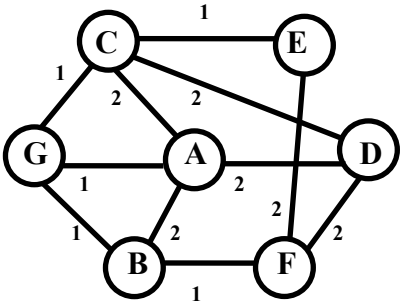
# Obligatorisk oppgave nr.5

## IDATG2102 – Algoritmiske metoder, høsten 2023

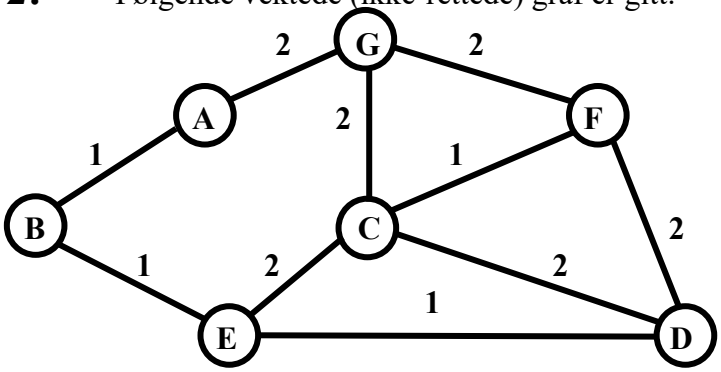
**Frist: 31.oktober 2023 kl.11:00 (*må* overholdes) *via Blackboard***

**Oppgave 1:** Følgende vektede (ikke-rettede) graf er gitt:




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

**Oppgave 2:** 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.*

# Oppgave 3:

Vi har en litt spesiell form for retted graf, der alle nodene har *maksimalt en etterfølger*, men *gjerne flere forgjengere*. Dvs. fra hver node går det *maksimalt en kant ut*, men *gjerne flere kanter inn*. Blant slike grafer finnes bl.a: enkelt-linkede lister, enkle sykliske strukturer (løkker) og rotrettede trær (der alle «barn» har *en* peker opp til «mora», mens «mor» har altså *ikke* en eller flere pekere ned til barna sine) + mange flere ulike grafer.

Nodene er representert ved:

```
struct Node {
  int   ID;                 // Nodens ID/key/data.
  Node* etterfolger;        // Peker til etterfølgeren, evt. nullptr.
  Node(const int id)  {  ID = id;  etterfolger = nullptr; }
};
```

Det er `N` noder. Den globale arrayen: `Node* gGraf[N+1]` har direkte pekere til *alle* grafens noder (vi bruker indeksene 1-`N`) i en eller annen *helt tilfeldig* rekkefølge.

**Lag funksjon bool erRotrettedTre() som returnerer `true/false` til om grafen *virkelig* er et rotretted tre eller ei. Beskriv også tankegangen for algoritmen/funksjonen**.

**NB:** Det skal *ikke* innføres flere globale data eller `struct`-medlemmer enn angitt ovenfor. Det skal heller *ikke* brukes andre hjelpestrukturer (utover den angitte `gGraf`) - som f.eks. array, stakk, kø eller liste.

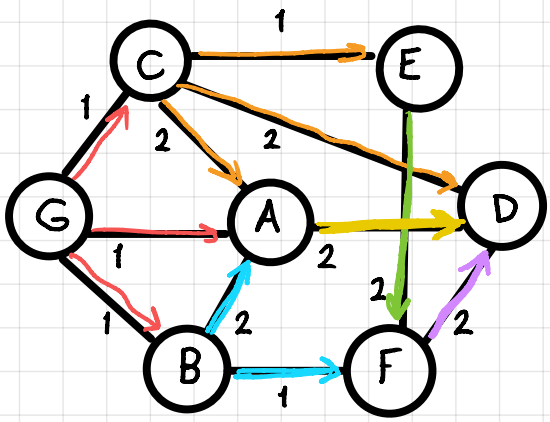
***Hele* obligen (både tegninger, tekst og kode for funksjonen) leveres som *en* PDF innen fristen via emnets rom i Blackboard.**

**FrodeH**

# oppgave 1

Fringe:

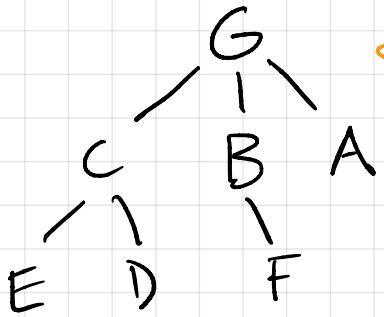
$F_{2e}$ blir til $F_{1b}$ fordi 1 er lavere vekt og lavere hvis vi finner lavere vekt, så bytter vi




| | | | | | |
|---|---|---|---|---|---|
| | | $E_{1c}$ | $B_{1g}$ | | |
| | $C_{1g}$ | $B_{1g}$ | $A_{1g}$ | $F_{1b}$ | |
| | $B_{1g}$ | $A_{1g}$ | $F_{2e}$ | $A_{1g}$ | $A_{1g}$ |
| G* | $A_{1g}$ | $D_{2c}$ | $D_{2c}$ | $D_{2c}$ | $D_{2c}$ $D_{2c}$ |

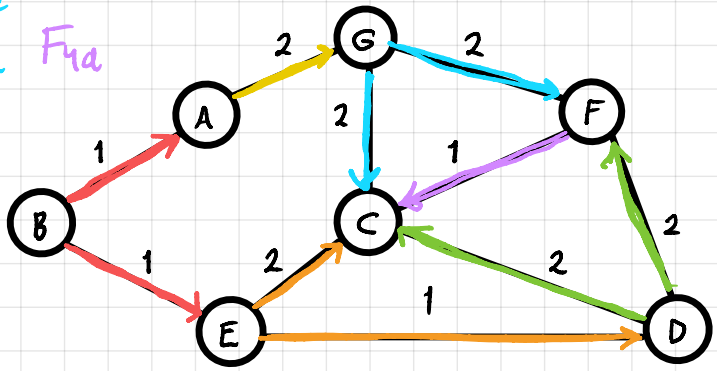
Hvis du ser på tallene og bokstavene i fringe, så ser du hvilke barn hver bokstav har. Feks $F_{1b}$ er da barnet til B.

MST:



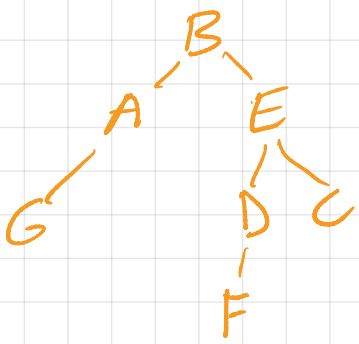

# Oppgave 2

| | | | | | |
|---|---|---|---|---|---|
| | | $A_{1b}$ | $D_{2e}$ | $G_{3a}$ | |
| | $E_{1b}$ | $D_{2e}$ | $G_{3a}$ | $C_{3e}$ | $C_{3e}$ |
| B* | $A_{1b}$ | $C_{3e}$ | $C_{3e}$ | $F_{4d}$ | $F_{4d}$ $F_{4d}$ |




BA BE AG ED EC DF

Deme er lettere å lage når du setter opp et MST →



# Obligatorisk Oppgave 5 Løsning

```
/**
* Oppgave nr 3 i oblig 5.
*
* Hentet fra oppgaven:
* Vi har en litt spesiell form for retted graf, der alle nodene har
* maksimalt en etterfølger, men gjerne flere forgjengere.
* Dvs. fra hver node går det maksimalt en kant
* ut, men gjerne flere kanter inn. Blant slike grafer finnes bl.a:
* enkelt-linkede lister, enkle sykliske strukturer (løkker) og
* rotrettede trær (der alle «barn» har en peker opp til «mora», mens
* «mor» har altså ikke en eller flere pekere ned til barna sine) +
* mange flere ulike grafer. Nodene er representert ved:
*
* struct Node {
* int ID; // Nodens ID/key/data.
* Node* etterfolger; // Peker til etterfølgeren, evt. nullptr.
* Node(const int id) { ID = id; etterfolger = nullptr; }
* };
*
* Det er N noder. Den globale arrayen: Node* gGraf[N+1] har direkte
* pekere til alle
* grafens noder (vi bruker indeksene 1–N) i en eller annen helt
* tilfeldig rekkefølge.
* Lag funksjon bool erRotrettedTre() som returnerer true/false til om
* grafen virkelig
* er et rotretted tre eller ei. Beskriv også tankegangen for
* algoritmen/funksjonen.
*
* NB: Det skal ikke innføres flere globale data eller struct-medlemmer
* enn angitt ovenfor.
* Det skal heller ikke brukes andre hjelpestrukturer (utover den angitte
* gGraf) – som
* f.eks. array, stakk, kø eller liste.
*
* Bruker DFS algoritme for å løse oppgaven.
*
* @file Oppgave3.CPP
* @author Sofia Serine Mikkelsen
*/

#include <iostream> //cout

struct Node {
        int ID;
        Node* etterfolger;
        Node(const int id) {
                ID = id;
```

```
                etterfolger = nullptr;
        }
};

const int N = 10; // Antall noder
Node* gGraf[N + 1]; // Globalt array med pekere til nodene

bool erRotrettedTreUtil(Node* node, bool visited[]) {
        if (visited[node->ID]) {
                // Hvis noden allerede er besøkt, så har vi en syklus
                return false;
        }

        visited[node->ID] = true;

        if (node->etterfolger != nullptr) {
                // Gå til neste node hvis den eksisterer
                return erRotrettedTreUtil(node->etterfolger, visited);
        }
        return true; // Dette er en bladnode
}

bool erRotrettedTre() {
        bool visited[N + 1] = {false};

        for (int i = 1; i <= N; i++) {
                if (!visited[i] && gGraf[i] != nullptr) {
                        if (!erRotrettedTreUtil(gGraf[i], visited)) {
                                return false; // Det er en syklus i grafen
                        }
                }
        }

        // Sjekk om det er akkurat ett rot-element
        int rotCount = 0;
        for (int i = 1; i <= N; i++) {
                if (!visited[i] && gGraf[i] != nullptr) {
                        rotCount++;
                        if (rotCount > 1) {
                                return false; // Mer enn ett rot-element
                        }
                }
        }

        return rotCount == 1; // Grafen er et rotretted tre hvis det er
akkurat ett rot-element
}

int main() {
        // Eksempel: Initialiser grafen med noder og pekere
        gGraf[1] = new Node(1);
        gGraf[1] -> etterfolger = gGraf[2];
        gGraf[2] = new Node(2);
```

```
        gGraf[2] -> etterfolger = gGraf[4];
        gGraf[3] = new Node(4);
        gGraf[3] -> etterfolger = gGraf[5];
        gGraf[4] = new Node(5);

        bool resultat = erRotrettedTre();
        std::cout << (resultat ? "Grafen er et rotretted tre."
        : "Grafen er ikke et rotretted tre.") << std::endl;

        return 0;
}
```