Tentamen i EDAF25

29 augusti, 2024

Skrivtid: 8-13

- Skriv bara på ena sidan av pappret
- Skriv tydligt – om texten inte går att läsa kan du inte få några poäng.
- Sätt identitet och sidnummer på varje inlämnat blad, kontrollera att sidnumret på din sista sida är samma som det antal blad du markerar på omslagspappret.

Tentamen består av uppgifter med totalt 40 poäng. Dessa poäng är fördelade på två avdelningar: *Grafteori* (10p) och *Design* (30p). För godkänt betyg kommer att krävas högst hälften av den sammanlagda poängen för de två avdelningarna.

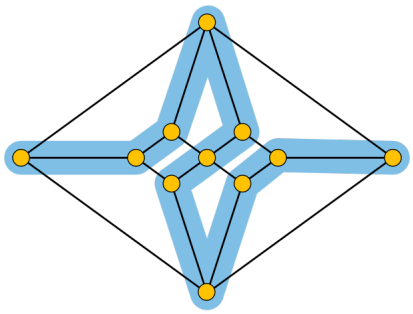
Vid bedömningen kommer hänsyn att tas till lösningens kvalitet. UML-diagram skall ritas i enlighet med UML-häftet.

Hjälpmedel:
- Andersson: UML–syntax – Enstaka exemplar finns att låna. Lämna tillbaka när tittat klart så andra kan låna dem.
- Holm: Java snabbreferens – Finns att låna.

Grafteori

Uppgift 1

En *Hamiltonväg* är en väg genom en graf som passerar alla noder i en graf en, och endast en, gång[1]. Ett exempel på en Hamiltonväg visas i nedanstående figur.



En enkel, men ineffektiv, algoritm för att avgöra om det finns en Hamiltonväg i en graf går ut på att gå igenom alla noderna i grafen i tur och ordning och för var och en prova om man från den kan hitta en väg genom grafen som går igenom alla andra noder. Denna algoritm kan beskrivas med nedanstående pseudokod där `g` representerar grafen som ska undersökas och ett anrop av `hamilton()` ger `true` eller `false` beroende på om det finns en Hamiltonväg i grafen eller inte.

```
hamilton(g) =
    foreach node u in g:
        m = a set containing all nodes in g except u
        if find_path(u,m)==true:
            return true
    return false


find_path(x,s) =
    if the set s is empty:
        return true
    foreach neighbour n to node x in set s:
        new_set = a set containing all nodes in s except x
        if find_path(n,new_set)==true:
            return true
    return false
```

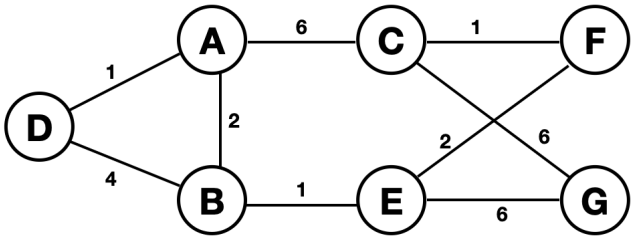
(a) Vilken från kursen känd algoritm gömmer sig bakom `find_path()`? *1 p*

(b) Förutom att avgöra om det existerar en Hamiltonväg så vill man också få veta hur denna ser ut (om det finns flera Hamiltonvägar kan en godtycklig av dem väljas). Skriv om `hamilton()`/`find_path()` så att `hamilton()` returnerar en lista med den funna Hamiltonvägen i stället för `true`/`false`, eller null om ingen väg finns. Svara med pseudokod med motsvarande abstraktionsgrad som i exemplet ovan. *4 p*

[1] Den minnesgode kanske kommer ihåg Hamiltonvägar från ”The Square-Sum Problem” som vi gick igenom kursivt på en av föreläsningarna i våras. Där skulle man ordna talen 1–15 så att summan av två tal som står intill varandra alltid var ett kvadrattal.

Uppgift 2

Nedanstående figur visar en graf med tillhörande avstånd (kostnader) för bågarna. Vi är intresserade av att hitta det kortaste avståndet från noden **B** till var och en av de övriga noderna.



(a) Antag att vi väljer att använda Dijkstras algoritm för att beräkna avstånden. I vilken ordning kommer algoritmen i så fall att fastställa det slutliga avståndet till noderna? Svara med att ange noderna i den ordning algoritmen bestämmer deras avstånd från noden **B**. *2 p*

(b) Även Bellman-Fords algoritm kan användas för att beräkna de kortaste avstånden i grafen. Nämn en användbar egenskap som Bellman-Fords algoritm har framför Dijkstras algoritm i det generella fallet. *1 p*

(c) Varför använder vi helst Dijkstras algoritm snarare än Bellman-Fords om vi kan? *1 p*

(d) Visa hur vi kan representera grafen ovan i matrisform. *1 p*

Design

Uppgift 3

Vi har följande interface för en plotter:

```
interface Plotter {
    void setDrawMode();
    void setEraseMode();
    void line(int x0, int y0, int x1, int y1);
    void arc(int x, int y,
             int width, int height,
             double startAngle, double arcAngle);
}
```

Plottern kan antingen rita eller sudda – metoden `setDrawMode` gör att efterföljande rit-kommandon faktiskt ritas, `setEraseMode` gör att efterföljande rit-kommandon suddas (tills vi anropar `setDrawMode` igen).

För metoden `arc` är parametrarna `x` och `y` mittpunkten i en ellips med bredden `width` (i x-led) och höjden `height` (i y-led) – `startAngle` är start-vinkeln (i förhållande till x-axeln) och `arcAngle` är bågens längd (båda mäts i radianer)[2].

Vi vill nu skriva programkod för att rita på denna plotter, och vi vill använda *Command Pattern*. Vi vill ha kommandon för att rita cirklar och rektanglar, och vi vill att man även skall kunna ångra sina kommandon (figurerna skall då suddas bort).

(a) Definiera först ett lämpligt kommando-interface – observera att vi både vill kunna rita och sudda.

Definiera sedan klasser för rit-kommandon för cirklar och rektanglar – när man skapar sina rit-kommandon skall man ange figurernas mittpunkt, och man skall ge dem en bestämd storlek samt andra nödvändiga parametrar som kan förekomma (de skall dock inte ritas ut förrän man ber om det[3]).

För full poäng måste du använda *Template Method*, men du kan få (max) 50% av full poäng även för en lösning utan Template Method.

Svara med javakod för dina interface/klasser. *8 p*

(b) Rita ett klassdiagram för klasserna ovan (om du inte implementerade Template Method så behöver du inte rita den i diagrammet, men poängen reduceras på samma sätt som ovan). Alla attribut och metoder i dina interfaces och klasser skall finnas med i diagrammet (för de klasser du inte skriver själv räcker det att du visar att klassen finns, och anger de metoder i dem som du anropar). *5 p*

(c) Vi vill nu kunna logga aktiviteten på vår plotter. Visa med javakod och text hur vi kan använda *Decorator Pattern* för att vi skall få utskrifter till en bestämd `PrintStream` varje gång vi gör något med vår plotter (ändrar rit/sudd-mode, ritar eller suddar linjer, etc.) – du får inte lägga till kod för att lösa detta i dina klasser i uppgift (a).

Vi vill att du skall implementera en lämplig dekorator-klass. Svara med javakoden för denna. *5 p*

[2] Man kan tänka sig parametrar även för att vrida ellipsen kring mittpunkten, men vi nöjer oss i uppgiften med denna enklare form av ellipser

[3] Detta är en viktig del av mönstret *Command Pattern*.

Uppgift 4

Vi har bank-konton med följande attribut och konstruktor:

```
class Account {

    private String accountNumber;
    private String customerId;
    private String bankBranch;
    private double balance;
    private double interestRate;

    public Account  (String accountNumber,
                     String customerId,
                     String bankBranch,
                     double balance,
                     double interestRate) {
        this.accountNumber = accountNumber;
        this.customerId = customerId;
        this.bankBranch = bankBranch;
        this.balance = balance;
        this.interestRate = interestRate;
    }

    // ... omitted code ...
}
```

Använd *Builder pattern* för att ersätta konstruktorn ovan med ett säkrare och tydligare sätt att skapa konton (man skall bara kunna skapa konton med hjälp av vår builder). Svara med javakod för din uppdaterade Account-klass med den nya builder-funktionaliteten.

Visa också hur man använder buildern genom att skriva programkod som skapar ett konto med kontonummer `12-1334236`, kundnummer `645-564-654`, branch-namnet ”Lund”, saldot 2 015 045 SEK och räntan 4,0. *6 p*

Uppgift 5

(a) Förklara begreppen *Coherence* och *Coupling*. *2 p*

(b) Förklara mönstret *Observer/Observable* med hjälp av ett klassdiagram och en kortfattad beskrivning av hur det fungerar. *4 p*