NTNU
Kunnskap for en bedre verden

Institutt for datateknologi og informatikk

# Eksamensoppgave i IDATG2102 – Algoritmiske metoder

| | |
|---|---|
| **Faglig kontakt under eksamen:** | **Frode Haug** |
| **Eksamensdato:** | **24.november 2025** |
| **Eksamenstid (fra-til):** | **09:00-13:00 (4 timer)** |
| **Hjelpemiddelkode/Tillatte hjelpemidler:** | **I - Alle trykte og skrevne**<br>(kalkulator er *ikke* tillatt) |
| **Annen informasjon:** | |
| **Målform/språk:** | **Bokmål** |
| **Antall sider (inkl. forside):** | **4** |

**Informasjon om trykking av eksamensoppgaven**

**Originalen er:**

**1-sidig** X **2-sidig** ☐

**sort/hvit** X **farger** ☐

**Skal ha flervalgskjema** ☐

**Kontrollert av:**

_____

Dato Sign

# Oppgave 1 (teori, 25%)

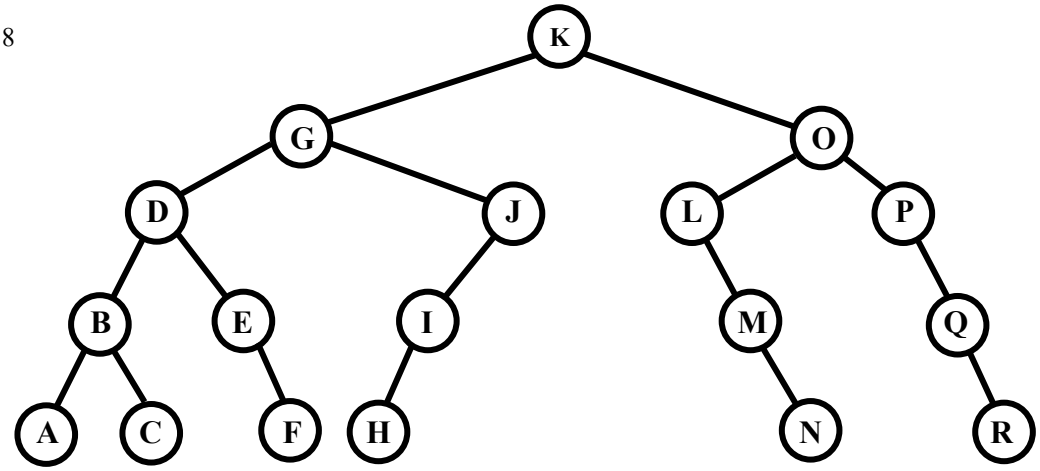
Denne oppgaven inneholder tre totalt uavhengige oppgaver fra pensum.

**a)** 9 Shellsort skal utføres på bokstavene «NOTUKHOST». For hver gang indre for-løkke i eksemplet med Shellsort er ferdig (dvs. rett etter: a[j] = 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.**

**b)** 8 I de følgende deloppgaver er det key'ene "N O T U K H O S T" (i denne rekkefølge fra venstre mot høyre, og blanke regnes *ikke* med) som du skal bruke. For alle deloppgavene gjelder det at den initielle heap/tre er *tom* før første innlegging ("Insert") utføres. **Skriv/tegn den *resulterende* datastruktur når key'ene legges inn i:**

1) **en heap**
2) **et binært søketre**
3) **et 2-3-4 tre**
4) **et Red-Black tre**

**c)** 8




Det skal fjernes («remove») noen noder fra det ovenfor gitte *binære søketreet*.
**Skriv/tegn treet for hver gang, og fortell hvilken av «if ..... else if ..... else»-grenene i `EKS_28_BinertSokeTre.cpp` (dvs. Case 1, Case 2, Case 3) som er aktuelle når det etter tur fjernes henholdsvis bokstavene 'J', 'K' og 'O'.**

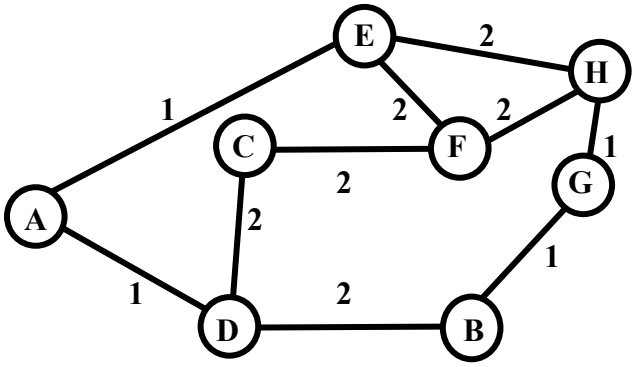
**NB:** For hver fjerning skal det *på nytt* tas utgangspunkt i *hele* det aktuelle treet. Dvs. *på intet tidspunkt* skal det fra treet være fjernet *mer enn en* bokstav.

# Oppgave 2 (teori, 25%)

Denne oppgaven inneholder tre totalt uavhengige oppgaver fra pensum.

**a)** 7 I forbindelse med dobbelt-hashing har vi teksten «NOTUKHOST» og de to hash-funksjonene `hash1(k) = k mod 11` og `hash2(k) = 3 - (k % 3)` der `k` står for bokstavens nummer i alfabetet (1-29). Vi har også en array med indeksene 0 til 10.
**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 vektede (ikke-rettede) graf er gitt:




Vi bruker nabomatrise, og starter i node C. **Skriv/tegn opp minimums spenntreet (MST)** for denne grafen, etter at koden i EKS_31_MST.cpp er utført/kjørt.
**Skriv også opp innholdet i/på fringen etterhvert** som konstruksjonen av MST pågår.
**NB:** *Husk at ved lik vekt så vil noden sist oppdatert (nyinnlagt eller endret) havne først på fringen ift. andre med den samme vekten.*

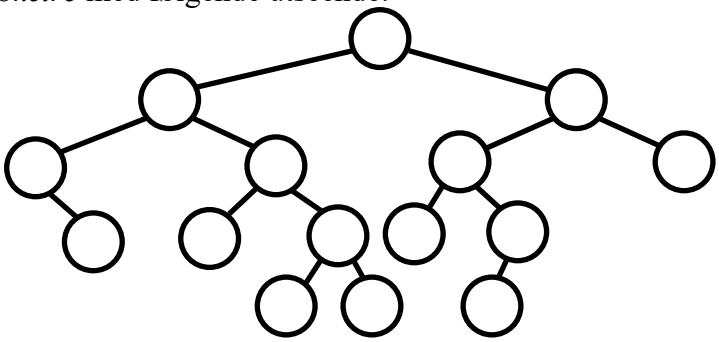
**c)** 9 Følgende kanter i en (ikke-retted, ikke-vekted) graf er gitt:

AC EB DB AD FG FC

**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.**

# Oppgave 3 (koding, 30%)

Vi har et *binært søketre* med følgende utseende:



Nodene i dette treet er representert vha:

```
struct Node {
  int   ID;          // Nodens ID/navn (et tall).
  Node* left;        // Peker til venstre subtre, evt. nullptr.
  Node* right;       // Peker til høyre subtre, evt. nullptr.
  int   leftNumber;  //  Antall noder totalt i venstre subtre.
  Node(char id) { ID = id; left = right = nullptr;  leftNumber = 0; }
};
```

Vi har også den globale variabelen:

```
Node* gRoot = nullptr; //  Rot-peker.
```

**NB:** Det skal i hele oppgave 3 *ikke* innføres noen flere globale variable, parametre, struct-medlemmer eller hjelpe-datastrukturer. Oppgavene kan løses vha. helt vanlig bruk av `if`-setninger/løkker.

**a)** 10 **1:** 4 **Skriv tallene 1-15 i en sekvens slik at hvis noder med disse IDen settes inn i denne rekkefølge (i et på forhånd tomt tre) så vil treet få samme form som figuren ovenfor.**

**2:** 2 Alle noder i et binært tre har et bestemt antall noder i sitt venstre subtre (null eller flere). **Tegn av treet ovenfor, og angi i forbindelse med hver node antallet i venstre subtre.**

**3:** 4 **Lag den *ikke-rekursive* funksjonen** `int minst()` **som returnerer `ID`en til den aller første noden i inorder rekkefølge i et vilkårlig binært søketre. Er treet tomt returneres 0 (null).**

**b)** 8 Koden for `insert` i EKS_28 litt omskrevet er som følger (**NB:** Vi bruker her *ikke* at `hode->right` egentlig er treets rot, men at `gRoot` *alltid initielt* peker til en *ekte* rot):

```
 1   void settInn(const int id) {
 2      Node* mor  = nullptr,
 3          * node = gRoot;
 4      if (node) {
 5        while (node) {
 6          mor = node;
 7          if (id < node->ID)
 8             node = node->left;
 9          else
10             node = node->right;
11        }
12        node = new Node(id);
13        if (id < mor->ID)  mor->left  = node;
14        else               mor->right = node;
15      }
16   }
```

Husk hvordan `leftNumber` skal brukes. **Endre/utvid koden ovenfor, slik at dette ivaretas for *alle* nodene som, ved innsettelse av *en* ny node, får *en* mer i sitt venstre subtre.**
Det holder at du angir (vha. linjenumrene) hvor den nye koden skal inn, og hva koden er.

**c)** 12 **Lag den *ikke-rekursive* funksjonen** `int hent(int n)` **som returnerer verdien til noden som er nr.'n' i inorder rekkefølge i treet.** Den første noden er nr.0, den andre er nr.1, osv. Du kan forutsette at `leftNumber` er korrekt satt i alle nodene, samt at `n` er et gyldig tall ift. totalt antall noder i treet.

# Oppgave 4 (koding, 20%)

Vi har i hvert fall:
```
const int ANT = 11;          ///< Antall punkter.
float avstand[ANT][ANT];     ///< Avstanden mellom alle punktene.
```

**Lag den rekursive funksjonen `void reise(int nr, int n, float curr)` og evt. max. *en* ekstra global enkelt-variabel og/eller *en* hjelpearray for å finne den korteste ruta/stien fra nr.0 og innom *alle* punktene (men *ikke* tilbake til nr.0 igjen).** Det skal *ikke* brukes Dijkstras algoritme på dette, men brukes rekursive kall av `reise(…)` og backtracking.
`nr` er punktet å besøke, `n` er antall punkter besøkt hittil og `curr` er *total*avstanden hittil tilbakelagt.
Det hele startes fra `main` ved kallet: `reise(0, 1, 0.0);`

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

**Løkke tæll!**
**FrodeH**