

Omkontrollskrivning, Programmeringsteknik för D/C

2014-08-21, 14.00-19.00

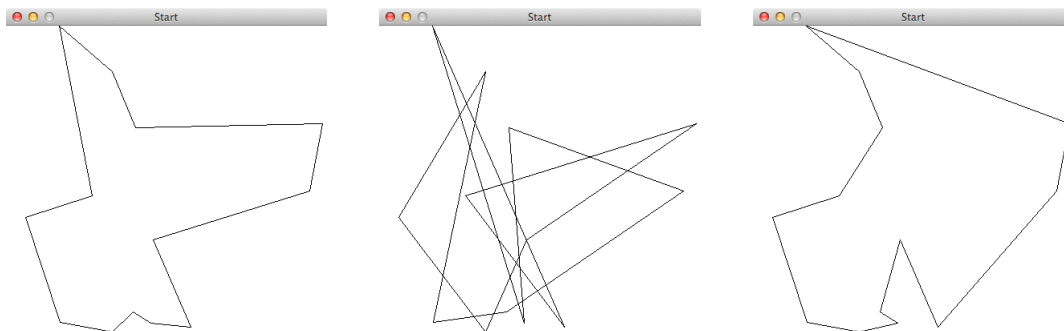
Anvisningar: Preliminärt ger uppgifterna $12 + 12 + 10 + 6 = 40$ poäng. För godkänt betyg krävs ungefär 25 poäng. Tillåtet hjälpmedel: Java-snabbreferens.

Betyget på denna skrivning utgör betyget på kontrollskrivningarna, oavsett resultatet på tidigare skrivningar (dock sänks inte ett tidigare godkänt betyg). Resultatlistan anslås på institutionens anslags-tavla när rättningen är klar.

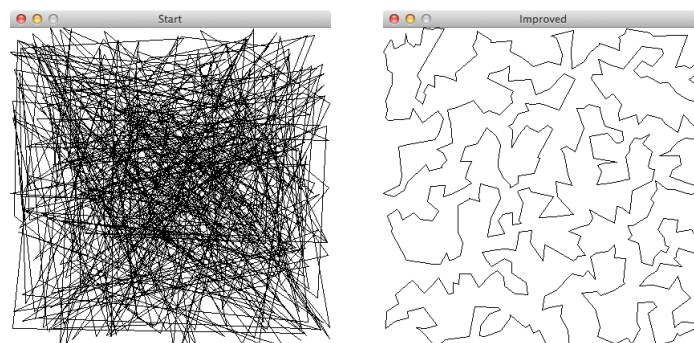
Introduktion

I handelsresandeproblemet (The Traveling Salesman Problem) ska man hitta den kortaste resvägen för en handelsresande som ska besöka ett antal städer. Varje stad får besökas bara en gång (men resan avslutas genom att man återvänder till startstaden). Vi förutsätter i fortsättningen att det finns vägar mellan alla städer och att avståndet mellan två städer är det euklidiska avståndet ("fågelvägen").

Det är tidskrävande att hitta den optimala lösningen till problemet, eftersom man måste gå igenom alla möjliga lösningar. För att lösa rimligt stora problem är man hänvisad till metoder som ger approximativa lösningar. Bilderna visar tre lösningar på ett problem med 13 städer: optimal, slumpmässig och "närmaste granne".



Det finns metoder som förbättrar en approximativ lösning. Man kan till exempel byta två vägar mot varandra (metoden, som kallas 2-opt, förklaras närmare i uppgift 4). Dessa bilder visar resultatet av denna förbättring av en slumpmässig lösning (500 städer):



1. Implementera en klass Map som beskriver en karta med ett antal städer. Klassen har följande specifikation:

```
/** Skapar en karta. Städernas koordinater läses in från filen med namnet
    fileName. Varje rad innehåller x- och y-koordinat för en stad */
Map(String fileName);

/** Returnerar antalet städer */
int getSize();

/** Returnerar staden med nummer i. Städerna numreras från 0 */
City getCity(int i);

/** Returnerar avståndet mellan stad nummer i och stad nummer j */
double getDist(int i, int j);
```

En stad beskrivs av den färdigskrivna klassen City:

```
City(double x, double y);
double getX();
double getY();
```

Implementera klassen Map. Metoden `getDist` kommer att anropas många gånger, och det skulle vara alltför tidsödande att beräkna avståndet vid varje anrop. Avstånden mellan städerna ska därför bara beräknas *en* gång och sparas i en matris, så att de kan hämtas i `getDist`.

Om filen med städernas koordinater inte kan öppnas ska programmet avbrytas med ett felmeddelande.

2. Skriv en klass Tour som representerar en lösning till handelsresandeproblemet. Klassen ska användas enligt följande exempel (fler metoder kommer att implementeras i de följande uppgifterna):

```
public static void main(String[] args) {
    Map map = new Map("mapdata.txt");
    Tour tour = new Tour(map);
    tour.createRandomTour(); // skapar en slumpmässig resväg
    Window w = new Window(400, 400, "Start"); // ritfönster
    tour.draw(w); // ritar upp resvägen
    System.out.println("Tour length: " + tour.getLength());
}
```

Resvägen *ska* representeras med en vektor med städernas nummer i den ordning de besöks. Den färdigskrivna klassen Window har bland annat en funktion `moveTo` som flyttar "pennan" till en punkt utan att rita och en funktion `lineTo` som flyttar pennan och ritar en linje (koordinatparametrarna är av typen `double` för att du ska slippa konvertera städernas koordinater till `int`):

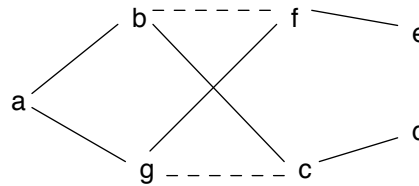
```
Window(double width, double height, String title);
void moveTo(double x, double y);
void lineTo(double x, double y);
```

I draw får du förutsätta att städernas koordinater håller sig inom fönstret.

3. Utöka klassen Tour med en metod som beräknar vägen med "närmaste granne"-metoden: välj en startstad, besök sedan närmaste stad som inte är besökt, besök sedan den närmaste staden till den nyss besökta, och så vidare tills alla städer är besökta.

4. Denna uppgift är främst avsedd för dig som eftersträvar betyg 5.

Skriv en metod i klassen Tour som förbättrar en lösning med 2-opt-metoden. Metoden går ut på att man upprepade gånger gör byten av två vägar, om bytet medför en förbättring. Exempel:



Om man här ersätter vägarna $b \rightarrow c$ och $f \rightarrow g$ med $b \rightarrow f$ och $c \rightarrow g$ blir resvägen kortare. Den ursprungliga resvägen $a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \rightarrow f \rightarrow g$ blir alltså $a \rightarrow b \rightarrow f \rightarrow e \rightarrow d \rightarrow c \rightarrow g$ (städerna i vägavsnittet $c \rightarrow d \rightarrow e \rightarrow f$ besöks i omvänd ordning i den nya resvägen).

Nedanstående algoritm ska upprepas så länge man har gjort något byte av vägar:

```
för i = 0, 1, ...
  för j = i + 2, i + 3, ...
    väg1 = vägen mellan stad nr i och stad nr i + 1      (b->c i exemplet)
    väg2 = vägen mellan stad nr j och stad nr j + 1      (f->g i exemplet)
    nyväg1 = vägen mellan stad nr i och stad nr j        (b->f i exemplet)
    nyväg2 = vägen mellan stad nr i + 1 och stad nr j + 1 (c->g i exemplet)
    Om de nya vägarna tillsammans är kortare:
      byt vägarna
```