NTNU
Kunnskap for en bedre verden

Institutt for datateknologi og informatikk

Kontinuasjonseksamensoppgave i IDATG2102 – Algoritmiske metoder

Faglig kontakt under eksamen:	**Frode Haug**
Eksamensdato:	**5.august 2025**
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 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) 8 Et av eksemplene i pensum leser og omgjør et infix-uttrykk til et postfix-uttrykk.
Vi har infix-uttrykket: ((((4 * 5) + 6) * (4 + (2 * 3))) + 8)
Hva blir dette skrevet på en postfix måte?
Skriv/tegn stakkens innhold etter hvert som koden leser tegnene i infix-uttrykket.

b) 8 I de følgende deloppgaver er det key'ene "S T R U T S E K L O"
(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**

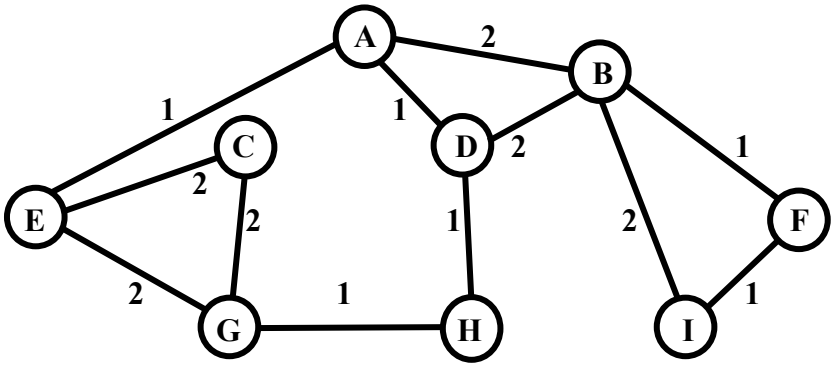
c) 9 Shellsort skal utføres på bokstavene «STRUTSEKLO». For hver gang indre for-løkke i eksemplet med Shellsort er ferdig (dvs. rett etter: a[j] = verdi;) :
Skriv/tegn opp arrayen og skriv verdiene til 'h' (4 og 1) og 'i' underveis i sorteringen.
Marker spesielt de key'ene som har vært involvert i sorteringen.

Oppgave 2 (teori, 25%)

Denne oppgaven inneholder tre totalt uavhengige oppgaver fra pensum.

a) 8 I forbindelse med dobbelt-hashing har vi teksten «STRUTSEKLO» og de to hash-funksjonene `hash1(k) = k mod 13` og `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 vektede (ikke-rettede) graf er gitt:



Vi bruker nabomatrise, og starter i node E. **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.*

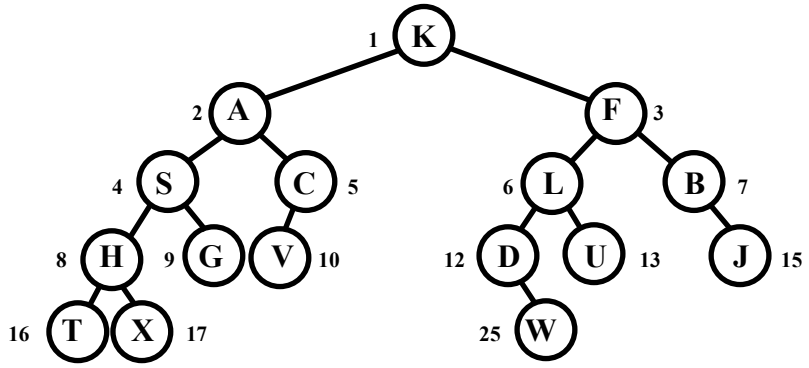
c) 8 Følgende kanter i en (ikke-retted, ikke-vekted) graf er gitt:

AB EB CB DA DB

Utfør Union-Find *m/kun path compression (PC)* på denne grafen.
Skriv/tegn opp innholdet i `gForeldre` etter hvert som `unionerOgFinn2` kjøres/utføres. Bemerk hvor PC er brukt.
Skriv/tegn også opp den resulterende union-find skogen.

Oppgave 3 (koding, 30%)

Vi har et *binært tre* (*ikke* søketre) som ser slik ut:



Roten er alltid nummerert som nr.1. Om node nr.i har et venstre barn, så er denne nr.i*2,
mens et evt. høyre barn har nr.i*2+1 – akkurat som ifm. heap i emnets pensum.

a) **Oppg. a1:** Når treet ovenfor traverseres på en *inorder* måte:
Skriv nodene på formen "(X, k)", der 'X' er nodens ID og 'k' er dets nummer.

Oppg. a2: Vi har gitt følgende om nodene i et binært tre (som beskrevet ovenfor):

X:	W	O	S	G	J	P	X	Z	B	Y	F	T	M
k:	1	2	3	4	7	14	15	28	29	56	57	114	229

Uti fra disse dataene: **Tegn det binære treet** (når de samme prinsippene som ovenfor er fulgt).

b) Nodene i vårt binære tre er representert vha:

```
struct Node  {
  char ID;          //  Nodens ID/navn (en bokstav).
  Node* left;       //  Peker til venstre subtre, evt. nullptr.
  Node* right;      //  Peker til høyre subtre, evt. nullptr.
  Node (char id, Node* l, Node* r) { ID = id; left = l; right = r; }
};
```

Vi har også den globale variabelen:

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

NB1: Legg merke til at `Node` *ikke* inneholder noen data om dets nummer (verdien 'k').
NB2: Keyene i treet trenger *ikke å være unike*, selv om de er det på figuren og i tabellen ovenfor.
NB3: Funksjonene nedenfor startes fra `main` ved bl.a. å sende med `gRoot` som parameter.
NB4: 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 (og evt. løkker).

Lag den *rekursive* funksjonen `void skriv(Node* node, int nr)`
Funksjonen skal (som i oppg. a1) sørge for at treet traverseres på en inorder måte, og at hver node får hver sin linje med utskrift på formen "(X, k)".

c) **Lag den *rekursive* funksjonen** `bool finn(Node* node, char id)`
Funksjonen(e) skal returnere `true/false` til om minst en node i treet har `ID` lik `id`.
NB: Husk at treet *ikke* er et binært søketre - det er totalt usortert.

Oppgave 4 (koding, 20%)

Vi har en lang tekst med *kun små og store bokstaver* fra a-z og A-Z. Kommer samme bokstav, bare skrevet som liten og stor rett etter hverandre, skal disse to fjernes fra teksten.
Dette fortsetter til det ikke er mulig å fjerne to bokstaver (som oppfyller dette prinsippet).

For eksempel:

dabA**cC**aCBAcCcaDA	- den første «cC» (fet) skal fjernes
dab**Aa**CBAcCcaDA	- dette skaper «Aa» (fet), som igjen skal fjernes
dabCBA**cC**caDA	- første «cC» (fet) fjernes
dabCBAcaDA	- intet mer kan fjernes, så dette er slutteksten

NB: Legg merke til at *til enhver tid* er det *de to første bokstavene* i teksten som tilfredsstiller vårt prinsipp som skal fjernes.

Skriv et komplett program som gjør denne jobben med en tekst/string, og som avslutter med å skrive ut slutteksten.

Ad C++-programmering:

Startteksten ligger i: `string txt = "..... ";`
Tekstens nåværende lengde kan hentes vha: `len = txt.length();`
Tegn kan fjernes fra en tekst vha: `txt.erase(<start index>, <antall tegn>);`
`islower(c) / isupper(c)` retunerer begge `true/false` til om `'c'` er henholdsvis en liten eller stor bokstav.
`tolower(c) / toupper(c)` retunerer begge `'c'` gjort henholdsvis om til en liten eller stor bokstav.

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