Norsk informatikkolympiade 2012–2013 — 1. runde

Uke 45, 2012

Tid: 90 minutter
Tillatte hjelpemidler: Kun skrivesaker. Det er *ikke* tillatt med kalkulator eller trykte eller håndskrevne hjelpemidler.
Instruksjoner: Oppgavesettet består av 16 oppgaver, med fire svaralternativer på hver oppgave. Det er kun ett riktig svar på hver oppgave. Du får fire poeng for hvert riktige svar, null poeng for feil svar, og ett poeng for hver oppgave du ikke svarer på (det vil si at det ikke lønner seg å gjette dersom du ikke vet hva svaret er). Du kan godt krysse av på alternativene i oppgaveteksten underveis, men du *må* føre inn svarene på svararket helt bakerst.
Oppgavene som handler om programmering starter med beskrivelser av temaet de handler om, slik at de som ikke har vært borti programmering før også kan prøve seg på disse oppgavene. De som kan programmere trenger ikke å lese disse beskrivelsene; all koden oppfører seg som i vanlige programmeringsspråk.

Navn: _____

Skole: _____

Studieretning og årstrinn: _____

Hvor gammel er du 30. juni 2013? _____

Epostadresse: _____

1. Du har et Excel-regneark hvor de to øverste cellene begge inneholder tallet 1. I den tredje øverste cellen skriver du formelen `=A1+A2` og kopierer denne nedover, slik at du får:

	A	B
1	`1`	
2	`1`	
3	`=A1+A2`	
4	`=A2+A3`	
5	`=A3+A4`	
6	`=A4+A5`	

Hva blir resultatet i celle A6?

A. 1

B. 5

C. 8

D. 13

Løsningskommentar: Hver celle fra og med den tredje er summen av de to cellene rett ovenfor — dette er kjent som *Fibonacci-tallene*: 1, 1, 2, 3, 5, 8, osv.

2. Hvert bildepunkt (piksel) på skjermen består av en rød, en grønn og en blå fargestripe som kan variere i lysstyrke. Hvilken farge får man hvis man skrur rødt og grønt på full styrke og skrur blått helt av?

A. Fiolett

B. Turkis

C. Brunt

D. Gult

Løsningskommentar: Du kan selv teste dette i bildebehandlingsprogrammer eller i HTML-kode — f.eks med RGB-verdien (Red, Green, Blue) `#ffff00`. Hvis du har en kraftig lupe, kan du prøve å se på en skjerm med den (gjerne en TV-skjerm, siden de har nokså store piksler); da vil du se at hver piksel faktisk består av en rød, en grønn og en blå stripe, og at rødt pluss grønt blir gult. Årsaken til at det fungerer slik er at øyet bare kan registrere rødt, grønt og blått, og at gult lys utløser de røde og grønne sensorene — så hjernen tolker en blanding av rødt og grønt som gult.

3. Ola og Kari leker en gjettelek: Ola velger et hemmelig heltall som er større enn eller lik 1 og mindre enn eller lik 100, og Kari skal gjette hva det er. Hver gang Kari gjetter et tall, forteller Ola om tallet er for stort, for lite eller om det er riktig. Hvis Kari gjetter på smartest mulig måte, hvor mange gjetninger må hun i verste fall bruke (inkludert den siste gjetningen som treffer rett tall)?

A. 6

B. 7

C. 50

D. 100

Løsningskommentar: Den beste strategien er hele tiden å gjette slik at antallet alternativer i verste fall blir minst mulig etter gjetningen. I dette tilfellet er det å velge tallet på midten hele tiden. Å søke etter noe på denne måten ved stadig å halvere søkeintervallet kaller vi *binærsøk*, og det er en viktig ingrediens i svært mange *algoritmer* (fremgangsmåter for å løse problemer). Her bør Kari gjette 50 først, og så (hvis Ola sier at det for lite) 75, og så (hvis Ola igjen sier at det er for lite) 88, osv. Ved hjelp av denne strategien vil man ved hjelp av n gjetninger garantert finne rett tall hvis det er opptil $2^n - 1$ tall å velge mellom. Dermed trenger vi 7 gjetninger når det er 100 tall, siden $2^6 < 100$ og $2^7 \geq 100$.

4. I *boolsk logikk* jobber vi kun med verdiene **true** og **false**. Vi skal se på fire operatorer som kan brukes til å lage boolske *uttrykk* (formler):

   - NOT-operatoren gir ut motsatt sannhetsverdi av det man mater inn i den: NOT a blir **true** dersom a er **false**, og **false** dersom a er **true**.
   - AND-operatoren brukes med to verdier: a AND b, og gir ut **true** bare hvis både a og b er **true**.
   - OR-operatoren brukes slik: a OR b, og gir ut **true** så lenge minst én av a og b (eller begge to) er true. Altså gir den ut **false** bare hvis både a og b er **false**.
   - XOR-operatoren brukes slik: a XOR b, og gir ut **true** så lenge én av a og b er **true** og den andre er **false**. Hvis a og b har samme verdi, gir XOR ut **false**.

   Disse operatorene kan kombineres til større uttrykk. Parenteser angir rekkefølgen uttrykket skal evalueres i, slik som vi kjenner fra matematikken.

   Ola og Kari skal kjøpe bil. Det er veldig viktig for Ola at bilen er rød. Samtidig er det veldig viktig for Kari at bilen *enten* har soltak *eller* at den ikke er rød. De må selvfølgelig finne en bil som oppfyller begges ønsker. Dette kan uttrykkes som a AND (b XOR (NOT a)), hvor a betyr at bilen er rød og b betyr at bilen har soltak. Hvilken bil oppfyller kriteriet?

   A. En rød bil med soltak (a er true og b er true)

   B. En rød bil uten soltak (a er **true** og b er **false**)

   C. En svart bil med soltak (a er **false** og b er **true**)

   D. En svart bil uten soltak (a er **false** og b er **false**)

5. Vi har et boolsk uttrykk som vi vet at gir følgende resultater når vi mater inn forskjellige sannhetsverdier for a og b:

a	b	Uttrykket gir...
false	**false**	**false**
false	**true**	**false**
true	**false**	**true**
true	**true**	**false**

Hvilket av følgende uttrykk oppfører seg slik?

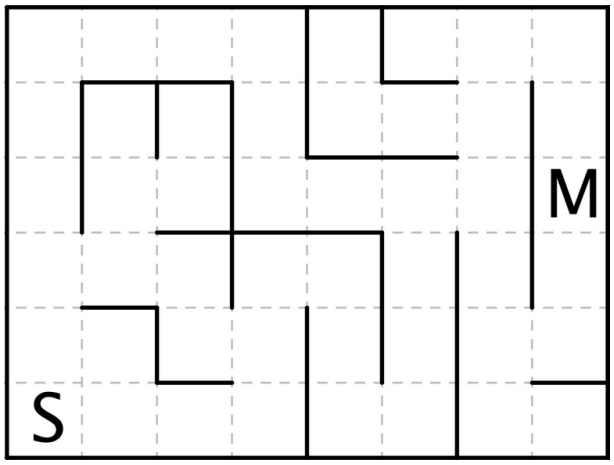
A. a AND b

B. a OR (NOT (a XOR b))

C. a XOR (NOT b)

D. a AND (NOT b)

6. Google jobber i disse dager med å lage en selvkjørende bil. Vi har laget en enklere utgave med en robot som beveger seg basert på følgende tre kommandoer: `F`: fram en rute; `V`: sving 90 grader til venstre på stedet; `H`: sving 90 grader til høyre på stedet. Roboten vil nekte å kjøre inn i hindringer og befinner seg til å begynne med i ruten merket start med fronten rettet nordover (oppover på arket):



Hva er det minste antallet kommandoer som må til for at roboten skal kunne komme seg fra start til mål (ruten som er merket med M)?

A. 18

B. 22

C. 23

D. 24

Løsningskommentar: Husk å telle kommandoer i stedet for ruter. Det er en ekstra kommando for hver sving, fordi roboten bruker en kommando på å svinge uten at den forflytter seg. Den optimale kommandosekvensen er `FFFFFHFFFHFFVFFFHFFVFVFF` (evt. `VFFHFHFF` på slutten i stedet).

7. Roboten fra forrige oppgave skal brukes til å utforske Saturns måne Hyperion. Der er internett tregt, og overføringen av kommandoer fra jorda må skje på en mest mulig effektiv måte. Hver kommando må ha en unik kode som består av bits (nuller og ettall). Alle kodene skal ha like mange bits. Hvor mange bits trenger man per kommando når det finnes tre kommandoer (F, V og H)?

   A. 2

   B. 3

   C. 8

   D. 32

   Løsningskommentar: Vi trenger 2 bits. F.eks kan vi bruke disse kodene: 00 = F, 01 = V og 10 = H. Vi kommer uansett til å få en mulighet til overs (11 i dette eksempelet), men vi kan ikke klare oss med færre bits, for én bit holder bare til to kommandoer.

8. Kakemonsteret vil så veldig gjerne ha 119 kaker, men har litt problemer med å fortelle dette til datamaskinen sin (som skal bestille dem), ettersom datamaskinen til kakemonsteret bare forstår binære tall og ordet "kake". Hvordan skriver man 119 binært?

   A. 100111

   B. 1110111

   C. 1110110

   D. 11110011

   Løsningskommentar: Bakerste siffer er verdt 1, nest bakerste 2, tredje bakerste 4 osv. $119 = 1 \cdot 64 + 1 \cdot 32 + 1 \cdot 16 + 0 \cdot 8 + 1 \cdot 4 + 1 \cdot 2 + 1 \cdot 1$, altså 1110111.

9. I de fleste programmeringsspråk brukes operatoren `=` på en annen måte enn i matematikken. Den instruerer nemlig datamaskinen om å regne ut verdien av uttrykket på høyre side av likhetstegnet og legge den inn i *variabelen* på venstre side. (Verdien *kopieres* alltid, selv hvis høyresiden er en enkelt variabel — den *flyttes* ikke.) En variabel holder på en verdi helt frem til du legger noe annet inn i den; da forsvinner den gamle verdien. Linjer som står etter hverandre utføres etter tur. For eksempel vil

   ```
   x = 4
   x = x + 3
   ```

   gjøre at variabelen `x` ender opp med å inneholde verdien 7.

   Hva blir verdien av `x` etter at følgende kode er utført?

   ```
   x = 10
   x = x + x
   x = x - 5
   ```

A. 5

B. 10

C. 15

D. 20

> **Løsningskommentar:** Etter første linje er utført har `x` fått verdien 10. Når andre linje kjøres, puttes denne verdien inn der `x` står på høyre side, slik at vi får $10 + 10$ som høyre side. Dette blir 20, og kopieres inn i `x`. På siste linje har `x` på høyre side verdien 20, som gjør at venstre side blir $20 - 5 = 15$.

10. La oss si at `x` og `y` allerede inneholder hvert sitt tall, og at vi ønsker å bytte om på tallene i de to variablene (slik at hvis `x` inneholdt 4 og `y` inneholdt 7, skal `x` ende opp med 7 og `y` skal ende opp med 4). Hvilken av følgende kodesnutter gjør dette riktig?

    A.
    ```
    x = y
    ```

    B.
    ```
    x = y
    y = x
    ```

    C.
    ```
    y = x
    t = y
    x = y
    ```

    D.
    ```
    t = x
    x = y
    y = t
    ```

> **Løsningskommentar:** En vanlig feil her er å overskrive en av variablene før man leser den av — husk at hvis man starter med `x = y`, vil `x` og `y` få samme verdi, og verdien som først lå i `x` er borte for alltid. Vi trenger en tredje variabel til å lagre en av verdiene midlertidig.

11. En *funksjon* er en navngitt samling med programinstruksjoner, som kan *kalles* (startes) med *inndata* og *returnere* (gi tilbake) *utdata*. Kommandoen `if` sjekker om uttrykket som står i parenteser er **true** eller **false**; hvis det er **true**, gjøres det som står i krøllparentesene etter `if`; hvis det er **false**, gjøres det som står i krøllparentesene etter den tilhørende `else`'n. `return` avslutter funksjonen og returnerer resultatet av uttrykket som står på samme linje. `<` er den vanlige mindre-enn-operatoren — f.eks. vil `3 < 4` bli **true**, og `4 < 3` blir **false**. `3 < 3` blir også **false**.

    Hva gjør funksjonen under? Merk at den første linjen bare oppgir navnet på funksjonen (`f`) og navnene på inndataene (`x`, `y` og `z`).

```
f(x, y, z) {
    if (x < y) {
        if (x < z)
```

```
            return x
        else
            return z
    } else {
        if (y < z)
            return y
        else
            return z
    }
}
```

A. Finner den minste av `x`, `y` og `z`

B. Løser en likning hvor `x`, `y` og `z` er de ukjente

C. Finner gjennomsnittet av `x`, `y` og `z`

D. Finner medianen av `x`, `y` og `z`

12. UKULT (Utrolig Kjedelige og Unaturlige LekeTøy) har produsert følgende spill for små barn. De har skrevet ut en haug papirbiter med bokstaven `a` på og en haug med ordet `bc` på. Fabrikanten bestemmer seg for å trykke noen fun facts på pakken, blant annet hvor mange forskjellige ord bestående av 9 bokstaver man kan lage ved å sette sammen lapper. Merk at `aabcbcaaa` er et veldig fint ord på lengde 9 etter UKULT sine standarder; ordene trenger altså ikke å eksistere i noe ordentlig språk. Hvor mange slike ord finnes det av lengde 9?

A. 23

B. 34

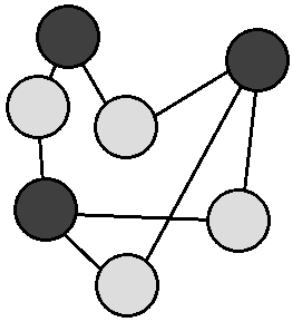
C. 55

D. 63

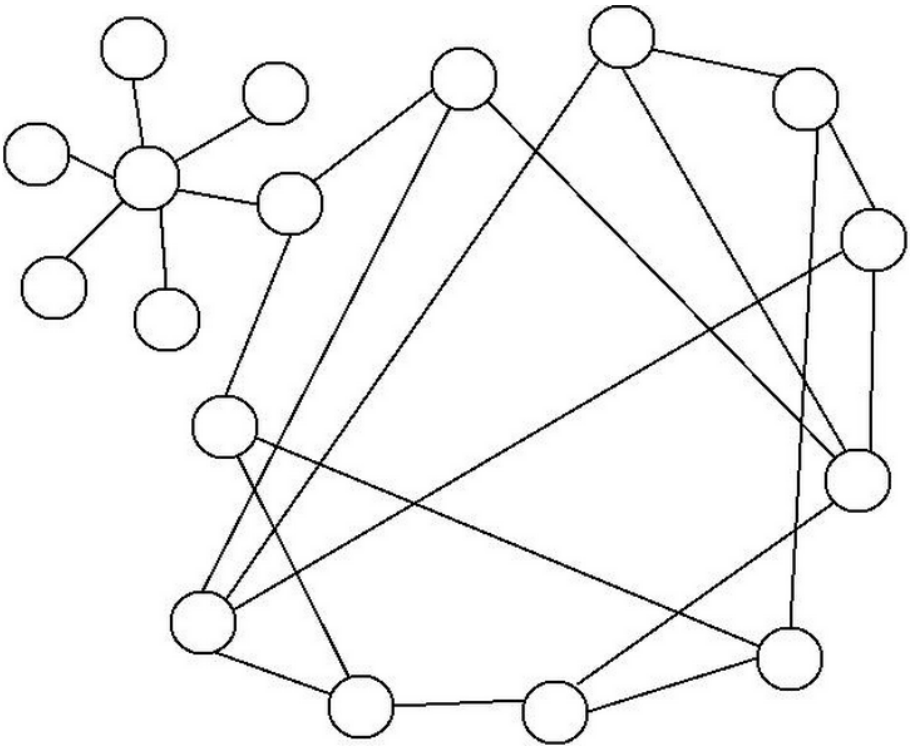
Løsningskommentar: Denne oppgaven er inspirert av en teknikk som kalles *dynamisk programmering*, som er mye brukt i programmeringskonkurranser. Denne teknikken går ut på å løse mindre utgaver av samme problem, og bruke løsningen på de mindre problemene til å finne løsningen på det store problemet. Vi legger merke til at det vi like gjerne kunne laget en oppgave om å finne antallet ord av lengde 8 eller 10, eller et hvilket som helst annet tall. I stedet for å hoppe rett på problemet med å finne antall ord av lengde 9, skal vi løse et enklere slikt problem: antallet ord med lengde 1. Det finnes bare ett slikt ord: `a`. Så prøver vi oss på antall ord av lengde 2: `aa` og `bc`.

Vi kan konstruere nye ord av lengde n ved å legge `a` på slutten av ord av lengde $n-1$ og ved å legge `bc` på slutten av ord av lengde $n-2$. Antallet ord av lengde n blir altså summen av antall ord av lengde $n-1$ og antall ord av lengde $n-2$, og som i oppgave 1 får vi Fibonacci-tallene (bortsett fra det første ettallet): 1, 2, 3, 5, 8, 13, 21, 34, 55.

13. I datasammenheng er en *graf* en samling av *noder* (punkter; her tegnet som sirkler) og *kanter* (streker) mellom nodene. Vi sier at en graf kan fargelegges med k farger dersom man kan fargelegge nodene ved hjelp av k forskjellige farger slik at hver kant går mellom noder med forskjellig farge (dvs. at noder med samme farge aldri er koblet direkte i hverandre). Grafen nedenfor er et eksempel på en graf som kan fargelegges med 2 farger.



Hva er det minste antallet forskjellige farger man trenger for å fargelegge grafen nedenfor?



A. 2

B. 3

C. 4

D. 5

Løsningskommentar: Algoritmen for å sjekke om vi kan farge en graf med to farger er ganske enkel når man først har kommet på den: Start på en vilkårlig node og farg den med

farge nummer 1. Så farger vi alle noder som er direkte koblet til denne med farge nummer 2. Så farger vi nabo-noder av disse nodene med farge 1, men vi passer på at vi aldri farger en node to ganger. Om dette fører til en konflikt, er det umulig å farge grafen med kun to farger. Slik fortsetter vi med å farge alle direkte nabo-noder av en farget node med den andre fargen, helt til vi blir ferdig eller det kommer en konflikt.

I vår graf får vi en konflikt med to-farge-algoritmen. Men konfliktene kan løses ved å introdusere en tredje farge og farge konfliktnodene med den fargen. Siden vi fant en løsning med tre farger og vi vet at det er umulig med to, blir svaret tre.

14. Merk: De siste tre oppgavene vil være utfordrende for dem som ikke har erfaring med programmering fra før.

    Funksjoner i programmeringsspråk kan også inneholde kall til seg selv. Dette kalles *rekursive* funksjoner. Når en rekursiv funksjon kaller seg selv, starter en ny utgave av den samme funksjonen, og den gamle utgaven venter til den nye er blitt ferdig. Gitt følgende definisjon av funksjonen `f`:

```
f(x) {
    if(x < 2)
        return 1
    else
        return x * f(x - 1)
}
```

    Hva returnerer funksjonskallet `f(7)`?

    A. 720

    B. 4001

    C. 5040

    D. 40320

Løsningskommentar: Denne funksjonen regner ut fakultet av `x`, altså $x!$, som er produktet av alle heltallene fra og med 1 til og med x. $7! = 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 5040$. Matematisk sett kan også fakultet defineres ved rekursjon, ved at $0! = 1$ og at $x! = x \cdot (x-1)!$ når x er større enn eller lik 1. (Koden skriver `x < 2` i stedet for `x < 1` fordi 1! også blir 1).

15. Et array er en samling med et bestemt antall elementer. `A[0]` er det første elementet, `A[1]` er det andre osv. Antallet elementer totalt er `length(A)`. Det siste elementet er dermed `A[length(A) - 1]`. Vi har et array som inneholder heltall, og vi ønsker å sortere elementene i stigende rekkefølge slik at `A[0]` er minst, `A[1]` er nest minst osv. Til å gjøre dette har vi laget følgende sorteringsfunksjon, der en av linjene dessverre har blitt overskrevet:

```
insertionSort(A) {
    j = 1
```

```
    while (j < length(A)) {
        k = A[j]
        i = j - 1
        while (i >= 0 AND A[i] > k) {
            ??????????????????????????????????????????????????
            i = i - 1
        }
        A[i+1] = k
        j = j + 1
    }
}
```

Hva skal stå på linjen som har blitt overskrevet med spørsmålstegn for at `A` skal bli sortert i stigende rekkefølge?

A. `A[i] = k`

B. `A[j] = A[i]`

C. `k = A[i]`

D. `A[i + 1] = A[i]`

Løsningskommentar: Insertion-sort fungerer ved å hele tiden sørge for at arrayet er sortert opp til (men ikke med) `j`. Når vi utvider dette sorterte området, tar vi det første tallet utenfor, altså `A[j]`, lagrer det, og flytter alle tallene på lavere indeks enn `j`, og som har høyere verdi, ett hakk oppover for å gjøre plass til det nye tallet. Det er denne flyttingen ett hakk oppover som den manglende linjen utfører.

16. Merk: Denne oppgaven er ekstra utfordrende og tidkrevende — forsøk å løse alle de andre oppgavene før du prøver deg på denne!

Gitt følgende definisjon av funksjonen `g`:

```
g(a, x) {
    if (a == 10) {  // A == B sjekker om A og B er like store
        if (x % 2 == 0)  // % er modulo-operatoren, se under
            return 1
        else
            return 0
    }

    r = 0
    s = 1
    while ((x * 10) / s > 0) {  // Vi bruker divisjon som alltid runder ned
        r = r + g(a + 1, (x / s) * 10 * s + s * a + x % s)
        s = s * 10
    }
    return r
}
```

Merk:

- `==` sjekker om to verdier er like
- `%` er modulo-operatoren: `a % b` gir resten etter å ha delt `a` på `b`; f.eks. vil `26 % 7` bli 5.
- `/` er heltallsdivisjon (dvs. at den alltid runder ned); f.eks. vil `26 / 7` bli 3.

Hva returnerer kallet `g(2, 1)`?

A. 121680

B. 161280

C. 253680

D. 362880

Løsningskommentar: g(2,1) generer alle permutasjoner (omstokkinger) av sifrene fra 1 til 9. Men den teller bare opp de permutasjonene som ender på et partall. Dette er 4/9 av det totale antallet, altså $9! \cdot 4/9 = 8! \cdot 4 = 161280$.

Svarark

Oppgave	**A**	**B**	**C**	**D**	Poeng (til bruk for læreren)
1			x		
2				x	
3		x			
4	x				
5				x	
6				x	
7	x				
8		x			
9			x		
10				x	
11	x				
12			x		
13		x			
14			x		
15				x	
16		x			
Poengsum					

Til læreren: Husk at korrekt svar gir 4 poeng, feil svar gir 0 poeng, og fraværende svar gir 1 poeng.