IN2010
Oblig 1

4. september 2020

Innlevering

Last opp filene dine på Devilry. Innleveringsfristen er fredag 18. september 2020, kl. 23:59.

Vi anbefaler så mange som mulig om å sammarbeide i små grupper på *opp til tre*. Dere må selv opprette grupper i Devilry, og levere som en gruppe (altså, ikke last opp individuelt hvis dere jobber som en gruppe).

Filene som skal leveres er:

- Én PDF som skal hete `IN2010-oblig1.pdf`
- En java-fil som skal hete `Teque.java` som inneholder `main` for oppgave 1.
- En java-fil som skal hete `BalanceArray.java` som inneholder `main` for oppgave 3 (a).
- En java-fil som skal hete `BalanceHeap.java` som inneholder `main` for oppgave 3 (b).
- Så mange java-filer du ønsker.

Filene skal ikke zippes eller lignende.

Oppgave 1: Teque

> Oppgaven er hentet fra Kattis[1]. Vi følger samme format på input- og output, slik at oppgaven deres kan lastes opp på Kattis, men dette er *ikke* et krav. Det er heller ikke nødvendig å oppfylle tidskravet som Kattis stiller.

Deque, eller *double-ended queue*, er en datastruktur som støtter effektiv innsetting på starten og slutten av en kø-struktur. Den kan også støtte effektivt

[1] https://open.kattis.com/problems/teque

oppslag på indekser med en array-basert implementasjon. (Merk at Java ikke tilbyr en Deque med effektiv oppslag på indekser.)

Dere skal utvide idéen om deque til *teque*, eller *triple-ended queue*, som i tillegg støtter effektiv innsetting i midten. Altså skal *teque* støtte følgende operasjoner:

`push_back(`x`)` sett elementet x inn bakerst i køen.

`push_front(`x`)` sett elementet x inn fremst i køen.

`push_middle(`x`)` sett elementet x inn i midten av køen. Det nylig insatte elementet x blir nå det nye midtelementet av køen. Hvis k er størrelsen på køen før innsetting, blir x satt inn på posisjon $\lfloor (k+1)/2 \rfloor$.

`get(`i`)` printer det i-te elementet i køen.

Merk at vi bruker 0-baserte indekser.

Input

Første linje av input består av et heltall N, der $1 \leq N \leq 10^6$, som angir hvor mange operasjoner som skal gjøres på køen.

Hver av de neste N linjene består av en streng S, etterfulgt av et heltall. Hvis S er `push_back`, `push_front` eller `push_middle`, så er S etterfulgt av et heltall x, slik at $1 \leq x \leq 10^9$. Hvis S er `get`, så S etterfult av et heltall i, slik at $0 \leq i <$ (størrelsen på køen).

Merk at du ikke trenger å ta høyde for ugyldig input på noen som helst måte, og du kan trygt anta at ingen `get`-operasjoner vil be om en indeks som overstiger størrelsen på køen.

Output

For hver `get`-operasjon, print verdien som ligger på den i-te indeksen av køen.

Eksempel-input	**Eksempel-output**
9	3
`push_back` 9	5
`push_front` 3	9
`push_middle` 5	5
`get` 0	1
`get` 1	
`get` 2	
`push_middle` 1	
`get` 1	
`get` 2	

Oppgaver

(a) Skriv et Java-program som leser input fra `stdin` og printer output slik som beskrevet ovenfor. Vi stiller ingen strenge krav til kjøretid. Du kan bruke hva du vil fra Java sitt standard-bibliotek.

(b) Oppgi en verste-tilfelle kjøretidsanalyse av samtlige operasjoner (`push_back`, `push_front`, `push_middle` og `get`) ved å bruke O-notasjon. I analysen fjerner vi begrensningen på N, altså kan N være vilkårlig stor.

(c) Hvis vi vet at N er begrenset, hvordan påvirker det kompleksiteten i O-notasjon?

Oppgave 2: Binærsøk

I forelesningen ble det nevnt at datastrukturen kan påvirke kjøretiden på en algoritme. Gi et worst-case estimat av algoritmen nedenfor, som implementerer binærsøk over lenkede lister. Oppgi estimatet ved bruk av O-notasjon. Hvordan påvirker valget av datastruktur kjøretidskompleksiteten i dette tilfellet?

Algorithm 1: Binærsøk med lenkede lister

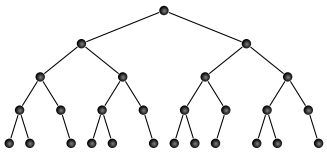
Input: En ordnet lenket liste A og et element x
Output: Hvis x er i listen A, returner **true** ellers **false**

```
1  Procedure BinarySearch(A, x)
2      low ← 0
3      high ← |A| − 1
4      while low ≤ high do
5          i ← ⌊(low+high)/2⌋
6          if A.get(i) = x then
7              return true
8          else if A.get(i) < x then
9              low ← i + 1
10         else if A.get(i) > x then
11             high ← i − 1
12     end
13     return false
```

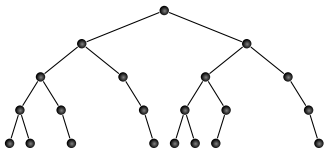
Oppgave 3: Bygge balanserte søketrær

I denne oppgaven ønsker vi å bygge et *helt balansert binært søketre*. Vi definerer dette som et binært søketre hvor det er 2^d noder med dybde d, der $0 \leq d < h$ og h er høyden på treet[2]. En annen måte å si det samme på er at den korteste

[2]Merk at dette er veldig likt definisjonen av et *komplett* binærtre, som forklares i videoen om heaps fra uke 37, men uten kravet om at noder med dybde h er plassert så langt til venstre som mulig.



(a) Et helt balansert binærtre



(b) Et ikke helt balansert binærtre

og den lengste stien fra roten til et tomt subtre (for eksempel representert med en `null`-peker) har en lengdeforskjell på 0 eller 1.

Du trenger ikke implementere et binært søketre. Alt du trenger å gjøre er å printe ut elementene du får som input i en rekkefølge som garanterer at vi får et balansert tre dersom vi legger elementene inn i binærtreet ved bruk av vanlig innsetting. Dette binære søketreet er *ikke selvbalanserende*. Input består av heltall i sortert rekkeføge, der ingen tall forekommer to ganger (altså trenger du ikke ta høyde for duplikater).

Eksempel-input	**Eksempel-output**
0	5
1	8
2	10
3	9
4	7
5	6
6	2
7	4
8	3
9	1
10	0

(a) Du har fått et *sortert array* med heltall. Print ut elementene i en rekkefølge slik at hvis de blir plassert i et binært søketre i den rekkefølgen så resulterer dette i et *balansert* søketre.

(b) Nå skal du løse det samme problemet ved bruk av *heap*. Input og output er det samme som i oppgave (a). Det første programmet ditt må gjøre er å plasere alle elementene på en `PriorityQueue`[3].

En viktig begrensning: Algoritmen din kan ikke bruke andre datastrukturer enn *heap*, men til gjengjeld kan du bruke så mange heaper du vil! De eneste operasjonene du trenger å bruke fra Java sin `PriorityQueue` (som implementerer en heap) er: `size()`, `offer()` og `poll()`. Merk at `offer()` svarer til `push()`, og `poll()` svarer til `pop()`.

[3] https://docs.oracle.com/javase/8/docs/api/java/util/PriorityQueue.html