Innlevering 3

Høst 2026

Om innleveringen

- Innleveringen skal leveres i innleverings- og tilbakemeldingssystemet Devilry innen **6. november klokken 23:59**.
- Dere skal helst *levere i grupper* på inntil tre personer, under forutsetning av at alle i gruppen både står for det som leveres og alene kan gjøre rede for alle delene av arbeidet.
- Du kan levere individuelt dersom du ønsker (du trenger ikke fylle ut et skjema eller lignende).
- Du skal bruke programmet `in2010-testrunner` som du finner her: `https://github.uio.no/IN2010/in2010-testrunner/`
- `in2010-testrunner` hjelper deg med å:
  - generere kodeskjelett,
  - kjøre koden mot testfiler, og
  - lage en zip-fil som skal leveres i Devilry.
- Med mindre noe annet er spesifisert kan dere bruke hva dere vil fra Java eller Python sine standardbibliotek, men ingen andre biblioteker.
- Kommenter **«Klar for retting»** i Devilry hvis du leverer i god tid før fristen, og innleveringen er klar for retting.

Krav til innleveringen

- Alle oppgaver skal være besvart.
- Programmene må kompilere og kunne kjøres med `in2010-testrunner`.
- Det som leveres skal være *lett forståelig, entydig og presist*.

Oppgave 1 – Effektive mengder

Den abstrakte datatypen for mengder kalles `Set`. Hvis `set` er av typen `Set`, så forventer vi at følgende operasjoner støttes:

`contains(set,` x`)`	er x med i mengden?
`insert(set,` x`)`	setter x inn i mengden (uten duplikater).
`remove(set,` x`)`	fjerner x fra mengden.
`size(set)`	gir antall elementer i mengden.

Husk at hverken rekkefølge eller antall forekomster noen rolle i mengder. Ved fjerning av et element som ikke er i mengden skal mengden forbli uforandret.

Input

Input skal leses fra `stdin`.

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 megnden.

Hver av de neste N linjene er på følgende format

`contains` x	der x er et heltall $1 \leq N \leq 10^9$
`insert` x	der x er et heltall $1 \leq N \leq 10^9$
`remove` x	der x er et heltall $1 \leq N \leq 10^9$
`size`	

Merk at du ikke trenger å ta høyde for ugyldig input på noen som helst måte.

Output

Output skal skrives til `stdout`.

For hver linje av input som er på formen:

`contains` x

skal programmet skrive ut `true` dersom x er med i mengden, og `false` ellers.

For hver linje av input som er på formen:

`size`

skal programmet skrive ut antall elementer som er i mengden.

Eksempel input/output:

Eksempel-input	Eksempel-output
9 insert 1 insert 2 insert 3 insert 1 contains 1 contains 0 remove 1 contains 1 size	true false false 2

Oppgaver

Begge deloppgavene består av å skrive et Java eller Python-program som leser input fra `stdin` og skriver ut output *nøyaktig* slik som beskrevet ovenfor. Java og Python sine innebygde implementasjoner av mengder skal ikke brukes i implementasjonene.

(a) Implementer grensesnittet for mengder som et AVL-tre.

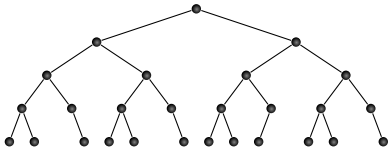
(b) Implementer grensesnittet for mengder med hashing. Du kan velge mellom *separate chaining* eller *linear probing*.

Dersom du ønsker en ekstra utfordring kan du alternativt implementere en kollisjonshåndteringsstrategi som ikke er gått gjennom på forelesning, for eksempel *cuckoo hashing* eller *quadratic probing*. Du kan finne flere eksempler her:

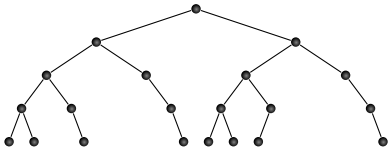
https://en.wikipedia.org/wiki/Hash_table#Collision_resolution

Oppgave 2 – 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[1]. En annen måte å si det samme på er at den korteste og den lengste stien fra roten til et tomt subtre (for eksempel representert med en `null`-peker) har en lengdeforskjell på 0 eller 1.



(a) Et helt balansert binærtre



(b) Et ikke helt balansert binærtre

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 som input. Lag en algoritme som skriver 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.

- Skriv et Java eller Python-program som implementerer algoritmen din. Det skal lese tallene fra `stdin` og skrive dem ut som beskrevet ovenfor.

(b) Nå skal du løse det samme problemet kun ved bruk av *prioritetskø*. Altså: Algoritmen din kan ikke bruke andre datatyper enn prioritetskøer, men til gjengjeld kan du bruke så mange prioritetskøer du vil!

- Skriv et Java eller Python-program som implementerer algoritmen din. Programmet må først plassere elementene som leses inn på en prioritetskø, og derretter kalle på implementasjonen av algoritmen du har kommet frem til.

  For Java kan du bruke `PriorityQueue`[2]. De eneste operasjonene du trenger å bruke fra Java sin `PriorityQueue` er: `size()`, `offer()` og `poll()`. Merk at `offer()` svarer til `push()`, og `poll()` svarer til `pop()`.

  For Python kan du bruke `heapq`[3]. De eneste operasjonene du trenger er: `heappush()` og `heappop()`, samt kalle `len()` for å få størrelsen på prioritetskøen.

[1] Merk at dette er veldig likt definisjonen av et *komplett* binærtre, som forklares i forelesningen om prioritetskøer, men uten kravet om at noder med dybde h er plassert så langt til venstre som mulig.

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

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