NTNU
Kunnskap for en bedre verden

Institutt for datateknikk og informasjonsvitenskap

Eksamensoppgave i TDT4120 Algoritmer og datastrukturer

Faglig kontakt under eksamen	Magnus Lie Hetland
Telefon	918 51 949
Eksamensdato	10. desember, 2018
Eksamenstid (fra–til)	09:00–13:00
Hjelpemiddelkode/tillatte hjelpemidler	D
Annen informasjon	Oppgavearkene leveres inn, med svar i svarrute under hver oppgave
Målform/språk	Bokmål
Antall sider (uten forside)	7
Antall sider vedlegg	0

Informasjon om trykking av eksamensoppgave

Originalen er

1-sidig ☑ **2-sidig** ☐

sort/hvit ☑ **i farger** ☐

Skal ha flervalgskjema ☐

Kontrollert av

Dato Sign

Merk: Studenter finner sensur i Studentweb. Har du spørsmål om din sensur må du kontakte instituttet ditt. Eksamenskontoret vil ikke kunne svare på slike spørsmål.

Spørsmål til fagstaben?

Vi kan kun svare på forespørsler om mulige feil, mangler eller uklarheter i oppgaveteksten.

Vi går én runde og har begrenset tid. Les igjennom alle oppgavene først og ha evt. spørsmål klare!

Les dette nøye

(*i*) Les hele eksamenssettet nøye før du begynner!

(*ii*) Skriv svarene dine i svarrutene og levér inn oppgavearket. Bruk gjerne blyant! Evt. kladd på eget ark først for å unngå overstrykninger, og for å få en egen kopi.

(*iii*) Ekstra ark kan legges ved om nødvendig, men det er meningen at svarene skal få plass i rutene på oppgavearkene. Lange svar teller ikke positivt.

Merk: Varianter av de foreslåtte svarene nedenfor vil naturligvis også kunne gis uttelling, i den grad de er helt eller delvis korrekte.

Oppgaver med løsninger

5% 1. I en sammenhengende graf med $n \geq 1$ noder og m kanter, hva er det minste og største antall kanter et spenntre kan ha?

$n-1$ og $n-1$

Her har noen også tolket det som at man tar maksimum over mulige verdier for n, og dermed endt med et minimum på 0. Det gir ikke helt mening, siden man da ikke kan finne noe maksimum. Likevel gis dette en viss uttelling.

Relevant læringsmål: Vite hva *spenntrær* og *minimale spenntrær* er

5% 2. I en Huffman-kode for et alfabet med $n \geq 1$ tegn, hva er den minste og største lengden et kodeord for ett tegn kan ha?

1 og $n-1$

Oppgaven er litt problematisk, siden den eksplisitt angir $n \geq 1$, og dermed antyder at man kan ha et alfabet med ett tegn. Dersom man konkluderer med at det ikke gir mening, får man svaret over, som gir full uttelling. (En grunn til at det ikke gir mening er at man da får en kode som ikke kan dekodes.) Dersom man konkluderer med at det gir mening, får man 1 og $n-1$ for $n > 1$ og 0 for både minimum og maksimum når $n = 1$. (Grunnen til at det kan gi mening er at det er dette man får fra prosedyren HUFFMAN dersom $C = \emptyset$.) Det gis full uttelling. Det gjør også om man oppgir 0 og $n-1$, selv om det ikke er helt korrekt.

Relevant læringsmål: Forstå HUFFMAN og *Huffman-koder*

5% 3. Anta at Π er en forgjengermatrise for alle-til-alle-varianten av korteste vei-problemet. Hva representerer π_{ij}, om du antar $\pi_{ij} \neq \text{NIL}$?

Forgjengeren til j langs en av de korteste veiene fra i til j.

Her er det også helt greit om man omtaler veien/stien som den korteste.

Relevant læringsmål: Forstå *forgjengerstrukturen* for *alle-til-alle*-varianten av korteste vei-problemet (PRINT-ALL-PAIRS-SHORTEST-PATH)

5% 4. Hva er et *snitt* (*cut*) i et flytnett (*flow network*) $G = (V, E)$?

En partisjonering (S, T) av V der $s \in S$ og $t \in T$.

Relevant læringsmål: Forstå hva *snitt*, *snitt-kapasitet* og *minimalt snitt* er

5% 5. Hva er et nodedekke (*vertex cover*) for en graf $G = (V, E)$?

En mengde $V' \subseteq V$ slik at om $(u, v) \in E$ så er minst én av u og v i V'.

Relevant læringsmål: Kjenne de NP-komplette problemene CIRCUIT-SAT, SAT, 3-CNF-SAT, CLIQUE, VERTEX-COVER, HAM-CYCLE, TSP og SUBSET-SUM

5% 6. TABLE-INSERT har i verste tilfelle kjøretid $\Theta(n)$, men *amortisert* kjøretid $O(1)$. Hva betyr det?

Det er snittet av n operasjoner.

Relevante læringsmål: Kunne definere *amortisert analyse*. Forstå hvordan *dynamiske tabeller* fungerer (TABLE-INSERT)

5% 7. I n-tårns-problemet har man et $n \times n$-sjakkbrett og skal plassere n tårn på brettet, så man har nøyaktig én brikke i hver rad og én i hver kolonne. (Evt. se grundigere forklaring på side 7.) Din venn Klokland har funnet en løsning på dette problemet, for én bestemt n – altså en bestemt plassering av de n brikkene. Din venn Smartnes vil vite hvilken av de mulige løsningene Klokland har funnet, men det vil han ikke fortelle henne. I stedet får hun prøve å gjette, ved å stille ja/nei-spørsmål. Hvor mange spørsmål trenger hun, i verste tilfelle, hvis du *ikke* kan anta noe om hvor smart Smartnes er? Oppgi svaret i asymptotisk notasjon. Forklar svaret.

$\Omega(n \lg n)$. Vi kan ikke anta noe om hvor smart hun er og kan ikke angi noen øvre grense for antall spørsmål, og kan dermed ikke bruke O eller Θ. Den nedre grensen kan ikke settes høyere enn det minimale antall spørsmål, siden vi heller ikke her kan anta noe om hvor smart hun er.

Akkurat som for sorteringsgrensen har vi $n!$ mulige løsninger (n posisjoner for den første $\times$ $n-1$ mulige for den andre, etc.), og i verste tilfelle kan vi ikke garantere noe bedre enn $\Omega(\lg_2 n!) = \Omega(n \lg n)$.

Merk: Denne oppgaven har vist seg å være vanskeligere enn antatt, og tas dermed ut av sensur der det gir økt reskalert poengsum.

For dem som ikke så koblingen til hardhetsbevis, og antall ja-nei-spørsmål for å skille $n!$ alternativer i verste tilfelle (som diskutert i detalj pensumdelkapittel 8.1) så fremsto det som forvirrende at man ikke kunne anta noe om hvor smart Smartnes er. Flere prøvde dermed tross alt å gi øvre grenser for hvor dårlig hun kunne gjøre det. Noen fikk også opplyst at Smartnes har hukommelse, noe som gjør at det ikke er helt urimelig at hun i hvert fall kan klare å spørre seg gjennom alle mulige konfigurasjoner, én etter én, som gir et antall på $O(n!)$, som vil gi en viss uttelling. Dersom man har oppgitt begge grensene ($\Omega(n \lg n)$ og $O(n!)$) gis det full uttelling. Om man har tolket tårnene som unike, og endt med $O(n!^2)$, så aksepteres det på lik linje med $O(n!)$.

Relevant læringsmål: Forstå hvorfor *sammenligningsbasert sortering* har en *worst-case* på $\Omega(n \lg n)$

5% 8. I 0-1-ryggsekkproblemet, anta at du har n gjenstander og en ryggsekk-kapasitet på m. Hva blir kjøretiden til pensumløsningen?

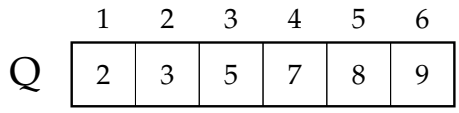
$\Theta(nm)$

Her har enkelte tolket m som antall bits i kapasiteten W, og fått $\Theta(n2^m)$. Det vil også gi en viss uttelling. $O(nm)$ er mindre presist, men poenget her var ikke skillet mellom O og Θ, så det gis full uttelling. $O(nW)$ og $\Theta(nW)$ gis også full uttelling, selv om det ikke direkte besvarer oppgaven.

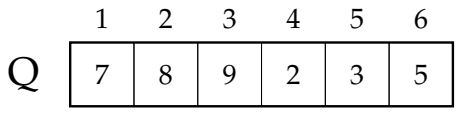
Relevant læringsmål: Forstå løsningen på *0-1-ryggsekkproblemet* (KNAPSACK, KNAPSACK′)

5% 9. Om du setter tallene $1, \ldots, n^3$ inn i et binært søketre i tilfeldig rekkefølge, der $n = 2^k - 1$ for et heltall $k \geq 1$, hva er den forventede høyden til treet som funksjon av k? Oppgi svaret i Θ-notasjon.

$\Theta(k)$



Figur 1: Eksempel-kø til oppgave 11. Her er verdiene fylt inn fra starten, så Q.*head* = Q.*tail* = 1



Figur 2: Eksempel-kø til oppgave 11. Her er verdiene *ikke* fylt inn fra starten. Q.*head* = Q.*tail* = 4

Den forventede høyden er $\Theta(\lg n^3) = \Theta(3 \lg n) = \Theta(\lg n) = \Theta(\lg(2^k - 1)) = \Theta(\lg 2^k) = \Theta(k)$.

Her er det tvetydig hvilken tallfølge det er snakk om, dvs., om det er $1^3, 2^3, 3^3, \ldots, (n-1)^3$, n^3 eller $1, 2, 3, \ldots (n^3 - 1), n^3$, men svaret blir det samme i begge tilfeller.

Relevant læringsmål: Vite at forventet høyde for et tilfeldig binært søketre er $\Theta(\lg n)$. (Være godt kjent med logaritmer med ulike grunntall. Kunne definere *asymptotisk notasjon*, O, Ω, Θ, o og ω.)

5% 10. Din venn Smartnes utfører DFS på en DAG. Hvilke typer kanter kan hun få?

(Her refererer *typer* til DFS sin kantklassifisering.)

Tre-kanter (*tree edges*), fremover-kanter (*forward edges*) og kryss-kanter (*cross edges*).

Relevant læringsmål: Forstå hvordan DFS *klassifiserer kanter*

5% 11. Din venn Gløgsund har satt inn n ulike tall i stigende rekkefølge i en FIFO-kø Q, implementert som en tabell. Køen er full, så tabellen inneholder kun elementene hun har satt inn. Elementene utgjør altså enten ett stigende segment, $\langle x_1, \ldots, x_n \rangle$ (se figur 1) eller to stigende segmenter, $\langle x_k, \ldots, x_n, x_1, \ldots x_{k-1} \rangle$ (se figur 2), der $x_i < x_{i+1}$ for $i = 1 \ldots n-1$. Dessverre har Gløgsund glemt hvor køen starter og slutter (dvs., Q.*head* og Q.*tail*). Beskriv en algoritme som lar henne finne ut dette, så effektivt som mulig. Hva blir kjøretiden?

(Her holder det at du gir en svært overordnet beskrivelse, uten å ta hensyn til praktiske detaljer eller spesialtilfeller.)

$\langle x_i, \ldots, x_j \rangle$ inneholder splittpunktet når $x_i > x_j$. Binærsøk basert på dette. $T(n) = \Theta(\lg n)$.

Hvis sekvensen ikke inneholder splittpunktet er situasjonen som i figur 1. Ellers deler vi sekvensen på midten, utfører testen på ett av intervallene, og søker så rekursivt i den relevante halvdelen.

Her har noen benyttet seg av kø-operasjoner for å løse problemet, f.eks. ved å bruke av DEQUEUE etterfulgt av ENQUEUE med samme verdi, for å avgjøre hvilken verdi som utgjør hodet i køen, og deretter bruke vanlig binærsøk etter den. Disse operasjonene forutsetter at man kjenner Q.*head* og Q.*tail*, så svaret er ikke helt ideelt, men gis likevel en viss uttelling. En slik løsning kan også terminere tidlig, så kjøretiden $O(\lg n)$ aksepteres også i det tilfellet.

Relevant læringsmål: Forstå designmetoden *divide-and-conquer* (*splitt og hersk*). (Forstå hvordan *stakker* og *køer* fungerer (STACK-EMPTY, PUSH, POP, ENQUEUE, DEQUEUE). Forstå BISECT.)

5% 12. I hver iterasjon av den ytterste løkka til INSERTION-SORT skal man utføre en operasjon på $A[1 \ldots j-1]$ og $A[j]$. Hva er denne operasjonen, og hva er kjøretiden som funksjon av j? Bruk O-notasjon.

Man skal sette $A[j]$ på rett sted i sortert rekkefølge i $A[1 \ldots j-1]$. Kjøretiden er $O(j)$.

Dette er altså operasjonen som beskrevet av kommentaren på linje 3 i pseudokoden («Insert $A[j]$ into the sorted sequence $A[1 \ldots j-1]$»), og som diskuteres ifm. løkkeinvarianten på s. 18–19. Her har det vært flere tolkninger av hvilken operasjon det var snakk om; om det f.eks. var tilordningen $A[i+1] = key$, med kjøretid $O(1)$. Dette er ikke et fullgodt svar, men vi gi en viss uttelling.

I den engelske oppgaveteksten var det oppgitt $A[1 \ldots n-1]$ i stedet for $A[1 \ldots j-1]$. Det utvises dermed noe fleksibilitet overfor besvarelser på denne utgaven som benytter n, selv om man er bedt om å uttrykke svaret som funksjon av j.

Relevante læringsmål: Forstå INSERTION-SORT

W

	1	2	3	4
1	0	1	5	6
2	∞	0	−3	4
3	∞	∞	0	2
4	∞	∞	∞	0

Figur 3: Vektmatrisen til G, brukt i oppgave 14

$T^{(1)}$

	1	2	3	4
1	1	0	1	0
2	1	1	1	0
3	0	0	1	0
4	0	1	0	1

Figur 4: Forrige tilstand, brukt i oppgave 15

5% 13. Din venn Lurvik har utviklet en ny prioritetskø som han er veldig stolt av. Den kan konstrueres i lineær tid, akkurat som en binærhaug (*binary heap*), mens EXTRACT-MIN og DECREASE-KEY har kjøretid på henholdsvis $O(n^2)$ og $O(n^3)$, der n er antall elementer i køen. Hva blir kjøretiden til DIJKSTRA om man bruker Lurviks prioritetskø? Begrunn svaret kort.

(Merk at Lurvik her *ikke* bruker en serie med kall til INSERT for å bygge køen sin, men bygger den i lineær tid i starten av DIJKSTRA.)

Bygging utføres én gang, EXTRACT-MIN én gang per node og DECREASE-KEY inntil én gang per kant. Kjøretiden blir dermed $O(V) + V \times O(V^2) + E \times O(V^3) = O(V^3 + EV^3) = O(EV^3)$.

Her er det flere som har brukt n som parameter i kjøretiden, noe ikke gir helt mening, siden n er oppgitt å være antall elementer i køen, noe som ikke er en del av probleminstansen. Det er likevel ikke urimelig å tolke det dithen at man har ment $n = |V|$, siden man tidlig i DIJKSTRA har tilordningen $Q = V$, så $O(En^3)$ gis full uttelling. Om man spesifiserer at $|E| = O(n^2)$ og at $T(n) = O(n^5)$ gis det også full uttelling.

Relevant læringsmål: Forstå DIJKSTRA. Analysere algoritmers korrekthet og effektivitet

5% 14. La G være en vektet graf, definert av vektmatrisen i figur 3. Utfør DAG-SHORTEST-PATH på grafen, med 1 som startnode.

Fyll ut avstandene til hver node etter hver iterasjon i hver rad i tabellen nedenfor.

	1	2	3	4
0	0	∞	∞	∞
1	0	1	5	6
2	0	1	−2	5
3	0	1	−2	0
4	0	1	−2	0

Relevant læringsmål: Forstå DAG-SHORTEST-PATH

5% 15. Du har utført én iterasjon av TRANSITIVE-CLOSURE, og endt opp med tabellen $T^{(1)}$ som vist i figur 4. Utfør neste iterasjon, og fyll inn de resulterende verdiene i tabellen nedenfor.

$T^{(2)}$

	1	2	3	4
1	1	0	1	0
2	1	1	1	0
3	0	0	1	0
4	1	1	1	1

Relevant læringsmål: Forstå TRANSITIVE-CLOSURE

5% 16. Hva er kjøretiden til algoritmen ALPHA, nedenfor, der $n \geq 1$ er et heltall? Oppgi svaret i Θ-notasjon, som funksjon av n.

```
ALPHA(n)
1  for i = 1 to n
2      for j = i to n
3          for k = 1 to n
4              print "Lurvik rulz!"
```

$\Theta(n^3)$

Relevant læringsmål: Analysere algoritmers korrekthet og effektivitet

5% 17. Hva er kjøretiden til algoritmen BETA, nedenfor, der $n \geq 1$ er et heltall? Oppgi svaret i Θ-notasjon, som funksjon av n.

```
BETA(n)
1  if n ≥ 2
2      m = ⌊n/2⌋
3      BETA(m)
4      BETA(n − m)
5      for i = 1 to n^2
6          print "Smartnes rocks!"
```

$\Theta(n^2)$

Relevant læringsmål: Analysere algoritmers korrekthet og effektivitet

5% 18. Hvilket problem løser algoritmen DELTA, nedenfor, der $n, m \geq 0$ er heltall?

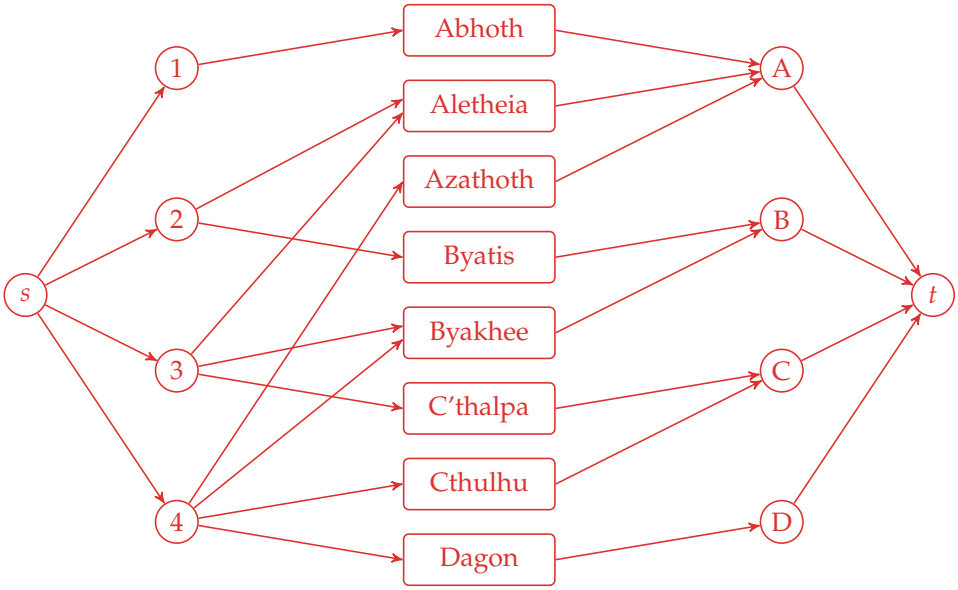
```
GAMMA(n, m)
1  if m == 0
2      return n
3  else return GAMMA(n, m − 1) + 1

DELTA(n, m)
1  if m == 0
2      return 0
3  else return GAMMA(n, DELTA(n, m − 1))
```

Multiplikasjon, dvs., $\text{DELTA}(n, m) = n \times m$.

Relevant læringsmål: Analysere algoritmers korrekthet og effektivitet

5% 19. Din venn Kvikstad skal gi navn til et sett med datamaskiner. Han vil at navnene skal begynne med ulike bokstaver, med ett navn på A, ett på B, osv., så langt han kommer. Han har ikke bestemt seg for hvilken maskin som skal få navn som begynner med hvilken bokstav, men han har noen navneforslag for hver maskin, og vil velge fra dem. Tegn opp et flytnett som finner et gyldig utvalg for ham, basert på følgende tabell:

Maskin	**Mulige navn**
1	Abhoth
2	Aletheia, Byatis
3	Aletheia, Byakhee, C'thalpa
4	Azathoth, Byakhee, Cthulhu, Dagon



Her er det noen svarvarianter som går igjen. Den første er å finne en generell løsning, og å bruke den på den oppgitte instansen, som vist i løsningen over. Den andre er å *nesten* løse problemet, ved å bruke flyt til å finne en bipartitt matching mellom maskiner og forbokstaver, og så velge vilkårlig navn fra gitt forbokstav for hver maskin. I dette tilfellet vil altså flytnettet ikke gi en gyldig tilordning av navn, men er en sentral komponent i løsningen. Dette vil gi en viss uttelling. Den siste kategorien baserer seg på det faktum at det kun finnes én løsning på den oppgitte instansen; man setter opp bipartitt matching mellom maskin og forbokstav, men angir ikke hvordan man skal velge navn, siden dette er éntydig gitt. Dette er jo også éntydig gitt selv *uten* en bipartitt match, så bidrar flytnettet i liten grad til en løsning. Dette vil likevel gi noe uttelling.

Relevant læringsmål: Formulere problemer så de kan løses av algoritmer; forstå Ford-Fulkerson

5% 20. En springer er plassert i rute a1 på et $n \times n$-sjakkbrett og skal flyttes k ganger, eller til et slikt trekk ville havnet utenfor brettet. Hver gang skal ett av de 8 mulige trekkene velges tilfeldig. Beskriv en algoritme basert på dynamisk programmering som beregner sannsynligheten for at springeren fortsatt er på brettet etter k trekk.

(Evt. se grundigere forklaring på neste side.)

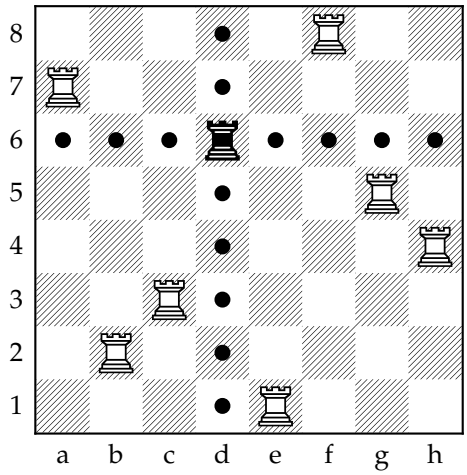
$P(x, y, k)$: Sannsynlighet om vi starter i rad og kolonne x og y, og skal flytte k ganger.

$P(x, y, 0) = 1$ for alle x og y, der $1 \leq x, y \leq n$.

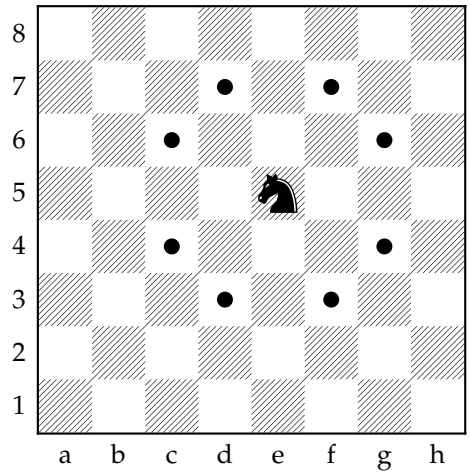
$P(x, y, i) = \frac{1}{8} \cdot$ (summen av $P(u, v, i-1)$ hvis vi kan flytte fra (x, y) til rute (u, v))

Svaret blir $P(1, 1, k)$. Funksjonen memoiseres, eller løses fra bunnen av med en trippel **for**-løkke.

Om man her har beskrevet algoritmen generelt, og ikke angitt at man finner svaret spesifikt for $P(1, 1, k)$, så gir det likevel full uttelling.



Figur 5: n-tårns-problemet. Det svarte tårnet truer rutene angitt ved svarte prikker



Figur 6: En springer kan hoppe til en av åtte ruter, som angitt ved svarte prikker

Relevante læringsmål: Konstruere nye effektive algoritmer. Forstå designmetoden *dynamisk programmering*

Om sjakkproblemene

I problemene vi ser på tillater vi vilkårlig store kvadratiske rutenett som sjakkbrett, så et $n \times n$-sjakkbrett består av n rader og n kolonner.

I n-tårns-problemet (se oppgave 7) har vi et $n \times n$-sjakkbrett og n tårn som skal plasseres på dette brettet, så ingen av dem *truer* hverandre. Et tårn *truer* alle brikker i samme rad eller kolonne, uavhengig av farge. (Se figur 5 for et eksempel.) Med andre ord går problemet ut på å plassere nøyaktig én brikke i hver rad og nøyaktig én brikke i hver kolonne. Det er naturligvis flere korrekte løsninger på dette problemet.

I springerproblemet (se oppgave 20) har vi kun én brikke på et $n \times n$-sjakkbrett. Denne brikken er en *springer*, som kan flyttes til én av (maksimalt) åtte ulike posisjoner, som vist i figur 6. Det vil si, den kan flyttes to ruter opp, ned, til høyre eller venstre, og deretter én rute til siden. For eksempel to ruter til høyre og så én rute ned, eller to ruter ned og én til høyre.

Dersom springeren står nært kanten av brettet, vil noen av disse åtte trekkene ikke lenger være lovlige, siden de havner utenfor brettet. I problemet vårt velger vi likevel blant alle åtte (med uniform sannsynlighet), og vil se om springeren havner utenfor brettet. Hvis den for eksempel står på rute a1 (nederst til venstre) og flytter to trinn opp og ett til venstre, som er et av de åtte mulige trekkene, så har den havnet utenfor og vi kan ikke lenger flytte den videre.

Vi velger hvert trekk uavhengig av de andre. Husk at sannsynligheten for at to uavhengige hendelser A og B med respektive sannsynligheter $\mathrm{P}(\mathrm{A})$ og $\mathrm{P}(\mathrm{B})$ begge skal inntreffe er produktet av sannsynlighetene, $\mathrm{P}(\mathrm{A}) \cdot \mathrm{P}(\mathrm{B})$.