Oppgaver

IDATG2102 - Algoritmiske metoder

Forord

Dette kompendie/hefte innholder oppgaveteksten for ulike oppgaver i emnet ”IDATG2102 – Algoritmiske metoder” ved NTNU.

Opplysninger om emnet er å finne på: **http://folk.ntnu.no/frh/algmet**

Under ”Opplegg” er det angitt hvilke oppgaver i dette heftet som til enhver tid er relevante som ukeoppgaver.

Alle løsningsforslag (og evt. datafiler som oppgaveteksten henviser til) er å finne på katalogen ”OPPGAVER”.

Nye/andre oppgavetekster *vil* bli laget i løpet av semesteret. Disse vil fortløpende bli lagt til bakerst i dette heftet.

NTNU

Frode Haug

Oppgave 1

Ta utgangspunkt i koden i EKS_01_Vector.cpp:

1. `main` inneholder initielt noe kode. **Skriv mer/annen kode som tester funksjonene i `Vector`-klassen**, også dersom vectoren består av `int` eller `char` (eller `string`).

2. **Lag innmaten til funksjonen** `void resize(const int nyLengde)`
   Funksjonen sikrer at `nyLengde` er større enn nåværende `kapasitet`. Er den det, oppdateres `kapasitet` til ny verdi, det lages en ny og lengre array (`new T[…]`), eksisterende array kopieres over til denne, gammel array slettes (`delete`) og `data` settes til å peke på den nye og nå lengre arrayen.

3. **Skriv litt om funksjonen** `bool insert(const int pos, const T t)`
   slik at den ikke lengre kommer med en melding dersom arrayen er full, men i stedet sørger for å øke nåværende arrays kapasitet med +100, før resten av funksjonens innmat bare fortsetter.

Oppgave 2

Implementer den abstrakte datatypen `Map` (etter mønster fra de fire andre container-klassene hittil presentert i emnet), med følgende tilgjengelige operasjoner:

- **Konstruere** en tom map
- **Antallet** i `map`'en
- **Skrive hele `map`**'ens innhold
- **Sette inn et *nytt* par** med key og data
- **Finne** (om mulig) en key
- **Hente ut** en keys tilhørende data
- **Endre** en allerede eksisterende keys tilhørende data (duplikate keyer får jo ikke lov til å forekomme, derfor endres evt. en eksisterende keys tilhørende data)
- **Slette helt paret** med key og data

Husk at en `map` inneholder *unike* keyer, hver med noe tilhørende data.
Denne keyen er nøkkelen/verdien/indeksen for å finne de tilhørende dataene.

Oppgave 3

EKS_05_InfixTilPostfix.cpp leser og omgjør et infix-uttrykk til et postfix-uttrykk.

a) Vi har infix-uttrykket: (((4 + 2) + (3 * 2)) + ((3 + 4) * (2 * 5)))
   Hva blir dette skrevet på en postfix måte ? Skriv/tegn opp stakkens innhold etter hvert som koden leser tegnene i infix-uttrykket.
b) Hva blir alt dette for infix-uttrykket: ((7 * 3) + (((4 * 5) * (3 + 2)) * (6 + 3)))

Oppgave 4

EKS_06_PostfixTilSvar.cpp leser et postfix-uttrykk og regner ut svaret.

a) Vi har postfix-uttrykket: 3 4 + 3 2 * + 2 + 5 3 * 4 2 + * +
   Hva blir svaret ? Skriv/tegn opp stakkens innhold etter hver gang den er endret.
b) Hva blir alt dette for postfix-uttrykket: 3 4 + 3 * 2 + 5 2 * + 2 *

Oppgave 5

Se EKS_08_TreTraversering.cpp. **Lag den manglende innmaten til funksjonen:**

```
void traverserPostorder(Node* node)
```

Det *skal* brukes stakk ifm. kodingen, og det *er lov til* å bruke `besokt` inni `Node`.

Oppgave 6

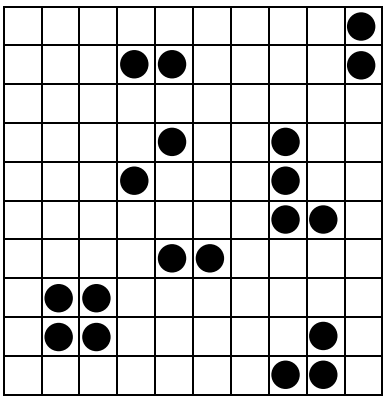
Implementer et binært tre (jfr. EKS_08 og EKS_12).
Lag *fire* rekursive funksjoner som hver for seg finner:

1. antall noder
2. antall pekere/referanser til `nullptr` / `null`
3. antall *fulle* noder (der `left` og `right` peker/refererer til noder, og *ikke* til `nullptr` / `null`), *og skriver ut disse nodenes ID/key/data*.
4. treets høyde

Alle funksjonene tar en `Node` (root) som input-parameter.
De skal enten returnere med en `int` *som svar, eller at de oppdaterer en global variabel.*

Oppgave 7

Vi har et NxN rutenett. I en del av rutene er det plassert brikker. Dette kan f.eks. se slik ut:



To ruter med brikker i sies å tilhøre den samme gruppen, dersom de vannrett eller loddrett (men *ikke* diagonalt) er i naboruter. På figuren ovenfor vil det derfor være to grupper med fire brikker, en gruppe med tre brikker, tre grupper med to brikker og to grupper med en brikke. Anta at rutenettet er representert vha. den to-dimensjonale `int`-arrayen: `brett[N][N]`. Vi bruker indeksene fra 0 til N-1. Initielt inneholder en rute '0' dersom den er tom, og '1' dersom det står en brikke i vedkommende posisjon. *Bruk gjerne '2' for å markere at ruten er registrert som del av en gruppe.*

Skriv et komplett program (bl.a. bestående av *en* rekursiv funksjon) som finner grupper, og *for hver av dem* skriver:

- **gruppens nummer** (fortløpende fra 1 og oppover, etter som man finner ulike grupper)
- **hvilke ruter** (identifisert vha. et 'i' og 'j' par) **som inngår i gruppen**
- **antall ruter i gruppen** (dvs. totalt antall brikker som utgjør hver enkelt gruppe.

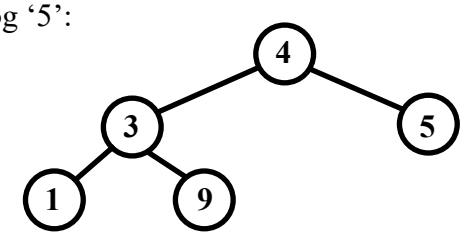
Oppgave 8

Ta utgangspunkt i koden på filen: OPPG_08-HOVEDRISS.cpp/java

a) **Lag den *rekursive* funksjonen** `int finnMin(Node* node)`
Funksjonen returnerer den *minste* verdien i treet tilpekt av og under `node`.
Er treet tomt returneres `MAX`. (Treet er *ikke* nødvendigvis et binært søketre.)

Lag også den *rekursive* funksjonen `int finnMax(Node* node)`
Funksjonen returnerer den *største* verdien i treet tilpekt av og under `node`.
Er treet tomt returneres `MIN`. (Treet er *ikke* nødvendigvis et binært søketre.)

b) **Lag den *rekursive* funksjonen** `bool erBST(Node* node)`
Funksjonen returnerer `true/false` til om treet tilpekt av og under `node` er et binært *søketre* (BST) eller ei. Du bør bruke begge funksjonene fra oppgave a (bare tilkall/bruk dem, selv om du evt. ikke har implementert/kodet dem).
NB: For det holder nemlig *ikke* bare å sjekke om en nodes venstre barn er mindre enn noden selv, og om høyre barn er større eller lik. En slik algoritme/tankegang vil medføre at f.eks. følgende tre blir vurdert som BST – hvilket det *ikke* er, da '9' er til venstre for '4' og '5':



NB: I *hele* denne oppgaven:

- *kan* `n` inn til alle funksjonene og `gRoot` være `nullptr`.
- skal det *ikke* innføres flere globale data eller `struct`-medlemmer enn de gitte. Det skal heller *ikke* brukes andre hjelpestrukturer - som f.eks. stakk, kø eller liste.

Oppgave 9

Anta at vi har en skoleklasse, og at alle er stilt opp ved siden av hverandre på en lang linje. De tildeles fortløpende numre fra 1 og oppover til N, fra venstre mot høyre. Nå skjer det triste at alle begynner å krangle noe voldsomt med naboene sine. Dette medfører at nr.1 blir uvenn med nr.2 og omvendt. Nr.2 blir også uvenn med nr.3 og omvendt, ….. osv…..
helt opp til nr.N. **Finn *alle mulige lovlige måter* å stille opp klassen på igjen, *uten* at uvenner blir stående rett ved siden av hverandre (dvs. ingen tall med pluss/minus en ift. hverandre skal være naboer)**. Dvs. oppgaven går ut på å finne permutasjoner av N stk, der ingen tall står ved siden av tall som er nøyaktig *en* høyere eller *en* mindre enn seg selv.

Oppgave 10

a) Hvilken av metodene Selection sort og Insertion sort er *raskest for en array som allerede er ferdig sortert*?

b) Hvilken av metodene Selection sort og Insertion sort er *raskest for en array som er baklengs sortert*?

c) Selection sort er *ikke stabil* (se under «NB» på EKS_21_SelectionSort.cpp).
Men hva er Insertion sort og Shellsort?

Oppgave 11

Shellsort skal utføres på noen bokstaver/keyer. For hver gang indre for-løkke 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.

a) Gjør dette for bokstavene/teksten: SUMPSVAMP

b) Gjør dette for bokstavene/teksten: KARKINAGRI

Oppgave 12

Quicksort skal utføres på noen bokstaver/keyer.
Lag en oversikt/tabell der du for hver rekursive sortering skriver de involverte bokstavene og markerer/uthever hva som er partisjonselementet.

a) Gjør dette for bokstavene/teksten: PIEMONTE

b) Gjør dette for bokstavene/teksten: PROSEDYRE

c) Gjør dette for bokstavene/teksten: LIGAMESTER

Oppgave 13

Skriv den *resulterende* arrayen når key'er legges (fra venstre mot høyre) **inn i en heap.**

a) Gjør dette for bokstavene/teksten: EDINBURGH

b) Gjør dette for bokstavene/teksten: KARAVOSTAMO

Oppgave 14

a) Følgende heap er gitt: 98 87 72 49 73 70 70 45 40 46 42 31 39

Utfør etter tur følgende operasjoner på denne heap: insert(87), insert(86), remove(), remove() og replace(24). **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.

b) Gjør det samme med heapen: 74 58 59 52 57 56 58 47 48 49 50 51

og operasjonene: insert(57), insert(75), remove(), remove() og replace(50)

Oppgave 15

Se EKS_25_Heap.h **Lag innmaten til de to funksjonene `change` og `extract`**
Forklaring til hva funksjonene skal gjøre står allerede i .h-filen.

Hint: `change`: Dersom det nye elementet er større enn det erstattede,
så skal det upHeapes, eller så skal det downHeapes.
`extract`: Det aller bakerste elementet flyttes inn der det slettede lå.
Dersom det flyttede er større enn det slettede,
så skal det upHeapes, eller så skal det downHeapes.

Oppgave 16

Heapsort (vha. bottom-up heap konstruksjon) skal utføres på noen bokstaver/keyer.
Skriv/tegn opp heapens innhold etterhvert som heapen konstrueres og deretter sorteres.

a) Gjør dette for bokstavene/teksten: KNATTHOLMEN

b) Gjør dette for bokstavene/teksten: GDANSKSOPOT

Oppgave 17

Skriv/tegn den *resulterende* datastruktur når bokstaver settes inn i et *binært søketre*:

a) Gjør dette for bokstavene/teksten: EDINBURGH

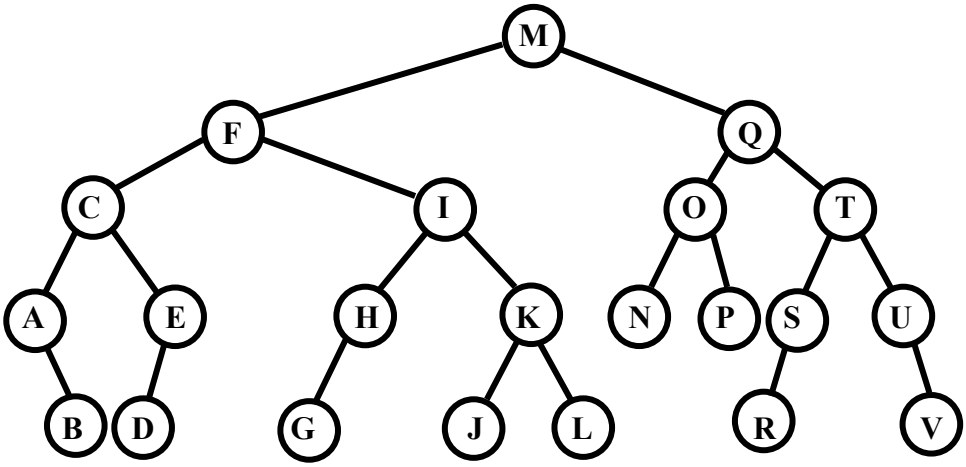
b) Gjør dette for bokstavene/teksten: KARAVOSTAMO

Oppgave 18

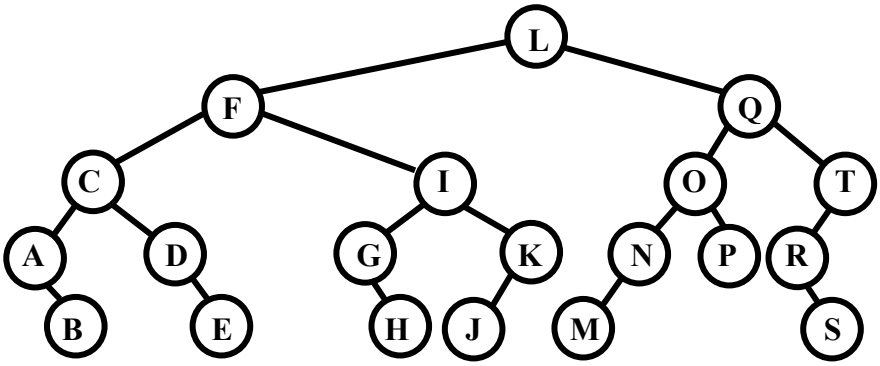
Det skal fjernes («remove») noen noder fra et *binært søktre*.
Skriv/tegn opp treet for hver gang, og fortell hvilken av «if ..... else if ..... else»-grenene i koden (dvs. Case 1, Case 2, Case 3) som er aktuelle når det etter tur fjernes ulike bokstaver.

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

a) Fjern etter tur fra treet bokstavene: **'H', 'F' og 'T'**



b) Fjern etter tur fra treet bokstavene: **'C', 'T' og 'L'**



Oppgave 19

Skriv/tegn det *resulterende* 2-3-4 treet når bokstaver settes inn i det.
Gjør også om sluttresultatet til et Red-Black tre.

a) Gjør dette for bokstavene/teksten: STRAFFESAKADVOKATEN

b) Gjør dette for bokstavene/teksten: PROSEDYREKONKURRANSE

Oppgave 20

Se EKS_29_Hashing.cpp. Vi lar hash-funksjonen være: `hash1(k) = k mod X`
der `k` står for bokstavens nummer i alfabetet (1-29). Vi har også en array med indeksene 0 til X-1. **Skriv hver enkelt bokstav sin returverdi fra hash1.**
Skriv også opp arrayen hver gang bokstaver legges inn i den vha. *linear probing*.

a) Gjør dette for bokstavene: RØDBILLESUPPE og X = 13

b) Gjør dette for bokstavene: EMIRATESSTADIUM og X = 17

Oppgave 21

Se EKS_29_Hashing.cpp. Vi lar hash-funksjonene være: `hash1(k) = k mod X`
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 X-1.
Skriv hver enkelt bokstav sin `k`-verdi og returverdi fra både `hash1` og `hash2`.
Skriv også opp arrayen hver gang en bokstav legges inn i den vha. *double hashing*.

a) Gjør dette for bokstavene: GDANSKSOPOT og X = 13

b) Gjør dette for bokstavene: KNATTHOLMEN og X = 13

Oppgave 22

Skriv/tegn opp Merkle treet som er basert på:

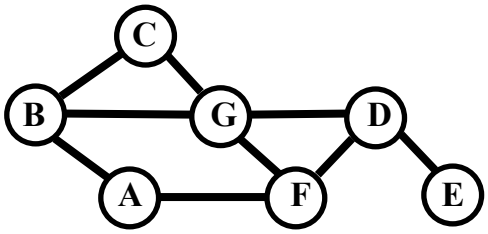
a) 4 blokker

b) 9 blokker

c) 12 blokker

Oppgave 23

Vi har grafen :



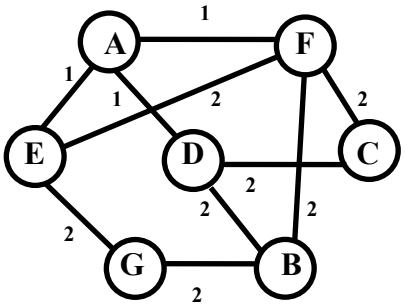
a) **Skriv opp nabomatrisen**.

b) **Skriv/tegn dybde-først-søketreet,** ved bruk av nabomatrisen og når koden `DFS(…)` i EKS_30_DFS_BFS.cpp brukes/kjører, og *vi starter i node A*.

c) **Skriv/tegn dybde-først søketreet** når vi i stedet *starter i node D*.

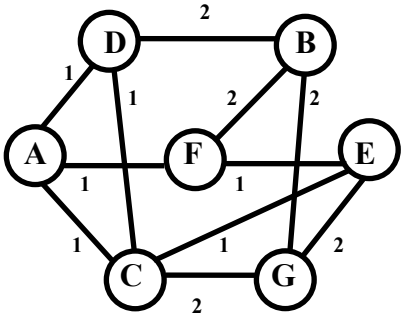
Oppgave 24

a) Følgende vektede (ikke-rettede) graf er gitt:



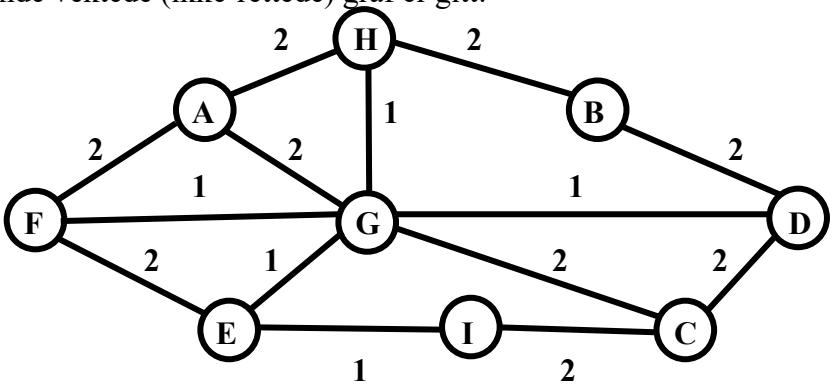
Vi bruker nabomatrise, og starter i node A. **Skriv/tegn opp minimums spenntreet (MST)** for denne grafen, etter at koden i EKS_31_MST.cpp er utført/kjørt. **Skriv også opp innholdet i/på fringen etterhvert** som konstruksjonen av MST pågår. **NB:** *Husk at ved lik vekt så vil noden sist oppdatert (nyinnlagt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

b) Gjør det samme som i oppgave a) med grafen (også startende i node A):



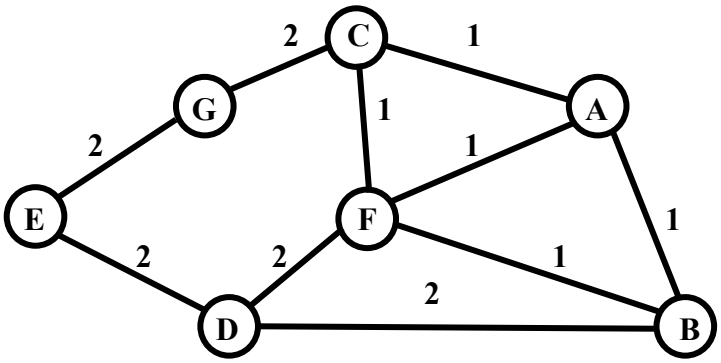
Oppgave 25

a) Følgende vektede (ikke-rettede) graf er gitt:



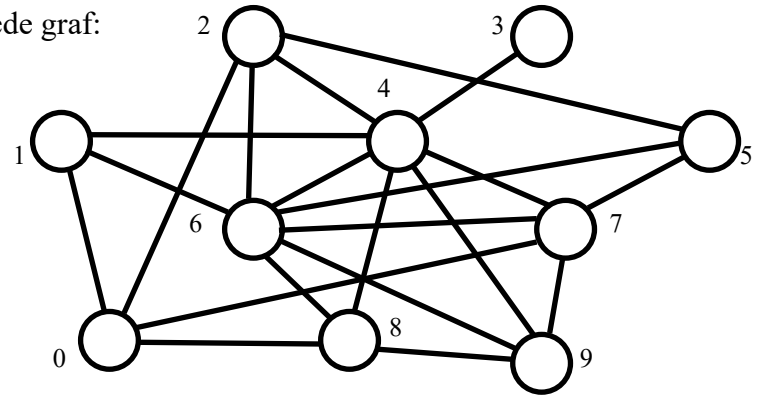
Vi bruker nabomatrise. Koden i EKS_32_SP.cpp utføres/kjøres på denne grafen. **Hvilke kanter er involvert i korteste-sti spenntreet fra noden B til *alle* de andre nodene? Skriv også opp innholdet i/på fringen etterhvert** som koden utføres. **NB:** *Husk at ved lik vekt så vil noden sist oppdatert (nyinnlagt eller endret) havne først på fringen ift. andre med den samme vekten.*

b) *Startende i node D*, gjør det samme som i oppgave a) med grafen:



Oppgave 26

Vi har følgende urettede/uvektede graf:

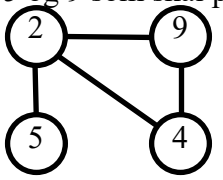


Tallene ved nodene er *ikke* deres id/navn/nummer, men bare en referanse til dem.
Utfordringen er å plassere (også) tallene/sifrene 0-9 som id/navn/nummer "*inni*" hver node.
Kravet til denne "navnsettingen" er at følgende blir oppfylt:

0: 21 1:33 2:14 3:27 4: 1 5:14 6:20 7:12 8:28 9:20

Dvs. tar man summen av id'ene (numrene) for naboene til den med id nr.0 (det er *ikke* den med *referanse* nr.0 ovenfor) får man 21, tar man summen av id'ene for naboene til den med id nr.1 får man 33, osv.

Eks (om det *kun* er tallene 2, 4, 5 og 9 som skal plasseres som id i en mindre/annen graf):



Så vil følgende være oppfylt: 2: 18 (4+5+9) 4: 11 (2+9) 5: 2 9: 6 (2+4)

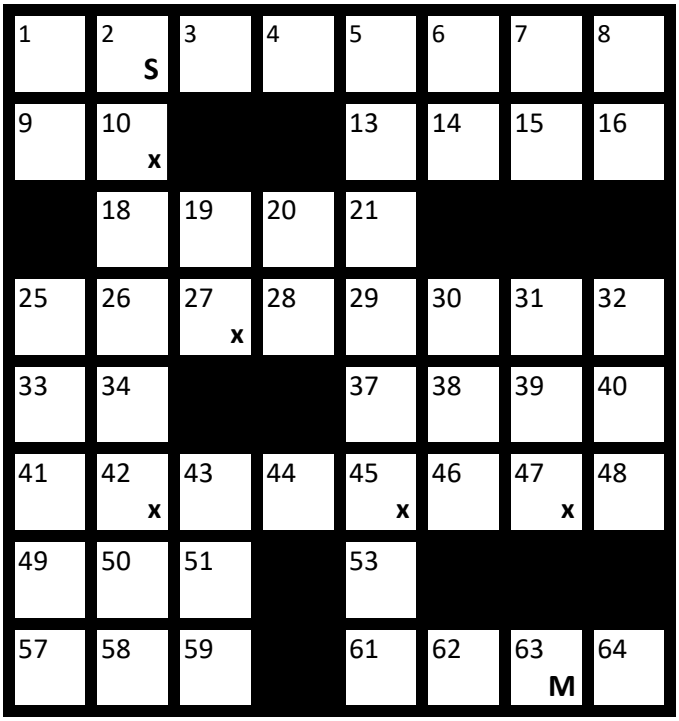
<u>Oppgave går ut på å skrive et komplett program (med hjelpearray(er) og funksjoner) som finner/skriver ut "alle" svarene på denne problemstillingen.</u>

Hint: Programmet bør nok bl.a. være basert på en rekursiv løsning, samt at kjent kode fra emnet kan sikkert brukes/utnyttes. Grafens nabomatrise ligger i:

```
        // Node nr:  0  1  2  3  4  5  6  7  8  9       Node nr:
int gNabo[N][N] = { { 0, 1, 1, 0, 0, 0, 0, 1, 1, 0 },    //  0
                    { 1, 0, 0, 0, 1, 0, 1, 0, 0, 0 },    //  1
                    { 1, 0, 0, 0, 1, 1, 1, 0, 0, 0 },    //  2
                    { 0, 0, 0, 0, 1, 0, 0, 0, 0, 0 },    //  3
                    { 0, 1, 1, 1, 0, 0, 1, 1, 1, 1 },    //  4
                    { 0, 0, 1, 0, 0, 0, 1, 1, 0, 0 },    //  5
                    { 0, 1, 1, 0, 1, 1, 0, 1, 1, 1 },    //  6
                    { 1, 0, 0, 0, 1, 1, 1, 0, 0, 1 },    //  7
                    { 1, 0, 0, 0, 1, 0, 1, 0, 0, 1 },    //  8
                    { 0, 0, 0, 0, 1, 0, 1, 1, 1, 0 } };  //  9
```

Oppgave 27

Vi har rutenettet med **S**(tart)- og **M**(ål)-ruter:



Hva vil den minste f-verdien (= g + h) i rutene 10, 27, 42, 45 og 47 (x) kunne være når:
(g er minste mulige vekt fra startruten og til vedkommende rute, jfr. Dijkstra.
h er verdien fra heuristikken brukt)

a) **det *kun* kan gås opp/ned/høyre/venstre (*ikke* på skrå) med en vekt på 1 (en), og som heuristikk brukes Manhatten distanse** (summen av antall ruter horisontalt og vertikalt til og inkludert målet)

b) **det kan gås opp/ned/høyre/venstre med en vekt på 1.00, på skrå i alle fire retninger med en vekt på 1.41, og som heuristikk brukes Euklidsk avstand**
(luftlinjen fra ruten og til målruten, dvs. hypotenusen (med to desimaler) i en trekant)
Ekstra: Hva er verdiene for de andre rutene på den korteste stien fra 'S' til 'M'?

Oppgave 28

Utfør Union-Find på en vanlig graf (ikke-retted, ikke-vekted).
Skriv/tegn opp innholdet i `gForeldre` etter hvert som `unionerOgFinn` kjøres/utføres. Skriv/tegn også opp den resulterende union-find skogen.

Gjør dette for grafen (med kantene):

a) AG GC CB BA FG BF AD DB AC EG

b) AF BG FG DC AG BE GD BD FC CE

Oppgave 29

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.**

Gjør dette for grafen (med kantene):

AE BC DF GA DB CE FE BF DG

Oppgave 30

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

Gjør dette for grafen (med kantene):

a) AE BA CA ED CE

b) BC AE DB BF EC (*i tillegg skal første bokstav få den andre som «foreldre»*)

Oppgave 31

Utfør Union-Find *m/weight balancing (WB) og path compression (PC)* på en vanlig graf (ikke-retted, ikke-vekted). **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.**

Gjør dette for grafen (med kantene):

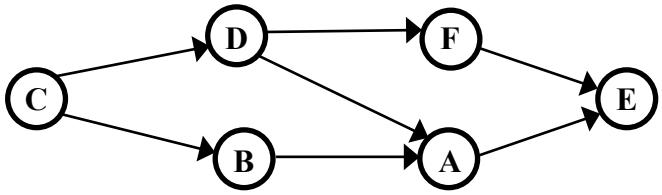
a) BE CD ED AC EA DA

b) FA EC BD DA EF DC CF

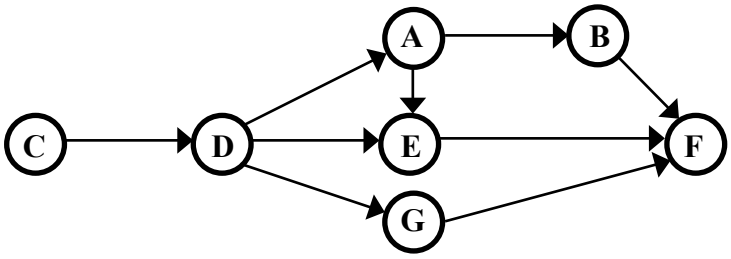
Oppgave 32

Angi ***alle* mulige topologiske sorteringssekvenser** av nodene for de rettede (ikke-vektede) asykliske grafene («dag»):

a)



b)



Oppgave 33

a) Vi har følgende bokstaver og `forelder`-array ifm. Huffman-koding (`frekvens`-arrayen er ukjent/irrelevant):

char(k)	‘ ‘	A	D	E	H	I	M	O	S	U
k	0	1	4	5	8	9	13	15	19	21
forelder[k]	-34	32	-30	-31	30	-27	29	-28	33	27

k	27	28	29	30	31	32	33	34	35
forelder[k]	28	-29	31	-32	-33	34	35	-35	0

Skriv/tegn opp Huffmans kodingstreet/-trien. **Skriv bokstavenes bitmønster**.

Vi har også følgende bitstrøm, som er kodet etter denne trien:
10100101100110100010100001101010110111001101100011
Hva står i denne meldingen?

b) Besvar de tre samme punktene som i oppgave **a)**, når `forelder`-arrayen og bitstrømmen er:

char(k)	‘ ‘	A	B	E	F	I	K	O	R	S
k	0	1	2	5	6	9	11	15	18	19
forelder[k]	33	-27	28	31	-30	-28	32	-34	27	-29

k	27	28	29	30	31	32	33	34	35
forelder[k]	30	29	-32	-31	34	-33	-35	35	0

00100000110000100010001001011000111110111111000101001111111

Oppgave 34

a) Vis **konstruksjonsprosessen** når koden **for Huffman-koding** i EKS_39_Huffman brukes på teksten SILDESALATEN SYNES SPESIELT SALT OG SYRLIG SIDEN DEN SMAKTE SLIK (inkludert de blanke – husk den ifm. linjeskiftet).
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

b) Besvar de fem samme punktene som i oppgave **a)**, bare med teksten:
TDF ER SYKKELTOUREN SOM STORT SETT SYKLES RUNDT OMKRING I FRANKRIKE