Norsk informatikkolympiade 2013–2014 — 1. runde

Sponset av

UNIVERSITETET I BERGEN
Institutt for informatikk

STARTUP LAB

FFI Forsvarets forskningsinstitutt

Uke 46, 2013

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.

FASIT

1. Du ønsker å sette opp et plankegjerde for å hindre elger i å komme seg inn på serverfarmen din. Gjerdet skal beskytte en 15 meter lang rett linje. Hver meter skal det settes ned en stolpe, og mellom to etterfølgende stolper skal det spikres opp 2 planker. Totalt hvor mange stolper og planker trengs det for å sette opp gjerdet?

   A. 45

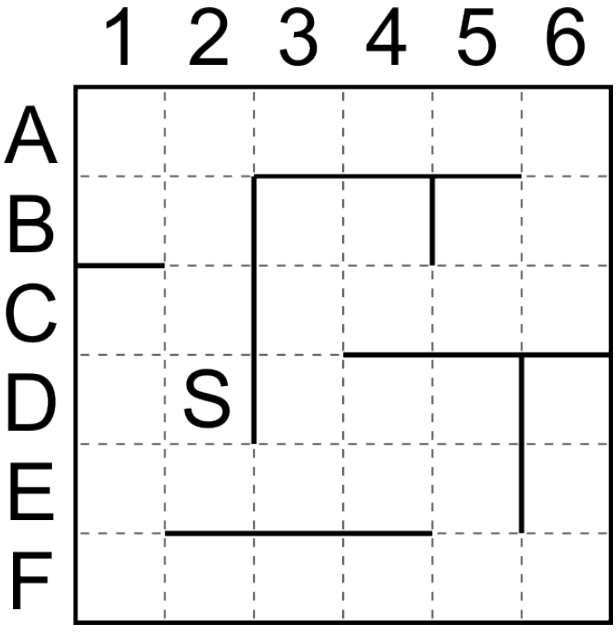
   B. 46

   C. 47

   D. 48

   Løsningskommentar: Ettersom det må være stolper både på begynnelsen og slutten av gjerdet så trenger man 16 stolper. Det trengs 30 planker.

2. Vi har konstruert 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. Et *program* består av en kjede med kommandoer som roboten skal utføre i rekkefølge. Roboten vil nekte å kjøre inn i hindringer; dersom den får en kommando om å gå framover men det er en vegg rett foran den så vil den ignorere denne kommandoen og fortsette på neste kommando i programmet. Roboten befinner seg til å begynne med i rute D2 på kartet (merket med bokstaven S) med fronten rettet nordover (oppover på arket):



   På hvilken rute vil roboten stå etter å ha utført følgende program?

   ```
   F F H F H F F F V F F V F F H F V V V F
   ```

   A. E5

   B. A6

   C. B4

   D. B2

3. Anne har en pose som inneholder 1000 små kuler. 999 av kulene er av massivt jern, men 1 av kulene har en kjerne av uran. Kari har en maskin som detekterer radioaktivitet som hun vil bruke til å finne ut hvilken kule som er av uran. Hun kan legge så mange kuler som hun ønsker i maskinen, og etter ett minutt så forteller maskinen henne om urankulen er blant de kulene som er i maskinen. Hun ønsker å bruke så få målinger som mulig for å finne ut hvilken av kulene som er av uran. Hvor mange målinger må hun i verste fall bruke før hun kan være helt sikker på hvilken kule som er av uran?

   A. 10

   B. 11

   C. 62

   D. 999

   Løsningskommentar: Ved å dele kulene i to like store hauger og måle den ene kan Anne finne ut om kulen er blandt de som ble målt eller i den andre haugen. Etter en måling har hun redusert søkerommet til å være 500 kuler; etter 2 målinger 250; etter 3 målinger 125; etter 4 målinger 63 (i værste fall), osv. Hvis hun følger denne teknikken så er det etter 9 målinger bare to muligheter for hvilken kule som er av uran. Ved å måle den ene av de to så vet hun om det er den eller den andre.

4. Leketøysprodusenten UKULT (Utrolig Kjedelige og Unaturlige LekeTøy) har nylig utgitt brettspillet "Snu brikken!" som spilles av kun 1 spiller. Spillet består av 2000 brikker som ser helt like ut på baksiden, men 1000 av dem har rød forside og 1000 av dem har blå forside. Før spillet begynner plukker man ut 1000 tilfeldige brikker og legger de ut med forsiden ned. (De restrerende brikkene legges tilbake i esken og brukes ikke.) Deretter snur man opp brikkene én etter én helt til alle brikkene har forsiden opp. Hver gang man snur en blå brikke så får man poeng tilsvarende antall røde brikker som har blitt snudd opp. Hva er det største antall poeng som det er mulig å få totalt i løpet av spillet?

   A. 999

   B. 125250

   C. 250000

   D. 1000000

   Løsningskommentar: Det ideelle er om alle de røde brikkene blir snudd før noen av de blå blir snudd. Hvis det er R røde og B blå så blir poengsummen da $R \times B$. Når vi har$R + B = 1000$ så får vi maksimalverdien til dette uttrykket når $R = B = 500$.

5. 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 x blir **true** dersom x er **false**, og **false** dersom x er **true**.
   - AND-operatoren brukes med to verdier: x AND y, og gir ut **true** bare hvis både x og y er **true**.

- OR-operatoren brukes slik: x OR y, og gir ut **true** så lenge minst én av x og y (eller begge to) er true. Altså gir den ut **false** bare hvis både x og y er **false**.
- XOR-operatoren brukes slik: x XOR y, og gir ut **true** så lenge én av x og y er **true** og den andre er **false**. Hvis x og y har samme verdi, gir XOR ut **false**.

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

To uttrykk sies å være *ekvivalente* hvis de gir det samme svaret uansett hvilke verdier man putter inn i de. For eksempel så er uttrykkene

x XOR y

og

(x AND NOT y) OR (y AND NOT x)

ekvivalente, da begge gir **true** hvis og bare hvis nøyaktig én av x og y er **true**.

Hvilket av følgende uttrykk er ekvivalent med

x AND (y OR z) ?

A. (x AND y) OR z

B. (x OR y) AND z

C. (x OR y) AND (x OR z)

D. (x AND y) OR (x AND z)

6. Hvilket av følgende uttrykk er ekvivalent med

   x XOR (NOT(y XOR x)) ?

   A. NOT(y)

   B. NOT(x XOR y)

   C. NOT(x) AND y

   D. NOT(x OR y)

Løsningskommentar:

x XOR (NOT(y XOR x)) =
NOT(x XOR (y XOR x)) =
NOT(y XOR (x XOR x)) =
NOT(y XOR **false**) =
NOT(y)

7. I det binære tallsystemet så bruker man kun to sifre: 0 og 1. Sifferet lengst til høyre er 1'er plassen, til venstre for dette har man 2'er plassen, deretter 4'er plassen, 8'er plassen, 16-plassen, og så videre. Hvert siffer er altså dobbelt så mye verdt som sifferet det står til venstre for. Verdien av et tall i det binære tallsystemet finner man ved å summere verdiene for posisjonene som har sifferverdi 1. For eksempel så tilsvarer det binære tallet 11010 verdien $16+8+2$, altså 26. Hvordan skriver man tallet 73 i binært?

A. 1001001

B. 1010101

C. 1011101

D. 1100011

Løsningskommentar: $73 = 64 + 8 + 1$

8. På en datamaskin har man som regel bare et visst antall binære sifre (også kalt *bits*) til å uttrykke tall med. Hva er det største tallet man kan uttrykke med 16 binære sifre?

A. 32767

B. 32768

C. 65535

D. 65536

Løsningskommentar: Det største tallet man kan skrive er det binære tallet 1111111111111111. Dette tallet er 1 mindre enn 2^{16}.

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 = 5
x = x + 2
x = x * x
```

A. 5

B. 25

C. 27

D. 49

Løsningskommentar: Først settes x til å være 5. Detter settes x til å være 2 mer enn x, altså 7. Til sist blir x satt til å være x $\times$ x, altså til $7 \times 7 = 49$.

10. 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 (`a`, `b` og `c`).

```
f(a, b, c) {
    if (a < b) {
        if (b < c) {
            return c - a
        } else {
            if (a < c) {
                return b - a
            } else {
                return b - c
            }
        }
    } else {
        if (a < c) {
            return c - b
        } else {
            if (b < c) {
                return a - b
            } else {
                return a - c
            }
        }
    }
}
```

    A. Finner differansen mellom det største og minste tallet blant `a`, `b` og `c`

    B. Finner en løsning på annengradsligningen $ax^2 + bx + c = 0$

    C. Finner medianen av `a`, `b` og `c`

    D. Finner den minste av alle mulige differanser mellom to av tallene `a`, `b` og `c`

11. Anta at du har en funksjon `max(x,y,z)` som returnerer det største av tallene `x`, `y` og `z`. (For eksempel så returnerer `max(4, 7, 2)` verdien 7.) Hvilket av følgende uttrykk gir medianen av `x`, `y` og `z`.

A. `max(x, x , z) + max(z, z, y) - max(y, y, x)`

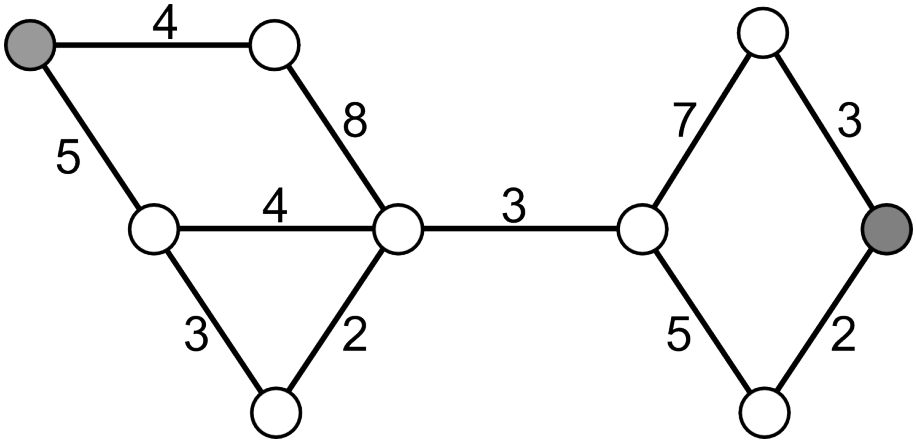
B. `x + y + z - max(x, y, z) + max(-x, -y, -z)`

C. `max(x, y, z) - max(x - y, y - z, z - x)`

D. `x + y + z - max(x - y, y - z, z - x) - max(y - x, z - y, x - z)`

Løsningskommentar: Hvis man summerer tre tall, trekker fra verdien av det største, og trekker fra verdien av det minste, så sitter man igjen kun med medianen. Medianen er dermed $x + y + z - max(x, y, z) - min(x, y, z)$. Vi har ikke min funksjonen tilgjengelig her, men siden $max(-x, -y, -z) = -min(x, y, z)$ så kan vi bruke max funksjonen i stedet, hvis vi trikser med fortegnene.

12. I datasammenheng er en *graf* en samling av *noder* (punkter; her tegnet som sirkler) og *kanter* (streker) mellom nodene. Vi vil i denne oppgaven se på grafer der hver kant har en *lengde* (her markert med tallene ved siden av strekene). *Avstanden* mellom to noder er den totale lengden av den korteste stien (sekvensen av kanter) mellom dem.



Hva er avstanden mellom de to fargelagte nodene i tegningen over?

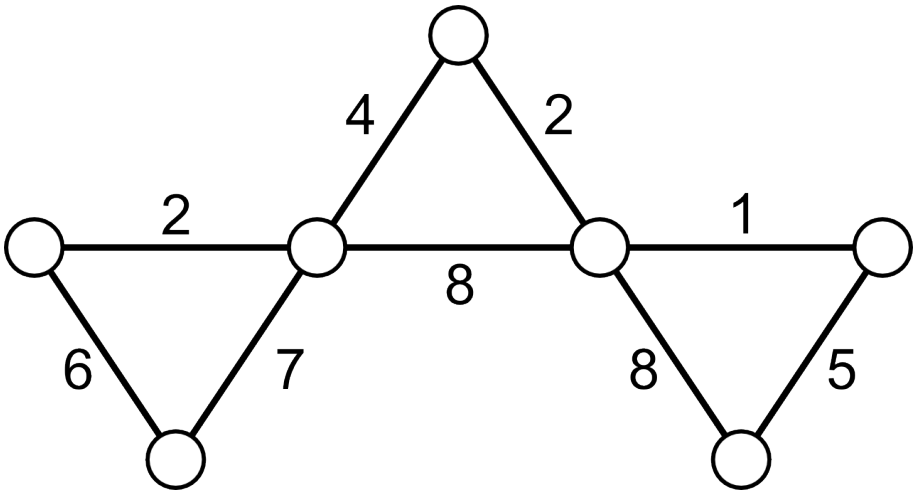
A. 17

B. 19

C. 22

D. 25

13. *Diameteren* på en graf er definert som avstanden mellom de to nodene som har størst avstand. (Merk at avstanden mellom to noder fortsatt er lengden på den korteste stien mellom de.)



Hva er diameteren på grafen i tegningen over?

A. 7

B. 9

C. 19

D. 21

Løsningskommentar: De to nodene som er lengst unna hverandre er de to nederste. Den korteste stien mellom disse nodene har lengde 19.

14. **Merk:** De siste tre oppgavene vil være ekstra 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,y) {
    if (x == 0) {
        return y
    } else {
        if (y == 0)
            return 0
        else
            return f(x-1,y) + f(x,y-1) + f(x-1,y-1)
    }
}
```

Merk at `==` sjekker om to verdier er like.

Hva returnerer funksjonskallet `f(4,3)`?

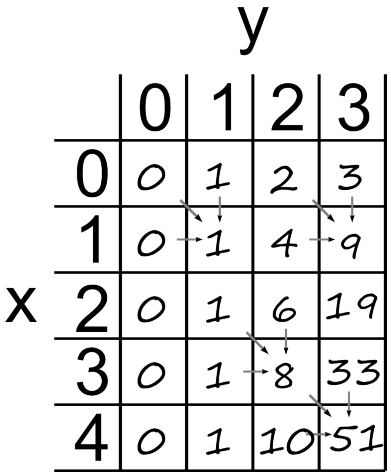
A. 51

B. 55

C. 58

D. 64

Løsningskommentar: Denne oppgaven kan løses med en teknikk som kalles *dynamisk programering*. Tegn opp et regneark med 5 rader og 4 kolonner. Marker radene med forskjellige verdiene for x fra 0 til 4 og kolonnene med forskjellige verdier for y fra 0 til 3. Fyll inn cellene der$x = 0$ med y og cellene der $y = 0$ med 0. Verdien i de andre cellene kan fåes ved å summere de tre cellene over og til venstre. Ved å jobbe seg systematisk bortover finner man fort verdien av $f(4, 3)$.



15. En *while-løkke* utfører koden inni krøllparentesene sine gjentatte ganger. Før hver utførelse sjekker løkken betingelsen inni parentesene rett etter `while`. Dersom betingelsen er usann, avsluttes løkken, og man fortsetter med koden etter avslutningskrøllparentesen (dersom det er noe kode der i det hele tatt).

    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 stokke om på tallene i arrayet slik at vi får de samme tallene men i en tilfeldig rekkefølge. Vi har en funksjon `random(X, Y)` som gir oss et tilfeldig heltall R slik at $X \leq R \leq Y$. For å stokke om på tallene har vi laget følgende funskjon, der en av linjene dessverre har blitt overskrevet:

```
shuffle(A) {
    j = 1
    while (j < length(A)) {
        k = A[j]
        r = random(0, j)
        ??????????????
        A[r] = k
        j = j + 1
    }
}
```

Hva skal stå på linjen som har blitt overskrevet med spørsmålstegn for at `shuffle(A)` skal fungere som en omstokkingsfunksjon?

A. `A[j] = r`

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

C. `k = k + A[r]`

D. `A[j-1] = A[r]`

Løsningskommentar: For hvert element $A[j]$ så plukker vi ut et element i en tilfeldig posisjon mellom 0 og j som vi skal bytte ut elementet med. (Dersom vi velger posisjonen j så *byttes* elementet ut med seg selv, altså beholder sin opprinnelige posisjon.) For å bytte om på element $A[j]$ og $A[r]$ så må vi gjøre

```
k = A[j]
A[j] = A[r]
A[r] = k
```

Den endre av disse linjene er den som har blitt overksrevet.

16. Gitt følgende definisjon av funksjonen `v`:

```
v(a, x) {
    r = 0;
    if (a == 0) {
        if (x > 0) {
            m = 5
            while (x % m == 0) {
                m = m * 5
                r = r + 1
            }
        }
    } else {
        r = v(a - 1, x * 2)
        r = r + v(a - 1, x * 2 + 1)
   }
    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.

Hva returnerer kallet `v(10, 0)`?

A. 253

B. 448

C. 1277

D. 2985

Løsningskommentar: `v` vil ende opp med å bli kallt med verdien $a = 0$ for alle mulige x mellom 0 og $2^{10} - 1$ (1024 forsjkellige verdier). For hver av disse, så teller den opp hvor mange ganger vi kan dele x med 5 og fortsatt få heltalls svar (mer presist, største heltall k slik at 5^k deler x). 204 av tallene mellom 0 og 1023 er delelige med 5 minst en gang. 40 av disse er delelige med 5 enda en gang. 8 av disse er delelige med 5 enda en gang. 1 av disse (tallet 625) er delelige med 5 enda en gang. Totalt så lar tallene seg dele med 5 $204 + 40 + 8 + 1 = 253$ ganger.

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.