Gomile i redovi

Gomile i redovi (stacks and queues) bili su među najranijim definisanim strukturama podataka. Iako je opšti slučaj da se ove dve strukture definišu pomoću klasa, kako je objektno orijentisano programiranje tema narednog kursa, u ovoj nastavnoj jedinici ćemo gomile i redove implementirati pomoću Python lista.

Gomile (stacks)

Gomile ili stekovi su strukture podataka koje funkcionišu po modelu sličnom organizovanju predmeta u realnom redu – slaganju predmeta jedan na drugi. Kada dodajemo jedan predmet na drugi, dodajemo ga na vrh prethodnog. Ako skinemo taj predmet sa prethodnog, skidamo ga opet – sa vrha, dakle ne odozdo. Kako bismo došli do poslednjeg, moramo prvo skloniti sve prethodne predmete. Ova metoda se u računarstvu zove LIFO metoda, što je akronim od last in – first out (poslednji ulazi, prvi izlazi).

Dakle, gomile predstavljaju strukturu podatka gde se objekti ubacuju ili izbacuju po redosledu kojim su ušli. Korisnik može bilo kad ubaciti element, ali može „dohvatiti" ili izbaciti samo element koji je poslednji ubačen.

Primer sa kojim se često srećemo, a gde je implementiran stek je istorija pretraživača. Kad god korisnik (u istom tabu) otvori novu adresu, ta adresa je ubačena na istorijski stek i kad god se korisnik putem `Back` komande vrati na prethodnu stranu, pretraživač izbacuje trenutnu stranicu iz istorijske navigacije i otvara prethodnu.

Takođe se i `Undo` funkcija zasniva na ovom principu: prva preduzeta akcija je na početku steka, a poslednja (i najskorija) na kraju. Pa kada korisnik izvrši `Undo`, program će poništiti najnoviju promenu koja se nalazi na kraju steka.

Da bi se realizovao stek, potrebno je imati implementaciju barem četiri osnovne metode i to: `push`, `peek`, `pop` i `size`:

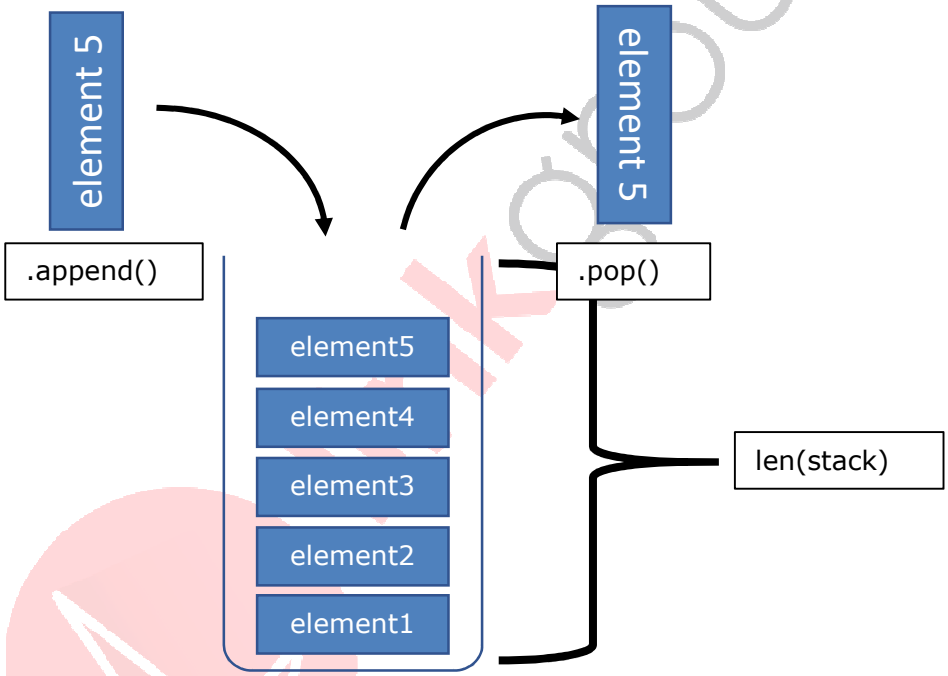
- Metodom `push` ubacujemo element na vrh steka (gomile).
- Metodom `peek` dobavljamo u program poslednji (najnoviji) element steka (gomile).
- Metodom `pop` izbacujemo poslednji (najnoviji) element steka (gomile).
- Metodom `size` dobavljamo trenutnu veličinu steka (gomile).

Ove metode su univerzalne za ovaj tip strukture podataka bez obzira na to o kom programskom jeziku je reč. U Pythonu, ove metode se mogu mapirati na postojeću funkcionalnost lista. Recimo da imamo gomilu: `stack = ['element1', 'element2', 'element3']`. Mapiranje funkcionalnosti iz Pythona na nativne metode steka bi izgledalo ovako:

Nativne metode steka (gomile)	Python implementacija	Rezultat
push	stack.append('element4') print(stack)	['element1', element2', 'element3', 'element4']
peek	print(stack[-1])	'element4'
pop	stack.pop() print(stack)	['element1', 'element2', 'element3']
size	print(len(stack))	3

Tabela 13.1. Nativne metode steka i njihove implementacije pomoću Python liste

Najbitnije metode za implementaciju gomila nad Python listama su `append()` i `pop()` i to se vidi na narednoj ilustraciji:



Slika 13.1. Ilustracija metoda append() i pop()

Metodom `append()` dodajemo element u naš stack (listu), a metodom `pop()`, bez prosleđenog indeksa, izbacujemo poslednju dodatu vrednost iz stacka.

Jedan tok implementacije i rada steka pomoću ugrađenih metoda Python liste bi mogao da izgleda ovako:

Primer implementacije i rada steka

Radno okruženje

```
stack = []
```

```
stack.append('proces1')
print("Stack after 1. append (push): {}".format(str(stack)))
print("Current size:  {}".format(len(stack)))


stack.append('proces2')
print("Stack after 2. append (push): {}".format(str(stack)))
print("Current size:  {}".format(len(stack)))


stack.append('proces3')
print("Stack after 3. append (push): {}".format(str(stack)))
print("Current size:  {}".format(len(stack)))

stack_pop_result= stack.pop()
print("Stack after popping element(pop): {}".format(str(stack)))
print("Current size:  {}".format(len(stack)))
print("(Element which we popped: {})".format(str(stack_pop_result)))
```

Objašnjenje:

Metodom append dodajemo novi element na kraj steka a metodom pop izbacujemo poslednji element. Metodom len dobijamo veličinu odn. broj elemenata unutar steka.

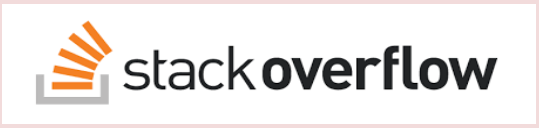
Pažljivim posmatranjem ispisa ovog programa uviđamo funkcionalnost stekova. Iako je na ovaj način implementiran stek pomoću metoda Python liste, ovo nije ispravan način. Na ovaj način smo želeli da pokažemo osnovne principe na kojima leži stek.

Python liste takođe dozvoljavaju korisniku da pristupi bilo kom elementu, bez obzira na redosled, da menja raspored elemenata unutar liste, briše elemente itd. Međutim, ništa od ovoga ne spada u osnovne principe na kojima se temelji stek kao struktura podatka. Zato smo i nagovestili, na početku ove nastavne jedinice, da je najbolji način implementacije steka zapravo definisanje klase koja će se nadograditi na klasu Python liste (najpre se misli na zabranu korišćenja metoda kao što su `insert()`, `remove()` i sličnih), čime se njena funkcionalost zapravo proširuje, ali i ograničava.

Zanimljivost

Popularni programerski sajt za traženje pomoći i postavljanje pitanja prilikom programiranja https://stackoverflow.com/, deo grupe Stack Exchange, dobio je ime po vrlo poznatoj grešci Stack overflow (prekoračenje steka). Kada u našem kodu pozovemo funkciju, sve instrukcije unutar nje i lokalne promenljive se smeštaju u deo radne memorije računara zvani stack. Kada ta funkcija završi izvršavanje, oslobađa se taj deo memorije u sistemu koji je predviđen za naš program. Stack overflow greška se dešava kada naš kod te funkcije prilikom poziva zauzme više memorije u steku nego što je to moguće. U ovom trenutku naš program puca i izlistavaju se svi trenutno aktivni pozivi iz steka, koji su na kraju pozvali funkciju zbog koje je došlo do pucanja.

Ovo listanje se zove stack trace. Najlakši način za demonstriranje ovoga je implementacija rekurzivne funkcije sa kojom se kalkuliše faktorijel nekog broja.



Slika 13.2. Logo sajta stackoverflow.com[1]

Pitanje

Koja od ove tri metode se ne računa kao metoda koja je deo koncepta implementacije steka?

- **sort()**
- append()
- pop()

Objašnjenje:

Tačan odgovor je da metoda sort() ne pripada fundamentalnim metodama na kojima se koncept stekova bazira. Sortiranjem bismo izgubili originalan redosled gomile i time izgubili svrhu tog steka.

Redovi (queues)

Red predstavlja strukturu podataka koja sadrži objekte koji su ubačeni i izbačeni FIFO (first in – first out) metodom (prvi ulazi – prvi izlazi), što znači da se novi elementi mogu konstantno ubacivati, ali se elementi izbacuju po vremenu koje provode u redu (od onog koji najduže stoji do onog koji je tu proveo najmanje vremena). Ovaj koncept se direktno translira i na primere iz pravog života, gde god imamo red u kojem moramo da čekamo – prvi će iz reda izaći onaj koji najduže čeka. Primera za redove ima svuda oko nas. U video-igrama se naredbe koje korisnik zada pamte u jednoj ovakvoj strukturi i izvršavaju upravo po FIFO metodi: od čitavog skupa zadatih naredbi, prvo se izvršavaju one koje smo prve zadali.

Opet, zbog načina na koji su liste u Pythonu implementirane, ovo nije efikasan način i treba pribeći ili implementaciji sopstvenih klasa ili korišćenju već gotovih biblioteka koje podržavaju ovu strukturu podataka (ugrađeni `collections module`). Na Python listama ćemo se samo upoznati sa konceptom redova. Razlog neefikasnosti je taj što se, kad god pozovemo .pop() metodu nad listom, interno izvršava for petlja, koja za svaki poziv .pop() metode mora pomeriti sve elemente ulevo. Ovo nije problem nad manjim listama, ali ako naše liste sadrže hiljade elemenata, vreme izvršenja se znatno uvećava.

Osnovne metode redova su enqueue, dequeue, peek i size:

[1] https://stackoverflow.com/

- metodom `enqueue` ubacujemo element na kraj reda;
- metodom `dequeue` brišemo i vraćamo nazad u program prvi (najstariji) element reda;
- metodom `peek` dobavljamo prvi element u redu (nulti indeks);
- metodom `size` dobavljamo trenutnu veličinu reda.

Generalno, i redovi i gomile počinju prazni.

Kao i sa gomilama, mapiraćemo osnovnu funkcionalnost redova sa ugrađenim odgovarajućim metodama Python lista u sledećoj tabeli na primeru: queue = ['element1','element2','element3','element4'].

Nativne metode reda	Python implementacija	Rezultat
enqueue	queue.append('element5') print(queue)	['element1', 'element2', 'element3', 'element4', 'element5']
dequeue	print(queue.pop(0))	'element1'
peek	print(queue[0])	'element2'
size	print(len(queue))	4

Tabela 13.2. Nativne metode redova i njihove implementacije pomoću Python liste

Upišite kod u radno okruženje i proverite rezultat:

Radno okruženje

Najbitniji koncept redova, enqueue-dequeue, vidi se na sledećem dijagramu:

Slika 13.3. Koncept redova enqueue-dequeue

Dakle, kada god ubacujemo element u red, koristimo naredbu `enqueue` ili, u slučaju Python liste, `.append()`. Ovim se nova vrednost uvek dodaje na kraj liste, dok se naredbom `dequeue` izbacuje prva vrednost, vrednost koja najduže stoji u listi. Bitno je napomenuti da se naredbom `dequeue`, ili u slučaju Python liste naredbom `.pop()`, sa nultim (prvim

indeksom) prvi element ne briše, već se izbacuje iz liste i vraća nazad u program. Implementacija redova uz Python liste bi mogla izgledati ovako:
Primer redova

Radno okruženje

```
queue = []

# enqueue
queue.append('DOWN')

queue.append('UP')

queue.append('A')

queue.append('B')

print("Number of commands in queue: {}".format(len(queue)))

# dequeue
command = queue.pop(0)
print("Next commands is {}".format(command))

command = queue.pop(0)
print("Next command is {}".format(command))

print("Number of commands in queue: {}".format(len(queue)))
```

Nakon analize ovog primera i ispisa, vidimo da su u redu ostale još dve komande (elementa): A i B. Prvo smo počeli sa praznim redom, u koji smo odmah potom dodali četiri komande (elementa) korisnika. Nakon tog dodavanja (`.append()` metodom), program je nastavio sa izvršavanjem zadatih komandi, i to po redu kojim su komande pristizale.

Rezime

- Gomile i redovi su strukture podataka, koncepti, koji se mogu koristiti i u drugim programskim jezicima.
- I gomile i redovi započinju kao prazne strukture podataka (bez elemenata).
- Gomile (stekovi) funkcionišu po metodi LIFO (last in – first out), gde poslednji element koji pristigne u gomilu – prvi izlazi.
- Da bi se gomila (stek) realizovala, potrebno je implementirati barem četiri metode, i to push, peek, pop i size.
- Metoda push se mapira na metodu .append() liste kojom element dodajemo na kraj Python liste.
- Metoda peek se mapira na indeksiranje liste, konkretno na `[-1]` indeks, čime dobavljamo poslednji element u listi.
- Metoda pop se mapira na .pop() metodu liste sa kojom izbacujemo poslednji element Python liste.

- Metoda size se mapira na ugrađenu funkciju len() kojom dobavljamo trenutnu veličinu naše liste (steka).
- Redovi (kjuovi, engl. queue) funkcionišu po FIFO (first in – first out) metodi, gde prvi element koji uđe u red prvi i izlazi.
- Da bi se red realizovao, potrebno je implementirati barem četiri metode, i to: enqueue, dequeue, peek i size.
- Metoda enqueue se mapira na metodu .append() liste, kojom dodajemo element na kraj Python liste.
- Metoda dequeue se mapira na .pop(0) metodu liste, pri čemu uvek prosleđujemo nulti indeks. Razlog ovome je to što uvek želimo da izbacimo prvi element u listi, koji se nalazi na nultom indeksu.
- Metoda peek se mapira na indeksiranje liste, konkretno na [0] indeks, čime dobavljamo prvi element u listi.
- Metoda size se mapira na ugrađenu funkciju len(), kojom dobavljamo trenutnu veličinu naše liste (reda).