Structuri de date – implementări Python

În această secțiune vom aminti pe scurt cele mai importante structuri de date și ce tipuri de date se pot folosi în Python pentru a implementa aceste structuri.

Stivă

O stivă în Python se poate implementa folosind clasa **`list`**. Pentru o stivă operațiile de inserare și eliminare se fac la același capăt, iar operațiile de adăugare la final și eliminare a ultimului element dintr-o listă în Python au complexitate O(1) (v. sectiunea Liste – Implementare internă. Complexități)

Exemplu: Crearea unei stive din numere date de la tastatură, urmată de două extrageri

```
ls_nr = (int(x) for x in input().split()) #generator
stiva = []
for x in ls_nr:
    stiva.append(x)
for i in range(2):
    if len(stiva)>0:
        print(stiva.pop())#extrage ultimul element adaugat
```

Coadă

- **`list`** – NU este eficient, extragerea de la un capăt al cozii cu pop(0) nu este O(1)

  Nu este recomandat să folosim clasa list pentru a implementa o coadă, deoarece operațiile de adăugare la începutul listei sau eliminare a primului element dintr-o listă au complexitate liniară în numărul de elemente (ca la un vector), iar la o coadă operațiile de adăugare și eliminare se fac la capete opuse.

Există două clase care se pot folosi în Python pentru a implementa o coadă: clasa **`deque`** din modulul **`collections`** și clasa **`Queue`** din modulul **`queue`** (aceasta are operațiile sincronizate, de aceea sunt mai lente). Operațiile de inserare și eliminare (la un capăt) pentru aceste tipuri au complexitate O(1)**(https://wiki.python.org/moin/TimeComplexity)**

- **`collections.deque`**
  https://docs.python.org/3.9/library/collections.html?highlight=deque#collections.deque

  În clasă există metode de adăugare și eliminare la ambele capete (coadă dublă).

  Exemplu: Crearea unei cozi din numere date de la tastatură, urmată de două extrageri:

```
import collections
ls_nr = (int(x) for x in input().split())
coada = collections.deque()
for x in ls_nr:
    coada.append(x)
for i in range(2):
    if len(coada) > 0:
        print(coada.popleft()) #extrage primul element adaugat
```

- **`queue.Queue`** – sincronizata, mai lenta
  https://docs.python.org/3/library/queue.html

  Exemplu: Crearea unei cozi din numere date de la tastatură, urmată de două extrageri

```
import queue
ls_nr = (int(x) for x in input().split())
coada = queue.Queue()
for x in ls_nr:
    coada.put(x)
for i in range(2):
    if coada.qsize() > 0:
        print(coada.get())#extrage primul element adaugat
```

Coadă de priorități

O coadă de priorități este o structură de date în care operația fundamentală este extragerea elementului cu prioritatea minimă (sau maximă, există ambele variante de cozi de priorități – de tip minim sau maxim). Pentru o mai bună înțelegere a modului de funcționare a unei astfel de structuri vom prezenta în secțiunea următoare câteva detalii legate de structura de date numită **`heap`** (ansamblu).

În Python o coadă de priorități (de tip minim) se poate implementa folosind modulul **`heapq`** sau clasa **`PriorityQueue`** din modulul **`queue`** (sincronizata, mai lentă, se bazează tot pe heapq). Operațiile de adăugare și extragere au complexitate O(log(n))

- **Modulul `heapq`** – pentru heap (binar)

https://docs.python.org/3/library/heapq.html

Exemplu: Crearea unei cozi de priorități de tip minim (min-heap) din numere date de la tastatură, urmată de două extrageri

```
import heapq
ls_nr = (int(x) for x in input().split())
h = [] #un heap este un vector
for x in ls_nr:
    heapq.heappush(h,x)
for i in range(2):
    if len(h) > 0:
        print(heapq.heappop(h)) #extrage elementul minim
```

- **Clasa queue.PriorityQueue**

https://docs.python.org/3/library/queue.html#

Clasa **PriorityQueue** este din modulul queue, ca și clasa Queue, având metode similare. Metodele clasei sunt sincronizate, deci mai lente (bazate tot pe metodele din modulul heapq)

Exemplu: Crearea unei cozi de priorități de tip minim din numere date de la tastatură, urmată de două extrageri. Metodele sunt similare celor din clasa **`Queue`** din același modul, față de exemplul de la **`queue.Queue`** se schimbă doar inițializarea

```
import queue
ls_nr = (int(x) for x in input().split())
coada = queue.PriorityQueue()
for x in ls_nr:
    coada.put(x)
for i in range(2):
    if coada.qsize() > 0:
        print(coada.get())#extrage elementul minim
```