OPPGAVER

for

”IMT2021 - Algoritmiske metoder”

Høsten 2017

NTNU

Forord

Dette kompendie/hefte innholder oppgaveteksten for ulike oppgaver i emnet ”Algoritmiske metoder” ved NTNU. Disse oppgavene brukes både som en del av ukeoppgavene og eksamens-forberedende oppgaver, men også som supplerende og frivillige ekstraoppgaver.

Opplysninger om emnet er å finne på Internet på URL’en:
http://folk.ntnu.no/frh/algmet

Under «Øvinger» er det angitt hvilke oppgaver i dette heftet som til enhver tid er relevante som oppgaver.

(Data)filer som oppgavetekstene henviser til er å finne under ”UKE_OPPG”.

Løsningsforslag til alle oppgavene er å finne under ”UKE_LOSN”.

NB: I tillegg til oppgavene i dette heftet, vil vi bruke mange av dem i læreboka.

NTNU
Frode Haug

Oppgave 1 (kap.4) - tidligere obligatorisk oppgave nr.1

Lag et program som inneholder koden for:

- Stack (jfr. den s.26, evt. ”til nød” den s.29-30).
- Bygging av binært tre som leser postfix aritmetisk uttrykk (s.41).
- Traversering av dette treet på en preorder måte (kode s.45).

Ekstra:

- Traversering av dette treet på en postorder måte (**hint** s.47).
  (Skal *ikke* løses vha. rekursjon, men ved bruk av stack.)

Oppgave 2 (kap.5)

Funksjonen aller øverst s.52 i læreboka regner ut 'N!' på en rekursiv måte.
Skriv en funksjon som regner ut 'N!' på en *iterativ* måte.

Oppgave 3 (kap.5)

Lag (rekursive) funksjoner som:

- finner antall (interne) noder i et binært tre.
- antall external-noder i et binært tre.
- antall fulle noder i et binært tre (som har *to* internal-noder).
- finner treets høyde

Alle funksjonene tar en node-peker (root) som input-parameter. Funksjonene skal enten returnere med en int som svar, eller at de oppdaterer en global variabel.

Starthjelp:

Bruk koden fra OPPG_01.CPP til å generere et binært tre fra et postfix uttrykk (som du taster inn). Test funksjonene mot dette treet.

Oppgave 4 (kap.5)

Bruk koden fra OPPG_01.CPP til å generere et binært tre fra et postfix uttrykk (som du taster inn).

Traverser (og skriv ut) dette treet rekursivt på en preorder, inorder og postorder måte (**hint** s.60).

Traverser så dette treet på en rekursiv postorder måte, der:

- brukeren for hver bokstav i treet blir bedt om å erstatte bokstaven med et tall.
- programmet skriver ut svaret på det regnstykket som når oppstår.

Hint: Regn ut verdien til venstre subtre, og adder/multipliser dette med verdien av det høyre subtreet.

Oppgave 5 **(kap.5)**

Lag en funksjon som lager et tilfeldig binært tre. Input til funksjonen skal være max. antall nivå i treet, og prosentsatsen (10-100%) for hvor stor sannsynlighet det er for at en node skal få henholdsvis venstre og høyre subtre. Fyll de ulike nodenes key/info/ID med data (tall) på en inorder måte. Dermed får du et sortert binært søketre, der *alle* nodene/subtrærne til venstre for en node har mindre verdi, og *alle* nodene/subtrærne til høyre har høyere verdi. Funksjonen skal returnere med en peker til rota i treet.

Oppgave 6 **(kap.5)**

Hanoi's tårn:
Et spill består av tre pinner, A, B og C, og N smultring-lignende brikker, alle med forskjellig størrelse (diameter). Disse kan tres ned på pinnene. Før spillet settes igang er samtlige brikker tredd ned på pinne A, slik at enhver brikke er plassert oppå en større, med unntak av den største som ligger underst.

Oppgaven består i å flytte brikkene, en og en, slik at samtlige brikker tilslutt er plassert på pinne C. De skal da ligge i samme orden som de opprinnelig var på pinne A. Alle tre pinnene kan benyttes som midlertidig lagringsplass for brikker under flytting, men det er til enhver tid forbudt å plassere en større brikke oppå en mindre.

Lag en rekursiv funksjon som skriver ut sekvensen av trekk som man må gjøre for å løse spillet. Dvs. for hvert trekk så skriver den bare ut hvilken pinne det skal flyttes fra og hvilken det skal flyttes til.

Oppgave 7 **(kap.5)**

Anta at vi har en skoleklasse, og vet hvilke (par) av elevene som er uvenner. Finn måter å stille elevene på rekke slik at ingen som er uvenner blir stående ved siden av hverandre. For enkelhets skyld kan vi her anta at de som er uvenner er nettopp de som har nabonummer i den nummerrekkefølgen vi gav dem. (Vi kan tenke oss at de er nummerert i den rekkefølgen de *pleier* å bli stilt opp, og at alle er blitt uvenner med sine naboer her. Dvs. nr.1 er uvenn med nr.2 (og omvendt), nr.2 er også uvenn med nr.3 (og omvendt), ..., nr.N-1 er uvenn med nr.N (og omvendt).) Oppgaven går altså ut på å finne permutasjoner av N stk, der ingen tall står ved siden av tall som er nøyaktig *en* høyere eller *en* mindre enn seg selv.

Oppgave 8 (kap.5) - tidligere obligatorisk oppgave nr.2

Innledning:
Av og til kommer man over oppgaver (i aviser, blader o.l.) som går ut på å endre rekkefølgen på bokstaver, slik at man får et meningsfullt ord, f.eks. stedsnavn eller et yrke. "BERGEN" kunne ha vært skrevet som "GEERBN", eller "HEISMONTØR" som "TØRMISNEHO".

Oppgaven:
Denne oppgaven går ut på å lese inn en tekst (*max.15 tegn*), og generere alle kombinasjoner/ permutasjoner av disse bokstavene.

En tekst på N bokstaver, vil jo som kjent ha N! ulike permutasjoner. For å sile vekk (avskjære) *noen* av alternativene så skal programmet *ikke* skrive ut ord der:

1) a: to *like* vokaler kommer etter hverandre.
   b: tre vokaler kommer etter hverandre.
   c: tre *like* konsonanter kommer etter hverandre.
   d: fire konsonanter kommer etter hverandre.
   e: to like bokstaver (vokaler eller konsonanter) starter et ord.

2) ett tegn havner i en posisjon der et identisk tegn allerede har vært.
   (F.eks. i ordet "BIRI" vil ombytte av 'I'ene være uinteressant.)

Output fra programmet *skal* være de permuterte ordene:
- Nummerert fra 1 og fortløpende oppover.
- Stans for hver 24.utskrift (og vent på at ett tegn blir skrevet/tastet inn)

Eksempel:
Ordet "SOLO" har egentlig 24 permutasjoner. Med avskjæringene ovenfor (pkt.1 og 2), så vil *kun* 6 permutasjoner bli skrevet ut. Disse er merket med '*' nedenfor. Forklaring på hvorfor de andre ikke skrives ut er å finne i høyre kolonne:

```
* SOLO
  SOOL  |
  SLOO  | Avskjæring 1a:     To like vokaler etter hverandre
  SLOO  |
  SOOL  |
  SOLO    Avskjæring 2
* OSLO
* OSOL
* OLSO
* OLOS
  OOSL    Avskjæring 1e
  OOLS    Avskjæring 1e
  LSOO    Avskjæring 1a
  LSOO    Avskjæring 1a
* LOSO
  LOOS    Avskjæring 1a
  LOSO    Avskjæring 2
  LOOS    Avskjæring 1a
```

OSOL		
OSLO	Avskjæring 2:	'O' har allerede vært i 1.posisjon, og disse
OOSL		permutasjonene er allerede generert.
OOLS		
OLSO		
OLOS		

Hint / tips / fremgangsmåte:
Følgende måte å utvikle programmet på tilrådes:

A) Bygg på EKS_06.CPP (denne *skal* brukes). Endre denne slik at int(-arrayen) hele veien blir erstattet med char(-array). Sørg for at innholdet i arrayen blir lest inn fra tastaturet. Upcase alle bokstavene, også 'æ', 'ø' og 'å'.

B) Endre funksjonen 'display' slik at den ivaretar kravene til output.

C) Lag en funksjon 'navnOK(char a[], int len)' som sjekker at teksten "a" i de "len" første posisjonene tilfredsstiller kravene i gitt i 1). Denne kalles fra "passelige" *steder* inni funksjonen 'perm'. Funksjonen returnerer med '1' dersom navnet er OK, ellers '0'.

D) Lag en funksjon som sjekker etter om en bokstav allerede har vært i en posisjon, og som kalles et "passende" *sted* fra 'perm'. (Parametre og returverdi får du selv finne ut.)

Testing:
Dersom programmet ditt fungerer *helt* korrekt med *alle* avskjæringer, så skal følgende ord gi antall output som beskrevet i høyre kolonne:

OSLO	6	ULLA	10	
SOLO	6	ULLU	3	
LENA	24	ABBA	3	
TOTEN	54	AABBA	1	
HAMAR	36	AABBAB	4	
GJØVIK	576	MISSISSIPPI	4673	(av 39.916.800 mulige!)

Oppgave 9 (kap.5)

Lag et program (med en rekursiv funksjon) som forsøker å plassere N sjakkdronninger på et NxN brett. Dvs. to dronninger kan hverken stå på samme linje (i), kolonne (j) eller diagonal, ellers "slår/tar" de hverandre.

Hint: En rekursiv algoritme bør prøve å plassere en dronning i alle lovlige posisjoner på kolonne nr.'j'. (For-løkke som prøver å plassere på linje for linje.) Skal dette lykkes, må den kontrollere at ingen andre dronninger står hverken på samme linje eller på en diagonal i noen av de tidligere kolonnene. Deretter gjør den kall til at det samme skal gjøres for kolonne nr.'j+1'. Når brettet er fullt (dvs. funnet plass til dronning også i siste kolonne), så skrives løsningen ut.

Kontroll:

Programmet trenger *ikke* å ignorere/avskjære løsninger som egentlig bare er speilinger eller rotasjoner av tidligere fundne løsninger. Derfor skal programmet (for kontrollens skyld) for et N x N brett finne følgende antall løsninger:

4: 2 stk, 5: 10 stk, 6: 4 stk, 7: 40 stk, 8: 92 stk.

Oppgave 10 (kap.9)

Skriv inn Quicksort s.118 (evnt. kopier og skriv om EKS_08.CPP). Endre den slik at når subarrayen blir mindre enn en viss M (f.eks. 5), så foretas en 'Insertion' på denne delen av arrayen (jfr. aller nederst s.124 i læreboka). Du må også skrive om 'Insertion' s.100 i læreboka, slik at den kun virker på en del av en array (mellom 'l' og 'r'). Hvordan vil du løse problemet med 'sentinel key'? Legg til slutt inn at partisjonselementet velges og behandles som etter "median-of-three" forbedringen (jfr. andre avsnitt s.126 i læreboka).
('The three-exchange method' det henvises til er 'sort3' s.95 i læreboka.)

Oppgave 11 (kap.5)

Vi skal lage et program som finner en måte å bevege en hest/springer rundt på et sjakkbrett, slik at den er innom *alle* ruter *en* og *bare en* gang. En hest kan gå *to* skritt *fram* i vilkårlig retning (vannrett og loddrett) og *ett* skritt *til siden* (eller omvendt, dvs. ett frem og to til siden). Oppgaven løses for et N*N brett (brukeren angir 'N'), og hesten skal starte i en gitt posisjon (som leses fra tastaturet).

Hint: Lag først en rekursiv funksjon som genererer *alle* "hesteturer" ut fra en gitt posisjon. Legg så inn avskjæringer som:

- holder den innenfor brettet.
- sørger for at ingen rute besøkes mer enn en gang.

Kontroll:

For et N*N brett, når startposisjon er øvre venstre hjørne (1,1), og neste trekk er *to* til høyre og *en* ned, så fås følgende antall løsninger:

1: 1, 2: 0, 3: 0, 4: 0, 5: 152, 6: 262243(!)

(for brett *større enn* 6 så begynner dette å bli "håpløst").

Oppgave 12 (kap.5)

Vi skal fargelegge landene på et kart, slik at land som har felles grense ikke har samme farge. Dette lar seg *alltid* gjøre med max. fire farger. Vi har N land (nummerert fra 1 til N), og hvilke land som har felles grenser er gitt ved den "boolske" arrayen "felles[N][N]", som er '1' for de par (i,j og j,i) som har felles grense, ellers '0'. For "Europa" er denne ferdig laget, og ligger på filen: OPPG_12.TPL.

Vi skal lete opp mulig fargelegging, som skal produseres i en int array "farge[N]". Fargene er nummerert fra 1 til 4. Lag først en rekursiv funksjon som genererer alle mulige fargeleggінger, og legg så inn avskjæring slik at bare lovlige fargeleggінger blir generert.

(Alternativ oppgavetekst: se Kontinuasjonseksamen august 1994, nr.3.)

Oppgave 13 (kap.11)

Skriv inn klassen s.147 (unntatt 'Insert' og 'Remove') og legg inn tilleggskode også fra sidene 150, 152 og 153. Skriv selv funksjoner som tar seg av 'Delete/Extract' og 'Change' (jfr. avsnittet *rett over* 'Property 11.1' s.153). Test funksjonene og lag utskrift etterhvert.

Oppgave 14 (kap.8-9 og 11-12):

NB: Denne oppgaven løses enklest ved bruk av "dos.h" (da WATCH.H bruker den).

Filen OPPG_14.DTA inneholder 5000 tilfeldige tall i intervallet 1-30000. Les disse inn i en array og sorter dem med en eller flere av metodene i boka (selection, insertion, bubble sort, shellsort, quicksort, heapsort og/eller mergesort). Finn ut hvor lang tid hver av metodene bruker.

Hjelp: Fila WATCH.H inneholder kode for å start/stanse klokke, og for å se tidsdifferansen mellom disse. For å bruke denne, gjør følgende:

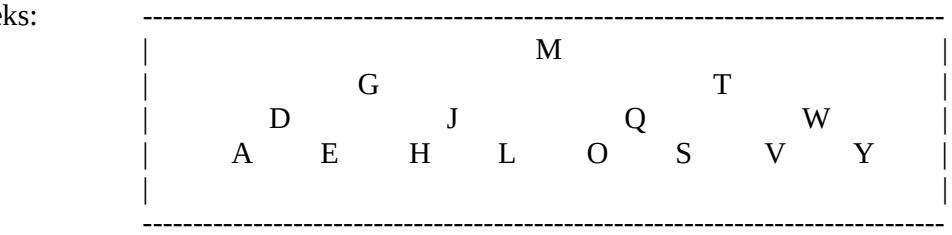
- Kopier WATCH.H til katalogen der programmet ditt ligger.
- Inkluder denne med: #include "watch.h"
- Lag et 'klokke-objekt' ved: Watch <variabelnavn>;
- Bruk klokka ved: <variabelnavn>.<funksjon>;
  der <funksjon> er: start(), stop() eller usedTime().

Oppgave 15 (kap.14)

NB: Denne oppgaven løses enklest ved bruk av "conio.h" og dens funksjon"gotoxy".

Skriv inn koden s.204, 206 og 210. Lag deretter en funksjon som skriver treet (nodenes key) på skjermen. Rota skrives midt på øvre linje, dens "barn" på neste linje, midt mellom der rota ble skrevet og høyre/venstre kant. Disses "barn" på neste linje midt .....

F.eks:



Lag et hovedprogram der du får testet ut de fire funksjonene.

Oppgave 16 (kap.15)

NB: Denne oppgaven løses enklest ved bruk av "conio.h" og dens funksjon"gotoxy".

a) Endre OPPG_15.CPP og BINTREE.H slik at den fungerer for et balansert tre istedet. De største endringene du må gjøre er:
   - Legge inn Red-Black "bit" i 'Node'.
   - Erstatte 'Insert' med ny kode s.221, 225 og 226.
   - Stryke 'Remove'.

   ('Search' trengs ikke videre(?), og 'Display' blir temmelig lik.)

b) Generer noen tilfeldige trær (ved å taste inn input til programmet i pkt.a). Sammenlign trærnes utseende med den samme inputen gitt til programmet OPPG_15.CPP. (Jfr. exercise nr.6 s.230 i læreboka.)

c) Generer et (eller flere) tilfeldig(e) 1000-noders tre. Endre koden fra pkt.a), slik at du enkelt kan få ut:
   1) antall rotasjoner som blir foretatt.
   2) gjennomsnittlige, lengste og korteste vei fra root til external ('z') noder.

   Hva sier disse resultatene? (Jfr. exercise nr.7 s.230 i læreboka.)

Oppgave 17 (kap.15) - tidligere obligatorisk oppgave nr.4

Innledning:
Noen varme og svette sommerdager så har det sittet en person fra Statistisk Sentralbyrå på den (heldigvis) forhenværende bomstasjonen ved Mjøsbrua. For alle bilene som har passert (i begge retninger), så har vedkommende registrert de to bokstavene i bilskiltene.
Tilsammen har hun/han notert seg 29312 slike to-bokstavs-sekvenser.
Alle disse ligger på filen "OPPG_17.DTA" (på katalogen: «uke_oppg»).

Oppgaven går ut på å:
- lese inn to og to bokstaver fra filen "OPPG_17.DTA" (til det er slutt).
- registrere forekomsten av de ulike bokstavkombinasjonene.
  Dvs. f.eks. hvor mange 'HS', 'JC' og 'FS' biler har passert.
- skrive antall forekomster av hver bokstavkombinasjon til "OPPG_17.RES".

Obligatorisk datastruktur:
Oppgaven *skal* løses vha. følgende datastruktur:
- Lag et binært søketre som inneholder den første bokstaven av de to.
  (Dette treet kalles heretter "1.tre".)
- For hver av nodene i det "1.treet", så er det en peker til et tre som består av bokstavene som forekommer på den andre plassen.
  (Dette treet kalles heretter "2.tre".)
- Inni hver av nodene i det "2.treet" så er det en heltallsvariabel som forteller om antall forekomster av to-bokstavs-sekvensen som vedkommende node i det "2.treet" representerer, kombinert med key'en til noden i det "1.treet" som peker til *dette* treet.

Hint / tips:
- Modifiserte utgaver av koden s.204 og 206 skulle klare mye av jobben.
- Nodene i "1.tre" og alle "2.trærne" blir forholdsvis like, bare at den ene har en peker til det tre, mens den andre har antall forekomster. Dette kan *bl.a.* løses/programmeres vha. "union" i C++.
- Skulle de føle for det, og sålenge vi *kun* legger noder inn, så kan du også bruke koden s.221+225+226 for å holde trærne balansert.

Sjekk:
Summen av alle to-bokstavs-forekomstene (når du skriver til fil) skal (selvagt) være 29312.

Oppgave 18 (kap.29)

Utvid programmet 'EKS_12.CPP' slik at:
- når kortest vei mellom to noder er funnet
- så traverseres grafen enda en gang og
- når man finner den første stien som har denne lengden så
- trekker man seg (baklengs) ut av rekursjonen igjen og
- key'en til nodene på veien tilbake skrives ut.