Problem implementacije sklada z uporabo vrste, Urša Kumelj

OSNUTEK

Težave:

Največ težav sem imela, ker sem se razlage svojega problema lotila tako, da sem izhajala iz vrste in ne iz sklada. Po konzultacijah me je profesor na te stvari opozoril, zato sem jih popravila. Prav tako mi je predlagal razlago načinov reševanja s prikazom tabele časovne odvisnosti, ki sem jo tako vključila v predstavitev in opis. Predlagal je tudi razmislek, če obstaja še kakšen boljši način poleg omenjenih. Vse pripombe po konzultacijah sem vključila v predstavitev in v opis.

Povzetek *** Končno***

V prispevku je predstavljen problem implementacije sklada z uporabo vrst (LeetCode). *Radi bi predstavili razred Sklad z uporabo razreda Vrsta, to pomeni, da z uporabo metod iz razreda Vrsta konstruiramo razred Sklad, ki se bo obnašal kot poznana podatkovna struktura. To pomeni, da bo imel metode odstrani (odstrani vrhnji element v skladu), vstavi (vstavi element na vrh sklada), vrh (vrne element na vrhu sklada) in prazen (pove ali je sklad prazen ali ne). Najprej si bomo ogledali primer in na podalgi le-tega opisali, kako bi k problemu sploh pristopili. V naslednjem razdelku predstavimo načine za reševanje tega problema. Kar z njim nismo zadovoljni, poiščemo boljšega, ki temelji na predstavitvi sklada z z vrsto z obratnim vrstnim redom elementov kot v skladu. Algoritem prikažemo še na primeru. Prispevek zaključimo s prikazom ključnih delov kode rešitve in obrazložitvijo časovne zahtevnosti.*

Problem***končno***

Na spletni strani (LeetCode) je podan problem implementacije sklada z uporabo vrst. Poglejmo, za kaj gre.

Implementiraj sklad (LIFO last in first out) z uporabo le dveh vrst. Implementirani sklad naj podpira vse funkcije običajnega sklada (vstavi, vrh, odstrani, prazen).

Implementiraj razred Sklad:

1. Vstavi(x): vstavi element x na vrh sklada
2. Odstrani(): odstrani element z vrha sklada in ga vrne
3. Vrh(): vrne element na vrhu sklada
4. Prazen(): vrne True, če je sklad prazen in False sicer

Zaznamek: Uporabljati smete samo običajne operacije vrste, kar pomeni, da so veljavne le operacije vstavi (vstavi element na konec vrste), odstrani (odstrani element na začetku vrste) in prazna (preveri ali je vrsta prazna ali ne).

Domenimo se, da bomo vrsto vedno zapisovali kot ZAČETEK: 1 4 6 8 90 :KONEC, sklad pa vedno kot DNO: 2:4:56:78:9 :VRH. Recimo tudi, da je vrstni red elementov vrste v enakem vrstnem redu kot v skladu, če je sklad enak DNO: 2:4:56:78:9 :VRH in vrsta ZAČETEK: 2 4 56 78 9 :KONEC.

Za lažjo predstavo si oglejmo primer:

Denimo, da je trenutno naš sklad enak DNO: 8:5:1 :VRH. V tem trenutku je ta sklad predstavljen z vrsto ZAČETEK: 1 5 8: KONEC. Sedaj potrebujemo premisliti, kaj se zgodi, če nad skladom pokličemo metodo odstrani, vrh, prazen in vstavi. Opazimo, da je vrh sklada (število 1) ravno začetek vrste. Prav tako bi z metodo odstrani želeli odstraniti vrhnji element sklada, ki je v tem primeru 1. Malce bolj pa potrebujemo premisliti, kaj se zgodi, ko želimo vstaviti element v sklad.

Ideja rešitve

Naj bo primer za sklad tak kot zgoraj, torej DNO: 8:5:1 :VRH in se nanj sklicujmo v nadaljevanju.

Ko se lotimo naloge, imamo dva načina predstave sklada z vrsto in sicer lahko ga predstavimo kot:

1. ZAČETEK: 8 5 1 :KONEC (vrsta z enakim vrstnim redom elementov kot v skladu)
2. ZAČETEK: 1 5 8 :KONEC (vrsta z obrtnim vrstnim redom elementov kot v skladu)

Če se odločimo za 1.način, bo naša implementacija bolj zakomplicirana, ker:

1. Za vrh sklada bi potrebovali iti skozi celotno vrsto in pri tem na koncu shraniti zadnji element. V primerjavi z drugim primerom se sliši težje, saj je vrh drugega primera ravno prvi element, za katerega poznamo metodo začetek iz vrste.
2. Za odstranitev iz sklada pa bomo prav tako rabili nekako odstraniti vrhnji element, pri tem pa ne smemo ustvariti nove vrste brez vrhnjega elementa in jo potem vrniti, ker ta metoda ne vrača. Zveni precej zakomplicirano, saj pri uporabi drugega primera le odstranimo prvi element v vrsti.
3. Za vstavitev elementa na vrh sklada pa ne bomo imeli preveč dela, saj bomo podatek samo vstavili na konec vrste.

Če pa se odločimo za 2. način, bomo prav tako imeli težave, ker bi za vstavitev elementa na vrh sklada potrebovali iti skozi celotno vrsto, kar je težje kot pri prvem primeru.

Da se o tej odločitvi prepričamo, si poglejmo še tabelo časovne zahtevnosti za posamezno metodo. Podrobnejša razlaga sledi na koncu.

METODA	1.NAČIN	2.NAČIN
Vstavi(x)	O(1)	O(n)
Odstrani()	O(n)	O(1)
Vrh()	O(n)	O(1)
Prazen()	O(1)	O(1)

Način predstavitve sklada z vrsto z obratnim vrstnim redom elementov kot v skladu *** Končno ***

Da se izognemo nepotrebnim zapletom, opisanim zgoraj, se je bolj smiselno odločiti za predstavitev sklada z vrsto z obratnim vrstnim redom elementov kot v skladu (2. način), ker bomo še vseeno imeli manj težav kot pri implementaciji sklada z vrsto z enakim vrstnim redom elementov kot v skladu (1. način). Prav tako pa iz tabele vidimo, da bomo imeli samo enkrat časovno odvisnost linearno, sicer pa konstantno.

Opišimo sedaj, kako bi se v splošnem lotili reševanja za posamezno metodo:

1. prazen_sklad(): tukaj nimamo posebnosti. V primeru, da je vrsta prazna, bo prazen tudi sklad. Tako vrnemo True v primeru, da to drži in False sicer.
2. odstrani_sklad(): kot smo že opisali zgoraj v naivnem načinu, bomo odstranili prvi element v vrsti.
3. vrh_sklad(): tudi tukaj bomo vrnili ravno prvi element vrste.
4. vstavi_sklad(x): imamo dve možnosti in sicer, da uporabimo metodo stražarja ali pa pomožno vrsto. To bo nekoliko bolj podrobneje razloženo v razdelku Programska rešitev.

Uporaba na primeru***končno***

Poglejmo si, kako bi zgoraj opisane metode delovale na konkretnem primeru.

Naj bo sklad s1 enak DNO: 1:7:33:0:500 :VRH. Njena vrsta v1 pa izgleda ZAČETEK: 500 0 33 7 1 :KONEC.

Kaj dobimo po klicu posamezne metode na zgornji primer vrste v1?

1. s1.prazen_sklad(): sklad sedaj ni prazen in bomo dobili True
2. s1.vrh_sklada(): dobili bomo 500
3. s1.odstrani_sklad(): dobili bomo sklad DNO: 1:7:33:0 :VRH, njena vrsta, s katero je implementiran pa bo izgledala tako ZAČETEK: 0 33 7 1 :KONEC.
4. s1.vstavi(9): dobili bomo sklad DNO: 1:7:33:0:9 :VRH, njena vrsta, s katero je implementiran pa bo izgledala tako ZAČETEK: 9 0 33 7 1 :KONEC.

Lahko pa bo nekdo želel poklicati metodi vrh in odstrani na že praznem skladu, takrat bomo sprožili napako.

Programska rešitev***končno***

Poglejmo si morda najtežji del kode. To je metoda vstavi_sklad(x). Pri tej je ključna ideja uporaba stražarja. Povedali smo, da je potrebno na začetku v prvotno vrsto vstaviti stražarja (vrstica 90). Kaj je vrednost našega stražarja, je vseeno. Naj bo to recimo None. Nato vstavimo še podatek x (vrstica 91). Potem se sprehajamo skozi vsak element v vrsti, dokler ponovno ne pridemo do stražarja (vrstica 92). V zanki vsak element iz začetka, prestavimo na konec (vrstici 93 in 94). Ne smemo pa pozabiti elementa iz začetka tudi izbrisati (vrstica 95). Zanka se nato ustavi, saj smo prišli do stražarja. Vendar stražar je še vedno v vrsti in sicer na začetku. Tega pa ne želimo, zato ga na koncu odstranimo (vrstica 96).

```
85      def vstavi_sklad(self, x):
86          '''na konec sklada vstavi podatek x, torej na zacetek vrste vstavi element x'''
87          # ideja: ker v vrsti vstavljamo na konec, dodamo na konec strazarja, potem vstavimo podatek x,
88          # nato pa po vrsti vstavljamo podatke iz vrste na konec
89          # vrsta: 1 5 8 -> 1 5 8 None x -> 5 8 None x 1 -> ... -> None x 1 5 8, na koncu se odstranimo None
90          self.podatki.vstavi(None)
91          self.podatki.vstavi(x)
92          while self.podatki.zacetek() is not None:
93              zacetek = self.podatki.zacetek()
94              self.podatki.vstavi(zacetek)
95              self.podatki.odstrani()
96          self.podatki.odstrani()
```

Časovna zahtevnost***končno***

Pri analizi se moramo sprva dogovoriti, kaj je velikost našega problema in kaj je karakterisitična operacija. Glede na to, da imamo 4 metode, potrebujemo to določiti pri vsaki posebej. Naredimo tabelo, kjer bomo to določili.

METODA	KARAKTERISTIČNA OPERACIJA	VELIKOST PROBLEMA
Vstavi(x)	prestavljanje elementov iz začetka na konec vrste	poljubno velika vrsta
Odstrani()	odstranjevanje	poljubno velika vrsta
Vrh()	pregledovanje	poljubno velika vrsta
Prazen()	pregledovanje	poljubno velika vrsta

Tabelo časovne zahtevnosti smo naredili že pri ideji rešitve. Razložimo lahko še zakaj O(1) in O(n):

- O(1): vedno potrebujemo pregledati/odstraniti le prvi element, zato je to delo vedno konstantno
- O(n): potrebujemo iterirati skozi celotno vrsto

Na tem mestu bi se morda vprašali, ali obstaja kakšen način, da bi povsod dobili konstantno časovno zahtevnost. Recimo, da bi uporabili dve vrsti ena, ki opisuje 1. način in druga, ki opisuje 2. način. Tudi tukaj bi se nam zataknilo in ne bi prišli do boljših časovnih zahtevnosti. Morda pa obstaja kakšen način, ampak jaz zanj ne vem.

Viri

LeetCode. (brez datuma). *Implement Stack using Queues.* https://leetcode.com/problems/implement-stack-using-queues/.