Eksamen i IN2010 høsten 2025

5. Desember 2025

Om eksamen

- Eksamen består av en bitteliten oppvarming, etterfulgt av to hoveddeler.
- Den første delen består av små oppgaver som rettes automatisk. Det gjøres ingen forskjell mellom ubesvart og feil svar; det betyr at det lønner seg å svare på alle oppgavene.
- Den andre delen består av større oppgaver hvor du blir bedt om å skrive og resonnere.
- Ingen hjelpemidler er tillatt.
- Hele oppgavesettet er vedlagt som PDF.
- An English translation of the full set of exercises is attached as a PDF.

Kommentarer og tips

- Det kanskje viktigste tipset er *å lese oppgaveteksten svært nøye*.
- Pass på at du svarer på nøyaktig det oppgaven ber om.
- Redegjør for eventuelle antagelser du gjør.
- Pass på at det du leverer fra deg er klart, presist og enkelt å forstå, både når det gjelder form og innhold.
- Hvis du står fast på en oppgave, bør du gå videre til en annen oppgave først.
- Alle implementasjonsoppgaver skal besvares med *pseudokode*. Det viktige er at pseudokoden er *lett forståelig, entydig og presis*.
- En *lett forståelig, entydig og presis* forklaring med naturlig språk, kan være mer poenggivende enn pseudokode som er vanskelig å forstå, tvetydig eller upresis.
- I implementasjonsoppgaver er lavere kjøretidskompleksitet mer poenggivende.
- Du kan anta at du har algoritmer og datastrukturer kjent fra pensum tilgjengelig, med mindre noe annet er spesifisert.

Oppvarming 2 poeng

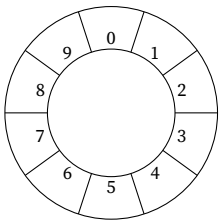
(a) Hva er en algoritme? Svar kort (maks fire setninger).

(b) Hva er en datastruktur? Svar kort (maks fire setninger).

Vi er ikke ute etter et «fasitsvar». Vi er ute etter å høre din forståelse av begrepene, og alle rimelige svar gir full uttelling.

Separate chaining 10 poeng

(a) Vi starter med et tomt array på størrelse $N = 10$.



Hashfunksjonen du skal bruke er $h(k, N) = k \mod N$, som for dette eksempelet blir $h(k, 10) = k \mod 10$. Altså hasher et tall til sitt siste siffer.

Bruk *separate chaining* til å sette inn disse nøkkel- og verdiparene i den gitte rekkefølgen:

$$(21, a), (54, b), (82, c), (10, d), (20, e), (44, f), (10, g)$$

Skriv ned resultatet etter siste innsetting, med hver linje på formen:

$$i\colon (k_1, v_1), (k_2, v_2), \ldots$$

der i er en indeks i arrayet, og $(k_1, v_1), (k_2, v_2), \ldots$ er nøkkel- og verdipar.

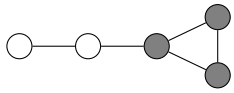
(b) Forklar kort hvordan algoritmen for innsetting ved separate chaining fungerer, og skisser algoritmen med pseudokode. Du kan anta at du har en hashfunksjon h slik at $h(k, N)$ gir et tall mellom 0 og $N - 1$ for en vilkårlig nøkkel k. Du trenger ikke ta høyde for rehashing.

```
Input: Et array A av størrelse N, en nøkkel k og verdi v
Output: Et array der (k, v) er satt inn med separate chaining
1 Procedure SeparateChainingInsert(A, k, v)
  | // ...
```

(c) Redegjør kort for kjøretidskompleksiteten til innsetting ved separate chaining.

Sykler i grafer 11 poeng

(a) Anta at grafen $G = (V, E)$ er en *urettet* graf. Grafen inneholder en sykel hvis det finnes en sti fra en node u tilbake til samme node u som går innom minst én annen node v.

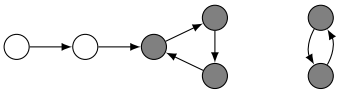


Skriv en prosedyre som avgjør om grafen inneholder en sykel.

Input: En urettet graf $G = (V, E)$
Output: Returnerer **true** hvis grafen inneholder en sykel, og **false** ellers

```
1 Procedure isCyclicUndirected(G)
  | // ...
```

(b) Anta at grafen $G = (V, E)$ er en *rettet* graf. Grafen inneholder en sykel hvis det finnes en sti fra en node u tilbake til samme node u.



Skriv en prosedyre som avgjør om grafen inneholder en sykel.

Input: En rettet graf $G = (V, E)$
Output: Returnerer **true** hvis grafen inneholder en sykel, og **false** ellers

```
1 Procedure isCyclicDirected(G)
  | // ...
```

Anti-AVL 12 poeng

AVL-trær bruker venstre- og høyrerotasjoner for å sørge for at det binære søketreet ender opp balansert etter innsetting og sletting. Her er innsetting for en variant av AVL-trær som vi kaller *Anti-AVL*.

```
Input: Rotnoden v av et Anti-AVL-tre B og et
       element x
Output: Et Anti-AVL-tre med x satt inn
1  Procedure InsertAntiAVL(v, x)
2      if v = null then
3          v ← new Node(x)
4      else if x < v.element then
5          v.left ← InsertAntiAVL(v.left, x)
6      else if x > v.element then
7          v.right ←
            InsertAntiAVL(v.right, x)
8
9      if v.left ≠ null then
10         return RightRotate(v)
11
12     return v
```

```
Input: En node z
Output: Roten til det
        høyreroterte treet
1  Procedure RightRotate(z)
2      y ← z.left
3      T2 ← y.right
4
5      y.right ← z
6      z.left ← T2
7
8      return y
```

(a) Sett inn tallene $3, 1, 4, 2, 5$ med `InsertAntiAVL`.

- Beskriv kort hvordan det resulterende Anti-AVL-treet ser ut.
- Angi det totale antallet rotasjoner som forekommer i løpet av innsettingene.

(b) Beskriv mer generelt hvordan Anti-AVL-trær ser ut.

(c) Et binært søketre er et binærtre der elementet i hver node er:

- større enn alle elementer i venstre subtre, og
- mindre enn alle elementer i høyre subtre.

Har Anti-AVL-trær den samme egenskapen? Forklar.

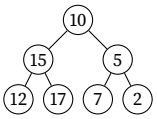
(d) Hva er kjøretidskompleksiteten på oppslag i Anti-AVL-trær? Du kan anta at oppslag er implementert identisk som ordinære binære søketrær og AVL-trær. Begrunn svaret kort.

(e) Beskriv kort hvordan innsetting i Anti-AVL-trær skiller seg fra innsetting i AVL-trær.

Level-order 12 poeng

En *level-order*-traversering besøker nodene i et tre etter *dybde*. Det vil si at roten besøkes først, så alle barna til roten, så alle barna til rotens barn, og så videre.

En *level-order*-traversering av følgende tre kan, for eksempel, besøke nodene i rekkefølgen $10, 15, 5, 12, 17, 7, 2$.



(a) Anta at treet er en *binær heap*, representert ved et array A med n elementer. Gi en algoritme som returnerer en liste med elementene fra A i *level-order*.

```
Input: En binær heap, representert ved et array A med n elementer
Output: En liste med elementene i heapen i level-order
1 Procedure LevelOrderHeap(A)
  | // ···
```

(b) Anta at treet er et *binært søketre* med n noder, der hver node v har følgende felter:

- v.`element` — verdien som er lagret i noden
- v.`left` — venstre barn av v
- v.`right` — høyre barn av v

Gi en algoritme som returnerer en liste med elementene fra treet i *level-order*.

```
Input: Roten v av et binært søketre med n noder
Output: En liste med elementene i søketreet i level-order
1 Procedure LevelOrderBST(v)
  | // ···
```

(c) Oppgi kjøretidskompleksiteten for algoritmene dine i (a) og (b).

(d) Se på følgende algoritme:

```
Input: Roten v av et binært søketre
Output: Et nytt binært søketre
1 Procedure ConstructBST(v)
2   | B ← empty binary search tree
3   | nodes ← LevelOrderBST(v)
4   | for u in nodes do
5   |   | B ← Insert(B, u)
6   | return B
```

Hva kan du si om relasjonen mellom input-treet og output-treet?

Finn k minste — 12 poeng

Gitt et array A med n sammenlignbare elementer, ønsker vi å finne de k minste elementene i A der $1 \leq k \leq n$.

Input: Et array A av n sammenlignbare elementer, og et heltall $1 \leq k \leq n$
Output: En liste med de k minste elementene fra A

```
1 Procedure FindMin(A, k)
  | // ···
```

Skriv en prosedyre som beregner `FindMin(`A`, `k`)` i:

(a) $\mathcal{O}(k \cdot n)$

(b) $\mathcal{O}(n \cdot \log(n))$

(c) $\mathcal{O}(n \cdot \log(k))$ *eller* $\mathcal{O}(n + k \cdot \log(n))$

Kollektivtilbudet i Oslo — 15 poeng

I det siste bystyremøtet ble det sluttet flere vedtak vedrørende kollektivtilbudet i Oslo. La kollektivtilbudet være representert som en sammenhengende, urettet og vektet graf $G = (V, E)$, der $u \in V$ er en holdeplass, en kant $\{u, v\} \in E$ er en *strekning* og vektfunksjonen w angir reisetid (i minutter) for en strekning.

(a) Grafen er *sammenhengende* og *urettet*. Forklar kort hva hver av egenskapene vil si for en som reiser kollektivt i Oslo.

(b) Bystyret har vedtatt at det skal bygges nye holdeplasser langs alle *strekninger* som overstiger k minutter. Under bygningsarbeidet vil det ikke være mulig å reise langs noen av disse strekningene. Skriv en algoritme som tar grafen som representerer det nåværende kollektivtilbudet, samt en grense k, og sjekker om det vil være mulig å reise mellom alle par av holdeplasser under bygningsarbeidet.

(c) Det viser seg at algoritmen fra deloppgave (b) svarer **`false`** for k-en bystyret hadde i tankene. Gi en algoritme som bystyret kan bruke til å finne den minste verdien for k som algoritmen fra deloppgave (b) svarer **`true`** for.

(d) Det siste året har det vært to hendelser der det har vært full stopp ved en holdeplass. Det ledet til at reisende ikke kunne nå destinasjonen sin. Bystyret har vedtatt en garanti: dersom det blir full stopp ved en holdeplass, skal det fremdeles være mulig for reisende å nå sin destinasjon via en alternativ rute. Garantien gjelder for alle holdeplasser.

Forklar hvordan vi kan bruke en algoritme kjent fra pensum til å sjekke om kollektivtilbudet leverer denne garantien.

(e) Møtet til bystyret varte for lenge, så de bestilte pizza, og etter det fløt det over med nye kreative idéer. Den siste i denne rekken er følgende: for å avlaste de mest trafikkerte rutene, vil de gi *negative billettpriser* på de minst trafikkerte strekningene.

Før de setter i gang med prosjektet må de forsikre seg om at det ikke er mulig å opptjene seg vilkårlig høye summer i negative billettpriser ved å reise rundt og rundt.

Billettprisen for en strekning er gitt av en vektfunksjon c. En strekning kan være svært trafikkert i én retning, men lite trafikkert i den andre, derfor kan $c(u, v)$ være ulik $c(v, u)$ for en strekning $\{u, v\} \in E$.

Forklar hvordan vi kan bruke en algoritme kjent fra pensum til å sjekke om det finnes måter å opptjene vilkårlig høye summer i negative billettpriser.