

Institutt for datateknologi og informatikk

Kontinuasjonseksamensoppgave i IDATG2102 – Algoritmiske metoder

Faglig kontakt under eksamen: Frode Haug
Tlf: 950 55 636

Eksamensdato: 14.august 2024
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:

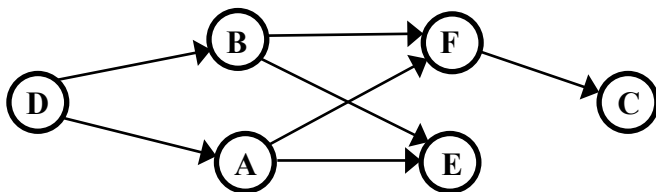
Dato

Sign

Oppgave 1 (teori, 25%)

Denne oppgaven inneholder tre totalt uavhengige oppgaver fra pensum.

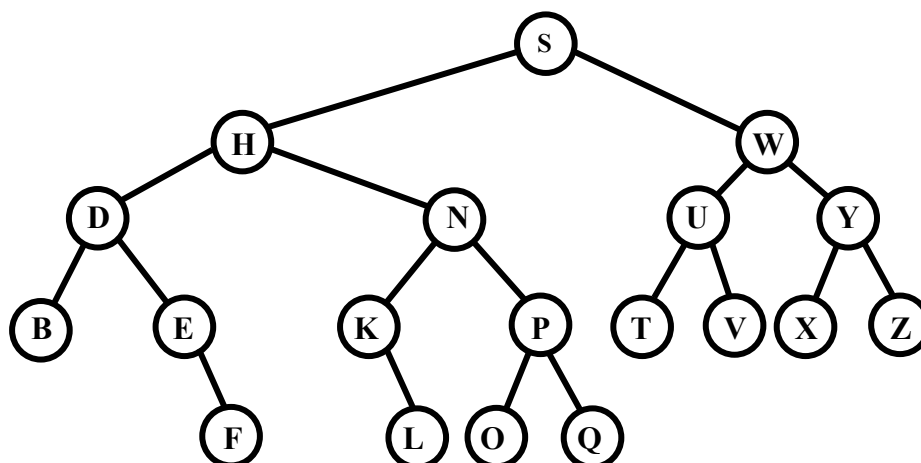
- a) 9 Quicksort skal utføres på bokstavene/keyene: STROGANOFF
Lag en oversikt/tabell der du for hver rekursive sortering skriver de involverte bokstavene og markerer/uthever hva som er partisjonselementet.
- b) 10 I de følgende deloppgaver er det key'ene "S T R O G A N O F F"
(i denne rekkefølge fra 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) 3 en heap
 - 2) 2 et binært søketre
 - 3) 3 et 2-3-4 tre
 - 4) 2 et Red-Black tre
- c) 6 Angi alle mulige topologiske sortingssekvenser av nodene for den rettede (ikke-vektede) asykliske grafen («dag»):



Oppgave 2 (teori, 25%)

Denne oppgaven inneholder tre totalt uavhengige oppgaver fra pensum.

a) 6



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 BinertSøkeTre.cpp (dvs. Case 1, Case 2, Case 3) som er aktuelle når det etter tur fjernes henholdsvis bokstavene 'W', 'D' og 'H'.

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.

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

BF CA CD BA EG GF

Utfør Union-Find *m/weight balancing (WB)* (ikke path compression - PC) på en vanlig graf (ikke-retted, ikke-vekted). Skriv/tegn opp innholdet i gForeldre etter hvert som unionerOgFinn2 (uten PC) kjøres/utføres. Bemerk hvor WB er brukt. Skriv/tegn også opp den resulterende union-find skogen.

c) 11 Vis konstruksjonsprosessen når koden for Huffman-koding i EKS_39_Huffman.cpp brukes på teksten: SPURS SNIKER SEG SAKTE ETTER ARSENAL (inkludert de blanke). Hvor mange bits trengs for å kode denne teksten? Dvs. skriv/tegn opp:

- frekvens-arrayen
- forelder-arrayen
- Huffmans kodingstreet/-trien
- bokstavenes bitmønster (kode) og lengde
- totalt antall bits som brukes for å kode teksten

Oppgave 3 (koding, 32%)

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

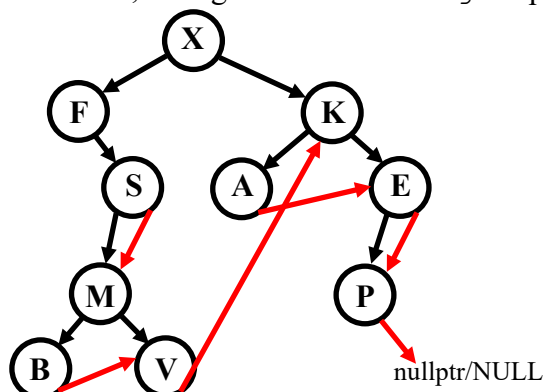
```
struct Node {
    char ID; // Nodens ID/key/nøkkel/navn (ett tegn).
    Node *left, *right; // Referanser til begge subtrærne eller neste i preorder rekkefølge
                        // eller nullptr/NULL (se resten av oppgaveteksten).
    bool nesteIPreorder; // Er false dersom høyre subtre finnes,
                        // er true dersom tomt høyre subtre og at right
                        // da i stedet peker til neste node i preorder rekkefølge.
                        // Har noden er tomt høyre subtre, og er den aller siste i
                        // preorder rekkefølge, vil den også være true,
                        // bare at right da vil peke til nullptr/NULL.

    Node (char id, int p) // Constructor:
    { ID = id; nesteIPreorder = false; left = right = nullptr; }
};
```

Vi har også de globale variablene:

```
Node* gRoot = nullptr, // Rot-peker (har altså ikke at head->right er rota).
* gForrige = nullptr; // Den forrige noden i preorder rekkefølge (jfr. oppg.3b).
```

Legg merke til kommentaren ovenfor ifm. *nesteIPreorder*. Om vi bare har en peker til en vilkårlig node (kanskje midt inne i treet), vil det da være både mulig og lett å finne den neste noden i preorder rekkefølge i treet. Eksempel på et slik tre, inntegnet bruken av *right* på denne måten (røde piler):



- a) 12** Lag den ikke-rekursive funksjonen `Node* neste(Node* node)`
 som for `node` returnerer en peker til den neste noden i preorder rekkefølge.
 Dersom `node` er den siste noden i preorder rekkefølge, eller at den initielt peker til `nullptr/NULL`, skal funksjonen returnere `nullptr/NULL`. *Legg vekt på at funksjonen blir effektiv!* (Hvordan treet allerede har blitt bygd/satt opp, trenger du ikke å tenke på.)
- b) 20** Vi har nå ett eller annet tre. *Men bruken av de røde pekerne er nå ennå ikke satt.*
 Dvs. nodene som *ikke* har reelle høyre subtrær/barn har ennå *ikke* fått sine `right` og `nesteIPreorder` korrekt oppdatert.
Lag den rekursive funksjonen: `void settNeste(Node* node)` som går gjennom hele treet under `node` og setter korrekt verdi i alle noderes `right` og `nesteIPreorder`.
Hint: Det kan være lurt å benytte seg av den globale variabelen `gForrige`. Denne bør alltid peke til den *forrige* noden i preorder rekkefølge, evt. bare til den forrige noden som *ikke* har et reelt høyre subtre, og dermed skal få sine to datamedlemmer oppdatert snart.
 Du trenger nødvendigvis ikke å tenke på å spesialhåndtere denne ifm. den aller siste noden i preorder rekkefølge.
- NB:** I hele oppgave 3 skal det *ikke* innføres globale data eller flere `struct`-medlemmer enn angitt ovenfor. Det skal heller *ikke* brukes andre hjelpestrukturer - som f.eks. array, stakk, kø eller liste.

Oppgave 4 (koding, 18%)

- Lag funksjonen `void skrivDato(const int dagNr)`
 Funksjonen skriver ut en feilmelding om `dagNr` er utenfor intervallet 1-365.
 Ellers beregner og skriver den ut hvilken dato (dag og måned) dette er.
 F.eks. om `dagNr` er 96, skriver den ut: 6/4 (altså. 6.april).
- NB1:** Aktuelt år er *ikke* et skuddår.
 Dvs. de 12 månedene har dagantallet: 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31
- NB2:** *For full score vektlegges kompakthet og effektiv kode.*

-
- 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