Høgskolen i Gjøvik
Institutt for informatikk og medieteknikk

E K S A M E N

FAGNAVN: **Algoritmiske metoder**

FAGNUMMER: **IMT2021**

EKSAMENSDATO: **8. desember 2005**

KLASSE(R): **04HBIND* / 04HBINFA / div. andre**

TID: **09.00-14.00**

FAGLÆRER: **Frode Haug**

ANTALL SIDER UTLEVERT: **5 (inkludert denne forside)**

TILLATTE HJELPEMIDLER: **Alle trykte og skrevne.**

- Kontroller at alle oppgavearkene er til stede.
- Innføring med penn, eventuelt trykkblyant som gir gjennomslag. Pass på så du ikke skriver på mer enn ett innføringsark om gangen (da det blir uleselige gjennomslag når flere ark ligger oppå hverandre).
- Ved innlevering skilles hvit og gul besvarelse, som legges i hvert sitt omslag.
- Oppgavetekst, kladd og blå kopi beholder kandidaten til klagefristen er over.
- Ikke skriv noe av din besvarelse på oppgavearkene. Men, i oppgavetekst der du skal fylle ut svar i tegning/tabell/kurve, skal selvsagt dette innleveres sammen med hvit besvarelse.
- Husk kandidatnummer på alle ark. Ta vare på dette nummeret til sensuren faller.

Oppgave 1 (teori, 25%)

Denne oppgaven inneholder tre uavhengige oppgaver fra kap. 3, 9 og 16 i læreboka.

a) Koden midt på side 27 i læreboka leser et postfix-uttrykk og regner ut svaret.
Vi har postfix-uttrykket: `5 2 * 2 3 * + 4 * 2 5 * 3 2 * + +` **Hva blir svaret ?**
Tegn opp innholdet på stakken etter hvert som koden side 27 foretar beregningen.

b) Du skal utføre Quicksort på teksten ”H I G H B U R Y” (blanke regnes ikke med).
Lag en figur tilsvarende fig. 9.3 side 119 i læreboka, der du for hver rekursive sortering skriver de involverte bokstavene og markerer/uthever hva som er partisjonselementet.

c) Vi har hash-funksjonen: `hash(k) = k mod 17`
der `k` står for bokstavens nummer i alfabetet (1-29). Vi har også en array med indeksene 0-16. **Tegn opp arrayen hver gang key’ene ”EMIRATES STADIUM” (i denne rekkefølge fra venstre til høyre, blank regnes ikke med) legges inn i den vha. linear probing.**

Oppgave 2 (teori, 25%)

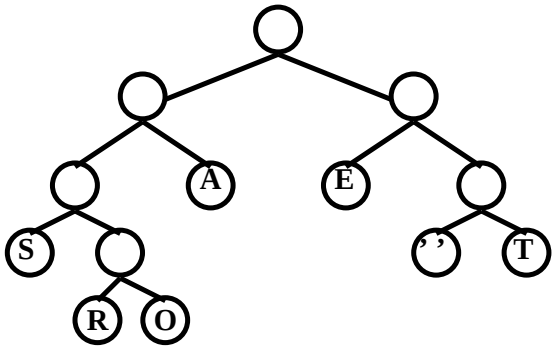
Denne oppgaven inneholder tre uavhengige oppgaver fra kap. 30 og 22 i læreboka og om FSM.

a) Følgende kanter i en (ikke-retted, ikke-vekted) graf er gitt:

BC DA FG CA GA ED AF GD

Vi skal nå utføre union-find m/path compression og weight balancing på denne grafen.
Tegn opp arrayen ”dad”s innhold etterhvert som skogen bygges opp (jfr. fig.30.9) vha. ”find”-funksjonen s.447 i læreboka. Ut fra dette: **tegn også opp den resulterende union-find skogen** (dvs. noe lignende til nedre høyre skog i fig.30.8).

b) Vi har følgende ferdiglagde Huffmans kodingstrie (dad’enes nr/indeks er uinteressant):



Lag en oversikt/tabell for bokstavenes bitmønster og ”len”.
Vi har også følgende bitstrøm, som er kodet etter denne trien:
000111010010111110100010110000111010010000110001101001101
Hva er teksten i denne meldingen?

c) Følgende deterministiske endelige tilstandsmaskin (FSM) er gitt:

$M = (\{q_0, q_1, q_2, q_3\}, \{ L, D, B \}, \delta, q_0, \{ q_1, q_2 \})$

Transisjonstabellen:

$\delta(q_0, L) = q_1$	$\delta(q_0, D) = q_2$	$\delta(q_0, B) = q_0$
$\delta(q_1, L) = q_1$	$\delta(q_1, D) = q_3$	$\delta(q_1, B) = q_1$
$\delta(q_2, L) = q_3$	$\delta(q_2, D) = q_2$	$\delta(q_2, B) = q_2$
$\delta(q_3, L) = q_3$	$\delta(q_3, D) = q_3$	$\delta(q_3, B) = q_3$

"L" står for "Letter", dvs. en av alfabetets bokstaver (A-Å eller a-å).
"D" står for "Digit", dvs. ett av sifrene 0-9.
"B" står for "Blank" (space/mellomrom).

Tegn tilstandsmaskinen.
Hva (slags setninger/språk) godtar den ?

Oppgave 3 (koding, 30%)

I denne oppgaven skal vi fokusere på binære trær, der hver node har en *unik* ID/key/nøkkel/navn.
Får vi oppgitt pre- og inorder rekkefølgen for nodene i et slikt treet kan vi *alltid* entydig tegne/generere treet. Det samme er også sant om vi får oppgitt in- og postorder rekkefølgen.
I denne oppgaven skal du **lage *to* ulike og *uavhengige rekursive* funksjoner**

```
node* pre_in (int pre[],  int & n, int in[], int l, int r);
node* post_in(int post[], int & n, int in[], int l, int r);
```

som bygger et binært tre ut fra om pre- og inorder arrayen er gitt (`pre_in(…)`), eller at in- og postorder arrayen er gitt (`post_in(…)`). Parametrene til funksjonene ovenfor er som følger: `pre`, `in` og `post` inneholder fra indeks nr.0 og oppover id'ene til nodene når de er traversert på den aktuelle måten. `n` angir siste indeks i `pre` / `post` som ble visitert. `l` og `r` angir henholdsvis venstre og høyre grense for aktuelt intervall av `in`.
Hvordan de tre arrayene har blitt fylt med sine verdier trenger du ikke å tenke på i denne oppgaven.

En node er definert som:

```
struct node {
   int   ID;      //  Nodens ID/key/nøkkel/navn (et tall).
   node* left;    //  Peker til venstre subtre, evt. z-noden om det er tomt.
   node* right;   //  Peker til høyre subtre, evt. z-noden om det er tomt.
};
```

Vi har de globale variablene:

```
node* z = new node;
node* root = z;
int ant;          //  Totalt antall noder i treet.
int nr;           //  Kommet til indeks nr.'nr' i pre-/post-array.
                  //  Referanseoverføres mellom de rekursive kallene som 'n'.
const int N = 1000;     // Sikkert nok for de fleste trær vi gidder å lage ……
int preorder[N];  //  Arrayene er fylt med verdier i indeksene 0 – 'ant'-1.
int inorder[N];   //     De sendes initielt fra 'main' til de to funksjonene,
int postorder[N]; //     som igjen sender de videre i sine rekursive kall.
```

Kallene som starter de to uavhengige funksjonene fra `main` er som følger:

```
nr = -1;    root = pre_in(preorder, nr, inorder, 0, ant-1);
nr = ant;   root = post_in(postorder, nr, inorder, 0, ant-1);
```

Vi har også den ferdigdefinerte funksjonen (og som du fritt kan kalle/bruke når du trenger den):

```
int finn(int a[], int n, const int MAX)  { //  Returnerer indeksen for 'n' i
   for (int i = 0;  i < MAX;  i++)         //    arrayen 'a'. Brukes for å
        if (a[i] == n)  return i;          //    finne en ID i inorder-arrayen
}
```

Hint: Det kan sikkert være lurt å tegne opp et eksempel på et slikt tre for deg selv, og ut fra dette skrive opp hva som blir innholdet i de tre ulike arrayene. Slik *kan* det bli enklere å forstå systematikken i problemstillingene, og dermed enklere å skjønne hvordan dette bør kodes.

NB: Funksjonene bør bli *veldig* like, så "veien" er ikke være så lang til den andre når du har laget/funnet den første.

Oppgave 4 (koding, 20%)

En populær hjernetrimoppgave er for tiden SUDOKU. Utgangspunkt for en slik *kan* f.eks. være:

		8	9					5
	7					1	8	9
	3		1	7				6
9			8		6			4
8				5	3		9	
5	8	4					1	
6					5	4		

Oppgaven går kort og godt ut på å fylle de tomme feltene slik at *alle* rader/linjer, kolonner og 3x3 bokser (de ni rutene omsluttet av tykkere streker) hver inneholder *alle* tallene fra 1-9 *en* gang.

Lag den *rekursive* funksjonen `void finn_losning(int n)` som finner *alle* løsninger på en slik oppgave. Til dette *skal* det brukes arrayen: `int s[9][9];` (Vi bruker altså indeksene 0-8.)

Du kan forutsette at når funksjonen starter (kalles fra `main` med `finn_losning(0)`), så ligger de initielle verdiene for rutenettet (lest inn fra fil) allerede i `s`. For hver løsning du finner, så skrives `s` ut på skjermen ved å kalle på `skriv_losning()`.

NB: Slike SUDOKUer er av ulike vanskelighetsgrader. Eksamensoppgave nr.4 fra 9.august 2005 løser bare brett av den enkle typen. Dvs. når det til enhver tid er *minst en* rute med en entydig verdi. Det meste av koden for dette er å finne i vedlegget. I *denne* oppgaven skal du derimot skrive kode som, vha. rekursjon og backtracking, setter inn *alle* aktuelle verdier i en rute, og dermed finner *alle* mulige løsninger. (En *ekte* SUDOKU har dog bare *en* løsning.)

Hint1: *Ett* kall til `finn_losning(int n)` skal sette inn *alle* aktuelle/lovlige verdier i rute nr. `n`. `n` går fra 0 og oppover til 81 (= 9 x 9). Aktuell rute finnes ved: `i = n/9;  j = n%9;`

Hint2: Din besvarelse *skal* inneholde en *komplett* løsning/innmat for `finn_losning(int n)`. Men, *enormt mye* av koden fra vedlegget kan brukes fortsatt. Når det er aktuelt at du foretar en slik gjenbruk, så bare henvis til de aktuelle linjenumrene (i stedet for å skrive av koden).

Lykke til! (og alle verdens SUDOKUer er løst for godt!)

frode@haugianerne.no

Vedlegg (Essensen av koden som løser en enkel SUDOKU – jfr. eksamen 9/8-05 nr.4.)

```
 1  < Div. include og ”using namespace” er utelatt. >

 2  const int ANTX = 9, ANTY = 9, STRLEN = 80, TOTSUM = 45;
 3  int ant;
 4  int s[ANTX][ANTY];

 5  < Funksjonen ”void les_fra_fil()” – som leser inn i ”s” fra fil er utelatt. >
 6  < Funksj. ”void skriv_losning()” – som skriver ”s” til skjermen er utelatt. >

 7  bool finn_losning()  {
 8    int tot = ANTX*ANTY, sum, n;
 9    int i, j, k, ii, jj, tall, dx1, dx2, dy1, dy2;
10    bool funn;

11    while (ant < tot)  {               //  Løsning ennå ikke funnet:
12      funn = false;                    //  Nytt tall IKKE funnet ennå.
13      for (i = 0;  i < ANTX;  i++)     //  Går gjennom ALLE rutene:
14        for (j = 0;  j < ANTY;  j++) {
15          if (s[i][j] == 0)  {         //  Tom/ledig rute:
16             int brukt[ANTX+1] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
17             sum = n = 0;
18             for (jj = 0;  jj < ANTY;  jj++)  {    // Tallene brukt på EN rad:
19                tall = s[i][jj];
20                if (tall && !brukt[tall]) { n++; sum += tall; brukt[tall] = 1; }
21             }
22             for (ii = 0;  ii < ANTX;  ii++) {     // Tallene brukt på EN kol.:
23                tall = s[ii][j];
24                if (tall && !brukt[tall]) { n++; sum += tall; brukt[tall] = 1; }
25             }
26                                              //  De brukt i AKTUELL 3x3 rute:
27             ii = i % 3;  jj = j % 3;         //  Hvor 'i' og 'j' er i ruten.
28             switch (ii)  {                   //  Naborutenes relative
29                case 0:  dx1 = 1;   dx2 = 2;  break;
30                case 1:  dx1 = -1;  dx2 = 1;  break;
31                case 2:  dx1 = -2;  dx2 = -1; break;
32             }
33             switch (jj)  {                   //     posisjon ift.dette:
34                case 0:  dy1 = 1;   dy2 = 2;  break;
35                case 1:  dy1 = -1;  dy2 = 1;  break;
36                case 2:  dy1 = -2;  dy2 = -1; break;
37             }
38             for (k = 1;  k <= ANTX;  k++)  { //  Går gjennom 3x3 ruten:
39                switch(k)  {                  //  Beregner absolutt ruteindeks:
40                  case 1: ii = i;      jj = j;      break;
41                  case 2: ii = i+dx1;  jj = j;      break;
42                  case 3: ii = i+dx2;  jj = j;      break;
43                  case 4: ii = i;      jj = j+dy1;  break;
44                  case 5: ii = i+dx1;  jj = j+dy1;  break;
45                  case 6: ii = i+dx2;  jj = j+dy1;  break;
46                  case 7: ii = i;      jj = j+dy2;  break;
47                  case 8: ii = i+dx1;  jj = j+dy2;  break;
48                  case 9: ii = i+dx2;  jj = j+dy2;  break;
49                }
50                tall = s[ii][jj];             //  Er tallet brukt:
51                if (tall && !brukt[tall]) { n++; sum += tall; brukt[tall] = 1; }
52             }
53             if (n == ANTX-1)  {           //  Eksakt 8 tall er brukt:
54                ant++;  funn = true;       //  Kan da sette inn det
55                s[i][j] = TOTSUM - sum;    //     nye og kjente tallet.
56      }   }  }                            //  Ingen nye tall funnet ved
57      if (!funn)  return false;           //    gjennomgang av HELE brettet:
58    }
59    return true;
60  }

61  < ”Main” – som starter det hele er utelatt. >
```