

Institutt for datateknologi og informatikk

## Eksamensoppgave i IDATG2102 – Algoritmiske metoder

Faglig kontakt under eksamen:

Frode Haug

Tlf:

950 55 636

Eksamensdato:

14.desember 2023

Eksamenstid (fra-til):

09:00-13:00 (4 timer)

Hjelpemiddelkode/Tillatte hjelpemidler:

I - Alle trykte og skrevne  
(kalkulator er *ikke* tillatt)

Annen informasjon:

Målform/språk:

Bokmål

Antall sider (inkl. forside):

4

### Informasjon om trykking av eksamensoppgaven

Originalen er:

1-sidig ☒ 2-sidig ☐

sort/hvit ☒ farger ☐

Skal ha flervalgskjema ☐

Kontrollert av:

---

Dato

Sign

## Oppgave 1 (teori, 25%)

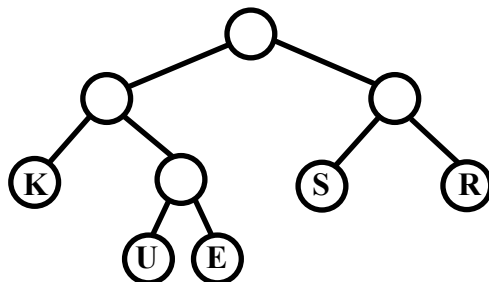
Denne oppgaven inneholder tre totalt uavhengige oppgaver fra pensum.

- a) 8 Et av eksemplene i pensum leser og omgjør et infix-uttrykk til et postfix-uttrykk. Vi har infix-uttrykket:  $((2 * 3) + ((4 * 2) + ((5 + 4) * 3)))$   
**Hva blir dette skrevet på en postfix måte?**  
**Skriv/tegn stakkens innhold etter hvert som koden leser tegnene i infix-uttrykket.**
- b) 9 Shellsort skal utføres på bokstavene «SKARSNUTEN». For hver gang indre for-løkke i eksemplet med Shellsort er ferdig (dvs. rett etter:  $a[j] = \text{verdi};$ ) :  
**Skriv/tegn opp arrayen og skriv verdiene til 'h' (4 og 1) og 'i' underveis i sorteringen.**  
**Marker spesielt de key'ene som har vært involvert i sorteringen.**
- c) 8 Følgende heap er gitt: 73 49 54 23 42 49 52 7 19 36  
Utfør etter tur følgende operasjoner på denne heap: insert(73), insert(91), remove(), remove() og replace(49). Skriv opp heapen etter at hver av operasjonene er utført. NB: For hver av operasjonene skal du operere videre på den heapen som ble resultatet av den forrige operasjonen.

## Oppgave 2 (teori, 25%)

Denne oppgaven inneholder tre totalt uavhengige oppgaver fra pensum.

- a) 8 I forbindelse med dobbelt-hashing har vi teksten «SKURSNUTEN» og de to hash-funksjonene  $\text{hash1}(k) = k \bmod 13$  og  $\text{hash2}(k) = 4 - (k \% 4)$  der  $k$  står for bokstavens nummer i alfabetet (1-29). Vi har også en array med indeksene 0 til 12.  
**Skriv hver enkelt bokstav sin k-verdi og returverdi fra både hash1 og hash2.**  
**Skriv også opp arrayen hver gang en bokstav hashes inn i den.**
- b) 9 Følgende kanter i en (ikke-retted, ikke-vekted) graf er gitt:  
FD AD DB CE EB AE  
Utfør Union-Find *m/weight balancing (WB) og path compression (PC)* på denne grafen.  
**Skriv/tegn opp innholdet i gForeldre etter hvert som unionerOgFinn2 kjøres/utføres. Bemerk hvor WB og PC er brukt.**  
**Skriv/tegn også opp den resulterende union-find skogen.**
- c) 8 Vi har følgende ferdiglagde Huffman kodingstrie (innholdet i gForeldre er uinteressant):



Vi har også følgende bitstrøm (melding), som er kodet etter denne trien:

1000010111101110 0011010011

**Hva er bokstavenes bitmønster og hva er teksten i denne meldingen?**

## Oppgave 3 (koding, 30%)

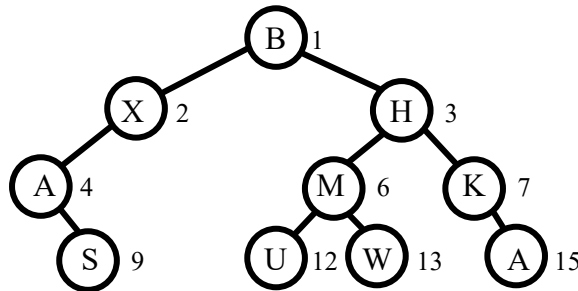
Vi har et *binært* tre (*ikke* nødvendigvis *søketre*) bestående av nodene:

```
struct Node {
    char ID;           // Nodens ID/key/nøkkel/navn (ett tegn).
    int pos;           // Posisjon i treet.
    Node *left, *right; // Referanser til begge subtrærne, evt. nullptr/NULL
    Node (char id, int p) // Constructor:
        { ID = id; pos = p; left = right = nullptr; }
};
```

Vi har også den globale variabelen:

```
Node* gRoot = nullptr; // Rot-peker (har altså ikke at head->right er rota).
```

`pos` er nodens posisjon i treet. Rota er nr.1. Om en node har `pos` lik  $i$ , vil et evt. venstre barn ha `pos` lik  $2*i$ , mens et evt. høyre barn ha `pos` lik  $2*i + 1$ . Altså etter samme prinsipp som i en heap (representert som en array), bare at subtrær/barn kan mangle (treet trenger ikke være fullt eller komplett). Eksempel på et slikt tre, der `pos`-verdier (etter reglene ovenfor) er skrevet til høyre for noden (noder må gjerne ha duplikat ID):



**NB1:** I *hele* oppgave 3 skal det *ikke* innføres globale data eller flere `struct`-medlemmer enn angitt ovenfor. Det skal heller *ikke* brukes andre større strukturer - som f.eks. array, stakk, kø eller liste.

**Men:** I oppg.3c *kan* det være aktuelt (men *ikke* nødvendig) å bruke/innføre en litt mindre/kortere array.

**NB2:** `n` inn til funksjonene og `gRoot` *kan* være `nullptr/NULL`.

### a) 9 Lag de to rekursive funksjonene

```
void speilvend(Node* n) og
void settPosisjoner(Node* n, int p)
```

Den første funksjonen skal speilvende noden `n` og alle dens subtrær/barn. Dvs. bytte om på sine venstre og høyre subtrær/barn, og at dette skjer rekursivt videre nedover. Kalles fra `main` med: `speilvend(gRoot)`; Når dette har skjedd vil ikke `pos` lengre være korrekt i nodene. Den andre funksjonen skal derfor rekursivt gå gjennom noden med alle dets subtrær/barn, og sette rett verdi for `pos` i alle nodene igjen. Kalles fra `main` med: `settPosisjoner(gRoot, 1)`;

### b) 9 Lag den rekursive funksjonen

```
void speilvend(Node* n, int p)
```

Funksjonen skal *alene* utføre det samme som *begge* funksjonene ovenfor til sammen utfører. Kalles fra `main` med: `speilvend(gRoot, 1)`;

### c) 12 Lag den ikke-rekursive funksjonen

```
Node* finnNode(int p)
```

Funksjonen starter å lete i rota. Den returnerer en peker til noden som har klassesmedlemmet `pos` lik parameteren `p`. Finner den *ikke* en slik node, returneres `nullptr/NULL`.

**Forklar/redegjør klart for hvordan funksjonen din virker (dens tankegang/algoritme).**

**Hint:** Binærkoden for `p` er vesentlig.

## Oppgave 4 (koding, 20%)

Tallet 2.6 ganget med 135 er 351. Dvs. svaret er bare at det første sifferet er flyttet bakerst.

Det samme gjelder for alle disse regnestykkene, om faktoren er 2.6 og tallet det ganges med er mindre eller lik 1 million:

$$2.6 * 270 = 702$$

$$2.6 * 135135 = 351351$$

$$2.6 * 270270 = 702702$$

**Skriv et fullverdig/komplett program som leser inn en faktor å gange med (f.eks. 2.6), og som for alle tall mindre eller lik 1 million, finner og skriver ut alle slike regnestykker som stemmer, ved at det første sifferet i tallet som faktoren ganges med flyttes bakerst, så får man svaret.**

Om ingen løsninger finnes, skal det komme en egen melding.

**NB:** Det skal ikke innføres noen ekstra hjelpe-datastrukturer for å løse/kode dette.  
Det holder med helt vanlig bruk av løkker og `if`-setninger.

---

**NB:** I *hele* dette oppgavesettet skal du *ikke* bruke kode fra (standard-)biblioteker (slik som bl.a. STL og Java-biblioteket). Men de vanligste `include/import` du brukte i 1.klasse er tilgjengelig. Koden kan skrives valgfritt i C++ eller Java.

**Løkke tæll!**  
**FrodeH**