Eksamen i IN2010 høsten 2021

10. Desember 2021

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 litt større oppgaver hvor du i større grad må programmere (pseudokode), skrive og resonnere.
- Alle besvarelser skal skrives inn i Inspera og det er ingen mulighet for opplasting av håndskrevede svar.
- Ingen hjelpemidler er tillatt.

Kommentarer og tips

- Det er lurt å lese raskt gjennom eksamen før du setter i gang. Hele oppgavesettet er lagt ved som PDF.
- Det kanskje viktigste tipset er *å lese oppgaveteksten svært nøye*.
- Pass på at du svarer på nøyaktig det oppgaven spør om.
- 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.

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.

Stabilitet 3 poeng

Anta at arrayet A er usortert og inneholder personobjekter som alle har et felt for alder. Anta videre at personobjektene som ligger på A[3] og A[42] begge er 22 år gamle.

Arrayet A blir sortert etter alder. Etter sorteringen får du vite at:

- Personobjektet som lå på A[3] før sorterting, ligger nå på A[9]
- Personobjektet som lå på A[42] før sorterting, ligger nå på A[7]

(a) Var sorteringen stabil?

(b) Hva er alderen til personobjektet som ligger på A[8] etter sortering?

(c) Dersom du får vite at ingen personer i A er eldre enn 100 år gammel. Hvilken sorteringsalgoritme bør da benyttes, med hensyn til kjøretidseffektivitet?

Litt sortering 10 poeng

For hver av påstandene nedenfor kan du anta at `A` er et array med n elementer, og at i er et heltall $0 \leq i < n$.

Bubble sort

(a) Etter i iterasjoner av den ytre loopen i Bubble sort, er de i *første* elementene sortert.

(b) Etter i iterasjoner av den ytre loopen i Bubble sort, er de i *siste* elementene sortert.

(c) Bubble sort bytter kun elementer som står direkte ved siden av hverandre.

(d) Bubble sort garanterer et minimalt antall bytter.

Selection sort

(e) Etter i iterasjoner av den ytre loopen i Selection sort, er de i *første* elementene sortert.

(f) Etter i iterasjoner av den ytre loopen i Selection sort, er de i *siste* elementene sortert.

(g) Selection sort bytter kun elementer som står direkte ved siden av hverandre.

(h) Selection sort garanterer et minimalt antall bytter.

Insertion sort

(i) Etter i iterasjoner av den ytre loopen i Insertion sort, er de i *første* elementene sortert.

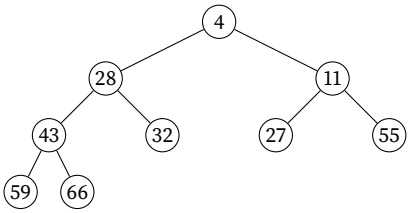
(j) Etter i iterasjoner av den ytre loopen i Insertion sort, er de i *siste* elementene sortert.

(k) Insertion sort bytter kun elementer som står direkte ved siden av hverandre.

(l) Insertion sort garanterer et minimalt antall bytter.

Binære heaps 6 poeng

Du skal ta utgangspunkt i følgende binære heap H_1, hvor både tre-representasjonen og array-representasjonen er gitt:



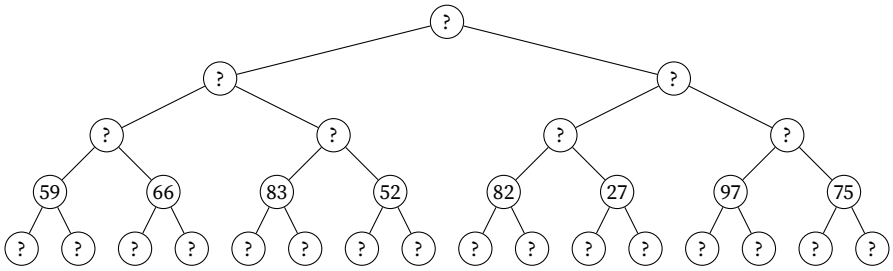
4	28	11	43	32	27	55	59	66
0	1	2	3	4	5	6	7	8

Du skal fylle inn tabellene som svarer til hver deloppgave i Inspera. Hver tabell svarer til array-representasjonen av H_1 etter de oppgitte operasjonene. Deloppgavene er uavhengig av hverandre, altså refererer H_1 til heapen som er gitt ovenfor, og endringer vi gjør i en deloppgave følger ikke med til neste deloppgave.

(a) Hvordan ser heapen ut etter ett kall på H_1.`RemoveMin`()?

(b) Hvordan ser heapen ut etter ett kall H_1.`Insert`(7)?

I en annen binær min-heap H_2 får du kun gitt verdiene på dybde 3:



(c) Hva er den minste verdien som kan ligge på dybde 4 av H_2?

Huffmantrær 5 poeng

En tekststreng vi vil komprimere består av 5 ulike tegn `a`,`b`,`c`,`d` og `e`. De relative frekvensene er gitt av følgende frekvenstabell:

a	b	c	d	e
2	1	12	1	4

(a) Hva er den lengste kodelengden for et symbol i det tilhørende huffmantreet?

(b) Hvor mange bits blir koden for tegnet `e`?

(c) Hvor mange bits brukes for å kode strengen `aaabbcdee`?

Nå skal vi ikke jobbe med noe konkret huffmantre for et en gitt frekvenstabell, men heller tenke på generelle huffmantrær.

(d) Hvis en frekvenstabell består av 8 forskjellige tegn, hvor mange noder har det tilhørende huffmantreet?

(e) I et huffmantre som har 8 løvnoder, hva er den lengste kodelengden et symbol kan ha?

Litt om beregnbarhet og kompleksitet 8 poeng

For alle spørsmålene nedenfor, svar **Sant** eller **Usant**.

(a) Det er bevist at $P = NP$.

(b) Det er bevist at $P \neq NP$.

(c) Alle NP-komplette problemer kan polynomtidsreduseres til hverandre.

(d) Alle problemer i P er også i NP.

(e) Alle avgjørelsesproblemer er enten i P eller NP.

(f) Alle problemer hvor JA-instanser kan verifiseres i polynomiell tid, er i NP.

(g) Hvis noen klarer å løse avgjørelsesproblemet **Hamiltonsykel**, **Knapsack** eller **Sudoku** i polynomiell tid, så har de bevist at $P = NP$.

(h) Alle avgjørelsesproblemer kan løses med en rask nok datamaskin.

Korteste avstander i grafer 8 poeng

For hver graftype, velg den raskeste algoritmen som finner korteste avstander fra en startnode til alle andre noder i grafen.

	Bredde-først søk	Dijkstra	Topologisk Sortering	Bellman-Ford
Ingen negative kanter				
Vektet DAG				
Uvektet				
Ingen negative sykler				

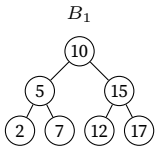
- Uvektet: en uvektet graf. Grafen er enten rettet eller urettet.
- Vektet DAG: en vektet, rettet asyklisk graf. Kantene kan ha negativ vekt.
- Ingen negative kanter: grafen er vektet, men ingen kanter har negativ vekt. Grafen er enten rettet eller urettet.
- Ingen negative sykler: grafen er vektet, men inneholder ingen sykler med negativ vekt. Grafen er enten rettet eller urettet.

Intervall i et binært søketre 10 poeng

Du er gitt et binært søketre B og to heltall a og b der $a \leq b$. Du skal gi en algoritme som skriver ut verdiene innenfor intervallet $[a, b]$ i *sortert* rekkefølge (fra minst til størst). Det vil si at hvis x er verdien i en node, så skal denne noden skrives ut hvis og bare hvis $a \leq x \leq b$.

Input: Et binært søketre B og to heltall a og b der $a \leq b$
Output: Skriver ut verdiene innenfor intervallet $[a, b]$ i sortert rekkefølge

```
1 Procedure InRange(B, a, b)
  | // ···
```



For det binære søketreet B_1 (vist i eksempelet ovenfor) skal for eksempel `InRange(`B_1`, 2, 7)` skrive ut 2, 5 og 7, og `InRange(`B_1`, 10, 20)` skrive ut 10, 12, 15 og 17.

(a) Oppgi algoritmen din (med psuedokode). Du kan gjøre rimelige antagelser om representasjonen av B. En mer effektiv algoritme er mer poenggivende. Du trenger ikke tenke på formateringen på utskriften.

I de neste deloppgavene skal du anta at B har n noder, er balansert og ikke inneholder noen duplikater.

(b) Anta at a er den minste verdien i B og b er den største verdien i B. Oppgi kjøretidskompleksiteten på algoritmen din med hensyn til n.

(c) Anta at $a = b$. Oppgi kjøretidskompleksiteten på algoritmen din med hensyn til n.

Flest forekomster — 8 poeng

STRATEGI

Input: Et array A med n positive heltall
Output: Tallet som forekommer flest ganger i A

```
1  Procedure MostFrequent1(A)
2      H ← new HashMap()
3      for x in A do
4          H.put(x, 0)
5      m ← A[0]
6      for x in A do
7          f ← H.get(x)
8          H.put(x, f + 1)
9          if H.get(x) > H.get(m) then
10             m ← x
11     return m
```

STRATEGI

Input: Et array A med n positive heltall
Output: Tallet som forekommer flest ganger i A

```
1  Procedure MostFrequent2(A)
2      k ← maximum element in A
3      B ← array of size k + 1
4      for i ← 0 to k do
5          B[i] ← 0
6      m ← A[0]
7      for x in A do
8          B[x] ← B[x] + 1
9          if B[x] > B[m] then
10             m ← x
11     return m
```

Ovenfor ser du to (korrekte) algoritmer som begge tar et array A med positive heltall som input, og returner tallet som forekommer flest ganger. Drøft fordeler og ulemper ved de to ulike strategiene. Du bør oppgi både kjøretidskompleksiteten i *verste* tilfelle, og hva som er den *forventede* kjøretidskompleksiteten, med hensyn til n og k. Den bør begrunnes *kort*. Du bør også oppgi hvilke situasjoner **Strategi 1** bør foretrekkes over **Strategi 2**, og vice versa, med hensyn til konkret kjøretid. Gjøre rede for eventuelle antagelser du gjør underveis.

Finn par som summerer til x — 10 poeng

Du er gitt et array med unike heltall `A` og et heltall x. Du skal skrive en prosedyre som skriver ut alle par av heltall y og z i arrayet, der $y + z = x$. Tallet på en gitt posisjon i arrayet kan maksimalt inngå i én utskrift. Rekkefølgen parene (y, z) skrives ut i spiller ingen rolle, og rekkefølgen innad i parene spiller heller ingen rolle.

Input: Et array `A` av heltall, og et heltall x
Output: Skriver ut alle par $y, z \in$ `A` slik at $y + z = x$

```
1 Procedure FindSummands(A, x)
  | // ···
```

For eksempel skal et kall på `FindSummands(`$[0, 2, 4, 6, 8, 10]$`, 10)` skrive ut parene $(0, 10)$, $(2, 8)$ og $(4, 6)$.

For begge deloppgaver: Gi pseudokode for en algoritme som løser problemet *og* oppgi kjøretidskompleksiteten på algoritmen (den trenger ikke begrunnes). Lavere kjøretidskompleksitet er mer poenggivende.

(a) Skriv en prosedyre `FindSummands` som beskrevet over, med antagelsen om at `A` er *sortert fra minst til størst*.

(b) Skriv en prosedyre `FindSummands` som beskrevet over, men du kan *ikke* anta at `A` er sortert. Hint: Du kan anta $O(1)$ for innsetting, oppslag og sletting i hashbaserte datastrukturer.

To-fargelegge en graf — 10 poeng

Du er gitt en *urettet og sammenhengende* graf $G = (V, E)$. Du skal lage en algoritme som fargelegger nodene i V med to farger R(rød) og B(blå), slik at ingen naboer $(u, v) \in E$ har samme farge; hvis dette *ikke er mulig*, så skal algoritmen skrive ut at det ikke er mulig.

Du kan anta at hver node i V har et felt $v.\texttt{color}$ som initielt er satt til `null`, og at du har tilgang på en funksjon `FlipColor` der `FlipColor(R)` $= B$ og `FlipColor(B)` $= R$.

Input: En sammenhengende graf $G = (V, E)$
Output: En graf der nodene er fargelagt hvis det er mulig

```
1 Procedure TwoColor(G)
  | // ···
```

Whops!-oppgjør 10 poeng

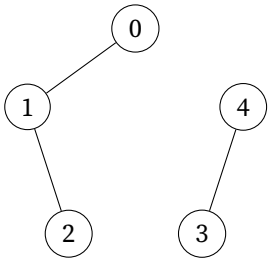
Ansatte på Institutt for informatikk har vært på instituttseminar hvor det har sklidd helt ut med utlån av små beløp mellom hverandre. De skal alle gjøre opp for seg ved å overføre penger via tjenesten Whops!. Dessverre har det oppstått et virvar av interne stridigheter mellom de ansatte, hvor mange har endt opp med å blokkere hverandre på Whops!, som gjør det vanskelig å få oppgjøret til å gå opp. Administrasjonen ser fortvilet på situasjonen, og skjønner at dette problemet vil oppstå igjen i årene fremover, hvor det også skal ansettes vanvittig mange. De har rutiner på plass for å få oversikt over hvor mye hver ansatt har lagt ut eller skylder totalt. Nå ber de om hjelp fra studentene i IN2010 (som på dette tidspunktet har blitt eksperter på algoritmer og datastrukturer) til å finne en generell måte å sjekke om oppgjøret kan gjennomføres gjennom Whops! eller ikke, som også vil skalere etterhvert som Institutt for Informatikk får ubegripelig mange ansatte.

Du setter flittig i gang, og formaliserer problemet som et grafproblem. Du lar $G = (V, E)$, der hver ansatt er representert ved en node $v \in V$, og lar det være en (urettet) kant $\{u, v\} \in E$ mellom to ansatte dersom de ikke har blokkert hverandre på Whops!. For hver ansatt $v \in V$ kan du slå opp i en tabell `T`, der `T[v]` er et heltall som angir hvor mye v skylder eller har lagt ut totalt. Dersom `T[v]` er negativ, betyr det at v skylder penger, men dersom `T[v]` er positiv betyr det at v har lagt ut penger. Du vet at summen av alle tallene i `T` er nøyaktig 0. Merk at det betyr at dersom alle kunne utveksle penger mellom hverandre gjennom Whops! (muligens via andre ansatte) ville oppgjøret alltid gått opp.

Du skal gi en algoritme som tar G og `T` som input og svarer **`true`** hvis oppgjøret kan gjøres gjennom Whops!, og **`false`** dersom det ikke er mulig.

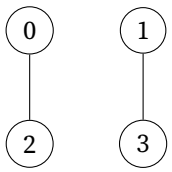
Her er to eksempler på hvordan en G og `T` kan se ut. For eksempelet med G_1 og T_1 kan oppgjøret gjøres gjennom Whops!, men for G_2 og T_2 er det ikke mulig å gjøre oppgjøret gjennom Whops!.

G_1



v	$\mathsf{T}_1[v]$
0	500
1	−200
2	−300
3	−50
4	50

G_2



v	$\mathsf{T}_2[v]$
0	300
1	−250
2	0
3	−50

Whops!-logger 10 poeng

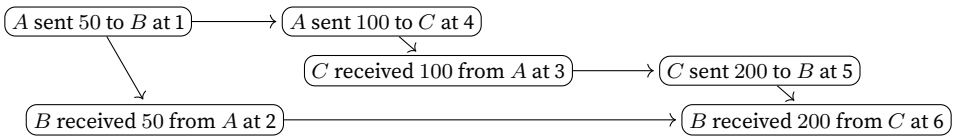
Tjenesten Whops! lar brukere overføre penger mellom hverandre gjennom en app. De prøver i så stor grad som mulig å unngå å lagre informasjon om brukerene sine sentralt, og lar heller informasjonen lagres på brukeren sin egen mobiltelefon.

En gang i blant har de behov for å anskaffe en komplett logg av overføringer for en periode for å kunne feilsøke systemene sine. Da henter de inn *lokale* logger fra noen utvalgte mobiltelefoner i systemet. En lokal logg er en liste av hendelser, der hver hendelse beskriver enten at et beløp er sent eller mottatt. I tillegg inneholder hver hendelse et tidsstempel som forteller når beløpet ble sendt eller motatt. For enkelhets skyld er tidsstemplene representert som positive heltall. Her er et lite eksempel på tre logger fra mobiltelefoner A, B og C.

A	B	C
A sent 50 to B at 1	B received 50 from A at 2	C received 100 from A at 3
A sent 100 to C at 4	B received 200 from C at 6	C sent 200 to B at 5

Dessverre er tidsstemplene fra mobiltelefonene upålitelige. De observerer ofte at et mottak av et beløp forekommer *før* beløpet ble sendt, i følge tidsstemplene. I eksempelet ovenfor mottar C et beløp på 100 *før* A har sendt beløpet til C.

For å approksimere en riktig logg lar de hver hendelse fra de lokale loggene være noder i en rettet graf $G = (V, E)$. Dersom to hendelser u og v kommer direkte etter hverandre i en lokal logg, så finnes det en rettet kant (u, v) i grafen. For hver hendelse u som representerer at et beløp er sendt, så finnes det en rettet kant (u, v) der v er hendelsen for det korresponderende mottaket av det beløpet. Her er grafen for eksempelet ovenfor:



Den ønskede komplette loggen for dette eksempelet skal se slik ut:

A sent 50 to B at 1
B received 50 from A at 2
A sent 100 to C at 4
C received 100 from A at 3
C sent 200 to B at 5
B received 200 from C at 6

Hver node v kan aksessere tidsstempelet fra loggen ved $v.\texttt{ts}$. Skriv en algoritme som tar rettet graf $G = (V, E)$ som input, og skriver ut alle noder i V i en rekkefølge slik at:

- Dersom det finnes en kant fra u til v så skal u skrives ut før v.
- u skal skrives ut før v hvis $u.\texttt{ts} < v.\texttt{ts}$, så lenge det ikke bryter med det første kravet.