Innlevering 1

Høst 2024

Binærsøk med lenkede lister

Gi en verste tilfelle kjøretidsanalyse av algoritmen nedenfor, som implementerer binærsøk over *lenkede lister*. Hvordan påvirker valget av datastruktur kjøretidskompleksiteten i dette tilfellet?

ALGORITHM: 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     return false
```

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 oppslag på indekser med en array-basert implementasjon.

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 elememententet 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	

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

Oppgaver

(a) Skriv *pseudokode* for hver av operasjonene

- `push_back`
- `push_front`
- `push_middle`
- `get`

Lavere kjøretidskompleksitet på operasjonene er bedre.

(b) Skriv et Java eller Python-program som leser input fra `stdin` og printer output *nøyaktig* slik som beskrevet ovenfor.

(c) Oppgi en verste-tilfelle kjøretidsanalyse av samtlige operasjoner med $\mathcal{O}$-notasjon. I analysen fjerner vi begrensningen på N, altså kan N være vilkårlig stor.

(d) Hvis vi vet at N er begrenset, hvordan påvirker det kompleksiteten i $\mathcal{O}$-notasjon? Formulert annerledes: Hvorfor er det viktig at vi fjerner begrensningen på N i forrige deloppgave? (Hint: 10^6 er en konstant).