Red sa prioritetom

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

2022.

Red sa prioritetom

- red sa prioritetom čuva kolekciju elemenata
- svaki element je par (ključ, vrednost)
- osnovne operacije:
  - add(k, x): dodaje element sa ključem k i vrednošću x
  - remove_min(): uklanja element sa najmanjim ključem
- dodatne operacije:
  - min(): vraća, ali ne uklanja, element sa najmanjim ključem
  - len(), is_empty()

Primer operacija nad redom sa prioritetom

operacija	rezultat	sadržaj reda
`P.add(5, A)`	–	[(5,A)]
`P.add(9, C)`	–	[(5,A), (9,C)]
`P.add(3, B)`	–	[(3,B), (5,A), (9,C)]
`P.add(7, D)`	–	[(3,B), (5,A), (7,D), (9,C)]
`P.min()`	(3,B)	[(3,B), (5,A), (7,D), (9,C)]
`P.remove_min()`	(3,B)	[(5,A), (7,D), (9,C)]
`P.remove_min()`	(5,A)	[(7,D), (9,C)]
`len(P)`	2	[(7,D), (9,C)]
`P.remove_min()`	(7,D)	[(9,C)]
`P.remove_min()`	(9,C)	[ ]
`P.is_empty()`	True	[ ]
`P.remove_min()`	greška	[ ]

Ključevi i relacija poretka

- ključevi mogu biti bilo kog tipa za koga je definisana relacija poretka
- elementi u redu mogu imati jednake ključeve – u tom slučaju se primenjuje FIFO princip
- relacija poretka
  - refleksivna: $x \leq x$
  - antisimetrična: $x \leq y \wedge y \leq x \Rightarrow x = y$
  - tranzitivna: $x \leq y \wedge y \leq z \Rightarrow x \leq z$

Element RSP

```
class PriorityQueueItem:
  def __init__(self, k, v):
    self.key = k
    self.value = v

  def __lt__(self, other):
    return self.key < other.key

  def __le__(self, other):
    return self.key <= other.key
```

Implementacija RSP

- implementacija sa **nesortiranom** listom
- add je $O(1)$ jer dodavanje možemo raditi na bilo kom kraju liste
- remove_min i min su $O(n)$ jer moramo tražiti najmanji ključ u listi

- implementacija sa **sortiranom** listom
- add je $O(n)$ jer moramo da nađemo pravo mesto za ubacivanje novog elementa
- remove_min i min su $O(1)$ jer je najmanji ključ uvek na početku

RSP implementacija

```
class PriorityQueueBase:
  """
  Baza prioritetnog reda

  Klasa sadrži operacije nezavisne od organizacije strukture podataka
  koja se koristi za smeštanje elemenata prioritetnog reda.
  """

  def __init__(self):
    self._data = []

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

  def __str__(self):
    return ', '.join('(%s, %s)' % (e._key, e._value) for e in self._data)

  def is_empty(self):
    """
    Metoda proverava da li je red prazan.
    """
    return len(self) == 0
```

RSP sa nesortiranom listom$_1$

```
class UnsortedPriorityQueue(PriorityQueueBase):
  """
  Klasa modeluje prioritetni red korišćenjem nesortirane liste.
  """

  def __init__(self):
    super(UnsortedPriorityQueue, self).__init__()

  def _find_min(self):
    """
    Metoda pronalazi element sa najmanjim ključem.
    """
    if self.is_empty():
      raise PQError('Red je prazan.')

    min_item = self._data[0]
    size = len(self)
    for i in range(1, size):
      current_item = self._data[i]
      if current_item < min_item:
        min_item = current_item

    return min_item
```

RSP sa nesortiranom listom$_2$

```
def min(self):
    """
    Metoda omogućava pristup elementu sa najmanjim ključem.
    """
    min_item = self._find_min()
    return (min_item._key, min_item._value)

def remove_min(self):
    """
    Metoda uklanja element sa najmanjim ključem.
    """
    min_item = self._find_min()
    index = self._data.index(min_item)
    removed = self._data.pop(index)
    return (removed._key, removed._value)

def add(self, key, value):
    """
    Metoda dodaje novi element u red.
    """
    new_item = PriorityQueueItem(key, value)
    self._data.append(new_item)
```

RSP sa sortiranom listom$_1$

```
class SortedPriorityQueue(PriorityQueueBase):
  """
  Klasa modeluje prioritetni red korišćenjem sortirane liste.
  """

  def __init__(self):
    super(SortedPriorityQueue, self).__init__()

  def min(self):
    """
    Metoda omogućava pristup elementu sa najmanjim ključem.
    """
    if self.is_empty():
      raise PQError('Red je prazan.')

    min_item = self._data[0]
    return (min_item._key, min_item._value)

  def remove_min(self):
    """
    Metoda uklanja element sa najmanjim ključem.
    """
    if self.is_empty():
      raise PQError('Red je prazan.')

    removed = self._data.pop(0)
    return (removed._key, removed._value)
```

RSP sa sortiranom listom$_2$

```
def add(self, key, value):
  """
  Metoda dodaje novi element u red.
  """
  new_item = PriorityQueueItem(key, value)

  last = len(self)-1
  position = 0

  # pronalaženje pozicije za dodavanje elementa
  for i in range(last, -1, -1):
    current_item = self._data[i]
    if not new_item < current_item:
      position = i+1
      break

  self._data.insert(position, new_item)
```

Red sa prioritetom i sortiranje

- možemo upotrebiti red sa prioritetom za sortiranje niza elemenata
  - dodamo elemente jedan po jedan putem add operacije
  - uklonimo elemente jedan po jedan putem remove_min operacije
- vreme izvršavanja zavisi od načina implementacije

PQ_sort(S, C)

Input: sekvenca S, komparator C
Output: rastuće sortirana S u skladu sa C

```
P ← RSP sa komparatorom C
while ¬S.is_empty() do
    e ← S.remove_first()
    P.add(e, ∅)
while ¬P.is_empty() do
    e ← P.remove_min().key()
    S.add_last(e)
```

Implementacija u Pythonu

```
from pqueue import UnsortedPriorityQueue

def pq_sort(A):
  """
  Funkcija sortira listu koristeći dodatni prioritetni red

  Argument:
  - `A`: lista koja se sortira
  """
  size = len(A)
  pq = UnsortedPriorityQueue()

  # svi elementi liste prebacuju se u prioritetni red
  for i in range(size):
    element = A.pop()
    pq.add(element, element)

  # vraćanje elementa iz prioritetnog reda u listu
  for i in range(size):
    (k, v) = pq.remove_min()
    A.append(v)
```