push(4)`	–	[7,9,4]
`len(S)`	3	[7,9,4]
`S.pop()`	4	[7,9]
`S.push(6)`	–	[7,9,6]
`S.push(8)`	–	[7,9,6,8]
`S.pop()`	8	[7,9,6]

Primene steka

- neposredne primene
  - istorija poseta stranicama u web čitaču
  - undo sekvenca u tekst editoru
  - lanac rekurzivnih poziva u programu
- indirektne primene
  - pomoćna struktura podataka za mnoge algoritme
  - komponenta u okviru drugih struktura podataka

Stek pomoću niza

- implementiraćemo stek pomoću niza
- dodajemo elemente s leva u desno
- posebna promenljiva čuva indeks poslednjeg elementa

Stek pomoću niza 2

- ako se niz popuni...
- nova operacija push mora da proširi niz i iskopira sve elemente

Performanse steka

- neka je n broj elemenata na steku
- prostor u memoriji je $O(n)$
- svaka operacija je $O(1)$ (amortizovano za push)

Implementacija steka u Pythonu

```
class ArrayStack:
  def __init__(self):
    self._data = []

  def __len__(self):
    return len(self._data)

  def is_empty(self):
    return len(self._data) == 0

  def push(self, e):
    self._data.append(e)

  def top(self):
    if self.is_empty():
      raise Empty('stack is empty')
    return self._data[-1]

  def pop(self):
    if self.is_empty():
      raise Empty('stack is empty')
    return self._data.pop()
```

Primer: uparivanje zagrada

- svako (, {, [mora biti upareno sa), },]
- ispravno: ( )(( )){([( )])}
- ispravno: ((( )(( )){([( )])}
- neispravno:)(( )){([( )])}
- neispravno: ({[ ])}
- neispravno: (

Primer: uparivanje zagrada

ParenMatch(X, n)

Input: niz X od n tokena, koji mogu biti zagrada, promenljiva, aritmetički operator ili broj
Output: True akko su zagrade ispravno napisane

```
Neka je S prazan stek
for i ← 0 to n − 1 do
    if X[i] je otvorena zagrada then
        S.push(X[i])
    else if X[i] je zatvorena zagrada then
        if S je prazan then
            return False {nema odgovarajuće otvorene zagrade}
        if S.pop() ne odgovara vrsti zagrade od X[i] then
            return False {pogrešna vrsta zagrade}
if S je prazan then
    return True {svi simboli su u korektnom redosledu}
else
    return False {neki simboli nemaju svoj par}
```

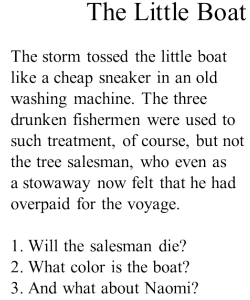
Primer: uparivanje zagrada

```
def is_matched(expr):
  """Vraća True ako se sve zagrade podudaraju."""
  lefty = '({['       # otvorene zagrade
  righty = ')}]'      # zatvorene zagrade
  S = ArrayStack()
  for c in expr:
    if c in lefty:
      S.push(c)             # stavi zagradu na stek
    elif c in righty:
      if S.is_empty():
        return False        # nema sa čime da se upari
      if righty.index(c) != lefty.index(S.pop())
        return False        # pogrešna zagrada
  return S.is_empty()       # da li su svi simboli upareni?
```

Primer: uparivanje HTML tagova

- svako <x> mora biti upareno sa </x>
- i pravilno ugnježđeno:
  - <x><y>...</y></x> je OK
  - <x><y>...</x></y> nije OK

```
<body>
<center>
<h1> The Little Boat </h1>
</center>
<p> The storm tossed the little
boat like a cheap sneaker in an
old washing machine. The three
drunken fishermen were used to
such treatment, of course, but
not the tree salesman, who even as
a stowaway now felt that he
had overpaid for the voyage. </p>
<ol>
<li> Will the salesman die? </li>
<li> What color is the boat? </li>
<li> And what about Naomi? </li>
</ol>
</body>
```



Primer: uparivanje HTML tagova

```
def is_matched_html(raw):
  """Vraća True ako se HTML tagovi podudaraju."""
  S = ArrayStack()
  j = raw.find('<')              # nađi prvi '<'
  while j != -1:
    k = raw.find('>', j+1)       # nađi sledeći '>'
    if k == -1:
      return False               # neispravan tag
    tag = raw[j+1:k]             # odbaci < >
    if not tag.startswith('/'): # ovo je otvarajući tag
      S.push(tag)
    else:                        # ovo je zatvarajući tag
      if S.is_empty():
        return False             # nema svog para
      if tag[1:] != S.pop():
        return False             # nije odgovarajući par
    j = raw.find('<', k+1)       # nađi sledeći '<'
  return S.is_empty()            # svi otvarajući tagovi upareni?
```

Primer: izračunavanje aritmetičkih izraza

- $14 - 3 * 2 + 7 = (14 - (3 * 2)) + 7$
- prioritet operatora
  - * je starije od +/-
- asocijativnost
  - operatori istog prioriteta izračunavaju se s leva u desno
- ideja:
  - stavi svaki operator na stek, ali prvo skini sa steka i izračunaj sve operacije višeg ili jednakog prioriteta

Primer: izračunavanje aritmetičkih izraza

- biće dva steka:
  - ops čuva operatore
  - $vals$ čuva vrednosti
  - koristićemo $ kao oznaku „kraja ulaza“ sa najmanjim prioritetom

Primer: izračunavanje aritmetičkih izraza

doOp()
  $x \leftarrow vals$.pop()
  $y \leftarrow vals$.pop()
  $op \leftarrow ops$.pop()
  $vals$.push(y op x)

repeatOps(refOp)
  while $vals$.size() $> 1 \wedge$ prec($refOp$) $\leq$ prec(ops.top()) **do**
    doOp()

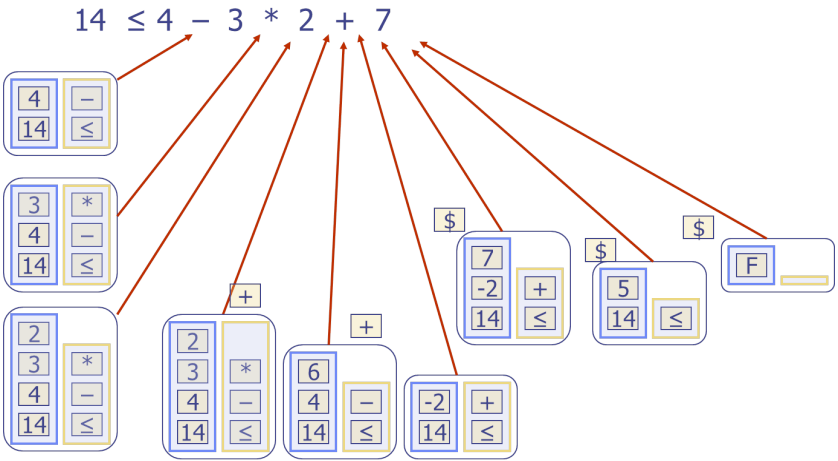
Primer: izračunavanje aritmetičkih izraza

EvalExpr()

Input: lista tokena koja predstavlja aritmetički izraz
Output: vrednost izraza

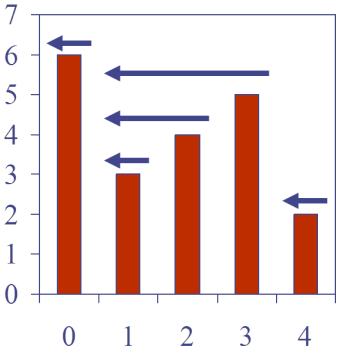
```
while ima novi token z do
    if z je broj then
        vals.push(z)
    else
        repeatOps(z)
        ops.push(z)
repeatOps($)
return vals.top()
```

Primer: izračunavanje aritmetičkih izraza



Primer: izračunavanje raspona

- za dati niz X, raspon $S[i]$ od $X[i]$ najveći broj susednih elemenata $X[j]$ koji neposredno prethode $X[i]$ takvi da je $X[j] \leq X[i]$
- primene u finansijskoj analizi



X	6	3	4	5	2
S	1	1	2	3	1

Kvadratni algoritam

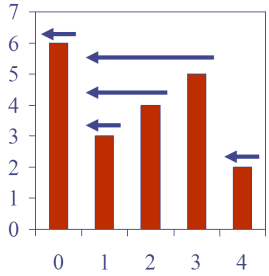
spans1(x, n)

Input: niz X od n celih brojeva
Output: niz S sa rasponima od X

```
S ← novi niz od n celih brojeva
for i ← 0 to n − 1 do
    s ← 1
    while s ≤ i ∧ X[i − s] ≤ X[i] do
        s ← s + 1
    S[i] ← s
return S
```

Izračunavanje raspona pomoću steka

- na steku čuvamo indekse elemenata koji su vidljivi kada „gledamo unazad“
- skeniramo niz s leva u desno
- neka je i tekući indeks
- skidamo indekse sa steka sve dok ne nađemo indeks j takav da je $X[i] < X[j]$
- računamo $S[i] \leftarrow i - j$
- stavimo i na stek



X	6	3	4	5	2
S	1	1	2	3	1

Linearni algoritam

spans2(x, n)

Input: niz X od n celih brojeva
Output: niz S sa rasponima od X

```
S ← novi niz od n celih brojeva
A ← novi prazan stek
S[0] ← 1
for i ← 1 to n − 1 do
    while ¬ A.is_empty() ∧ X[A.top()]≤ X[i] do
        A.pop()
    if A.is_empty() then
        S[i] ← i + 1
    else
        S[i] ← i − A.top()
    A.push(i)
return S
```

Linearni algoritam

- svaki indeks u nizu...
- ...je stavljen na stek tačno jednom
- ...je skinut sa steka najviše jednom
- naredbe u while petlji su izvršene najviše n puta
- algoritam radi u $O(n)$ vremenu