Tentamen
C++-programmering

2014–08–29, 14.00–19.00

Hjälpmedel: En valfri C++-bok. OH-bilderna från föreläsningarna är *inte* tillåtna.

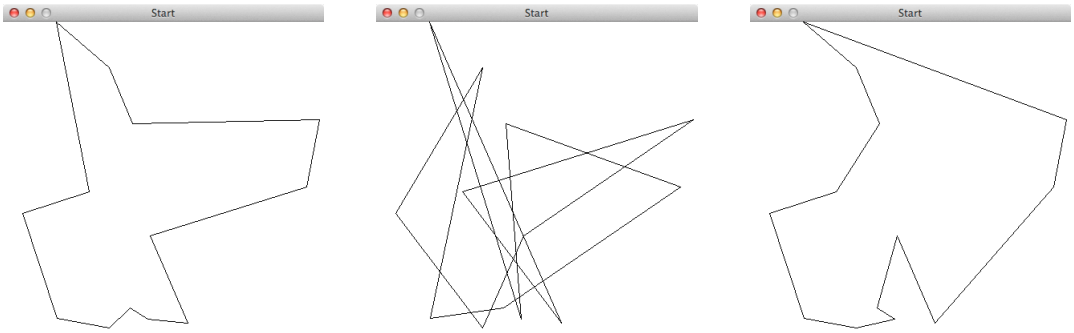
Du ska i dina lösningar visa att du behärskar C++ och att du kan använda C++ standardklasser. ”C-lösningar” ger inga poäng, även om de är korrekta.

Uppgifterna ger preliminärt $12 + 12 + 7 + 4 + 7 + 8 = 50$ poäng. För godkänt krävs 25 poäng (3/25, 4/33, 5/42).

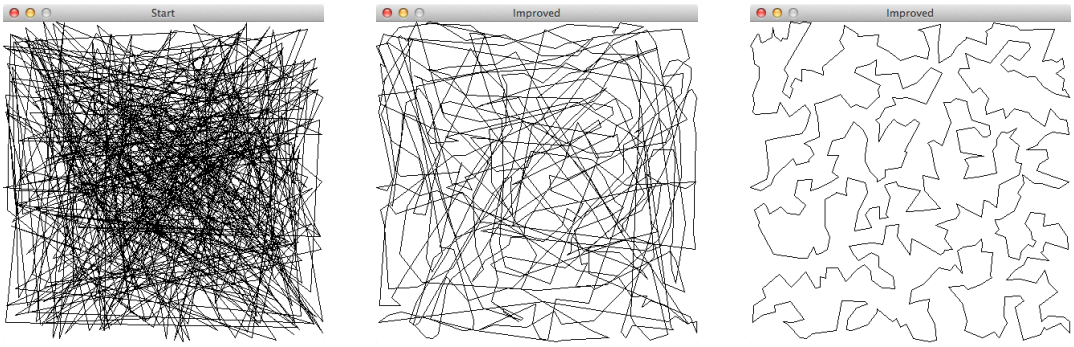
Introduktion

I handelsresandeproblemet (TSP, ”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. 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 städer mot varandra eller byta två vägar mot varandra (metoderna förklaras närmare senare). Dessa bilder visar resultatet av dessa förbättringar av en slumpmässig lösning (500 städer):



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

```
class Map {
public:
    Map(size_t n);
    Map(const string& file_name);
    size_t size() const;
    pair<double, double> get_coords(size_t i) const;
    double get_dist(size_t i, size_t j) const;
};
```

   Den första konstruktorn skapar en karta med `n` städer med slumpmässiga koordinater i intervallet $[0, 1]$. Den andra konstruktorn läser kartdata från en fil där varje rad innehåller x- och y-koordinat för en stad. Skala de inlästa koordinaterna så att de hamnar i intervallet $[0, 1]$.

   `get_coords` returnerar x- och y-koordinaten för staden med nummer `i` (städerna numreras från noll och uppåt). `get_dist` returnerar avståndet mellan stad nummer `i` och stad nummer `j`. Avstånden mellan städerna får bara beräknas *en* gång; det skulle bli alltför tidsödande annars.

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

```
int main() {
    Map map(50);
    Tour tour(map);
    tour.create_random_tour();   // create a random tour
    Window w(400, 400, "Start"); // a window to draw in
    tour.draw(w);                // draw the tour
    cout << "Tour length: " << tour.get_length() << endl;
}
```

   Resvägen *ska* representeras med en vektor med städernas nummer i den ordning de besöks. Klassen `Window` har bland annat en funktion för att rita en linje från en punkt `(x1,y1)` till en annan punkt `(x2,y2)`:

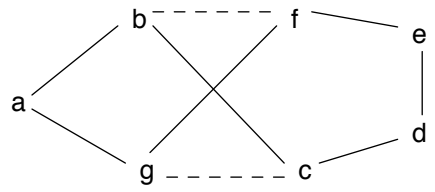
```
class Window {
public:
    Window(int width, int height, const char* title);
    int get_width() const;
    int get_height() const;
    void line(int x1, int y1, int x2, int y2);
};
```

3. Utöka klassen `Tour` med en funktion 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. Utöka klassen `Tour` med en funktion som beräknar den *optimala* vägen genom att undersöka alla möjliga resvägar (det vill säga alla möjliga kombinationer av städer i vektorn som representerar resvägen). Detta ska vara en enkel uppgift — man kan utnyttja en standardalgoritm. (Man kan bara lösa mycket små problem med den här metoden.)

5. Skriv en funktion i klassen `Tour` som förbättrar en lösning genom att slumpmässigt byta två städer mot varandra och se om den nya vägen är bättre. Gör detta några tusen gånger.

6. Skriv en funktion 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->c` och `f->g` med `b->f` och `c->g` blir resvägen kortare. Den ursprungliga resvägen `a->b->c->d->e->f->g` blir alltså `a->b->f->e->d->c->g`.

Man ska kontrollera alla par av vägar som inte har någon gemensam stad och byta vägarna om det innebär en förbättring. Detta håller man på med tills det inte längre blir någon förbättring. Observera att man kan beräkna huruvida ett byte medför en förbättring utan att göra bytet.