Lektion 1

Kattis

I kursen används onlinedomaren ***Kattis*** (från `http://kattis.com`) för att automatiskt rätta programmeringsproblem. För att få ett konto på Kattis anmäler du dig på Programmeringsolympiadens hemsida, `http://progolymp.se/elev/anmal`.

Den instans av Kattis vi kommer använda hittar du på `https://po.kattis.com`. Där finns också dokumentation om hur du skickar in lösningar till problem. Du ska också anmäla dig till kursen 'Algoritmer 2014-2015 (Matematikgymnasiet)' under *Kurser* i menyn.

En ***Kattisövning*** är en övningsuppgift som du kan skicka in din lösning till på Kattis. En ***Kattis-inlämning*** är en inlämingsuppgift som du får betygspoäng på beroende på hur mycket poäng din lösning fick på Kattis (en lösning kan ge olika mycket poäng beroende på hur bra den var). Varje Kattisproblem har ett ***problem-id***. För att se problemet kan du antingen söka efter det eller skriva in `https://po.kattis.com/problems/problemid`.

I Kattis får ska ditt program läsa all indata från standard in (scanf/cin/System.in/stdin/etc) och skriva utdatan till standard out (printf/cout/System.out/stdout). På din egen dator kommer detta vanligtvis vara samma sak som att läsa in och skriva ut data till terminalen.

Algoritmer och problem

En ***algoritm*** är en metod som tar emot någon mängd värden (det som kallas för ***indata***) för att producera nya värden, den så kallade ***utdatan***. Oftast använder vi algoritmer för att lösa olika sorters ***problem*** som uppkommer inom datavetenskapen.

Figur 1: indata kommer in, utdata kommer ut

Ett problem kan exempelvis vara att beräkna den största gemensamma delaren av två heltal A och B, som betecknas $\gcd(A, B)$ (greatest common divisor på engelska). Ett tal N är ett delare till ett tal A om kvoten $\frac{A}{N}$ är lika med ett heltal, dvs $A = N \cdot x$ för något heltal x. Vi beskriver problemet formellt:

Exempel 1. Största gemensamma delaren

Indata: två positiva heltal $B \geq A > 0$.

Utdata: den största gemensamma delaren av A och B, dvs $\gcd(A, B)$.

Lösning

För att en algoritm ska lösa ett problemet behöver den uppfylla två villkor:

1. Algoritmen får inte köra oändligt länge (dvs den måste ***terminera***)
2. När algoritmen terminerar måste den ha beräknat den korrekta utdatan

Till problemet **Största gemensamma delaren** finns en tämligen enkel algoritm som redan den gamle greken Euklides kom på, den så kallade ***Euklides algoritm***.

Euklides algoritm tar två positiva heltal A och B. Sedan, ända tills ett av talen blir 0, subtraherar man det mindre av talen från det större. När ett av talen blir 0 kommer det andra talet vara $\gcd(A, B)$. Vi kan specifiera algoritmen mer exakt med hjälp av ***psuedokod***:

Algoritm 1 Euklides algoritm

Antagande: $B \geq A \geq 0$

1: **Funktion** GCD(heltal A, heltal B)
2: **så länge** $A > 0$
3: $B \leftarrow B - A$ ▷ Subtrahera det mindre talet från det större
4: **om** A är större än B
5: byt plats på A och B ▷ Vi vill att $B \geq A$ alltid ska vara sant
6: **returnera** B

Det är fortfarande långt ifrån uppenbart varför algoritmen faktiskt fungerar, eller om den ens terminerar. Vi kan prova att exekvera algoritmen för oss själva, t.ex. med talen 15 och 6.

$B = 15, A = 6$	(start)
$B = 9, A = 6$	(subtrahera)
$B = 3, A = 6$	(subtrahera)
$B = 6, A = 3$	(byt plats)
$B = 3, A = 3$	(subtrahera)
$B = 0, A = 3$	(subtrahera)
$B = 3, A = 0$	(byt plats)
$gcd(15, 6) = 3$	(terminera)

Algoritmen fungerade onekligen för vårt lilla exempel - 3 är det största heltal som delar både 15 och 6. Eftersom vi är matematiker nöjer vi oss såklart inte med detta, men kanske har vi nu fått ett hum om varför algoritmen är korrekt:

1. Betrakta summan $A + B$. Efter en körning av loopen kommer summan minska med A. Men $A > 0$, eftersom loopen inte ännu avslutats. Alltså kommer summan $A + B$ bli strikt mindre. Men inget av talen är någonsin negativt, så $A + B \geq 0$. Summan kan alltså inte minska i all oändlighet, och för eller senare måste därför A vara 0 så att loopen avbryts.

2. Vi noterar först att $\gcd(B, 0) = B$. Vidare måste vi inse att ett tal D är en gemensam delare till A och B **om och endast om** D är en gemensam delare till talen A och $B - A$. Detta innebär rent konkret att efter varje steg i vår algoritm kommer de gemensamma delarna till talen vara oförändrade. I synnerhet kommer den *största* gemensamma delaren vara densamma.

   Vi vet att variabeln A förr eller senare kommer bli 0 i vår algoritm måste, vi vet att $\gcd(B, 0) = B$, samt att vi har samma gemensamma delare i början av algoritmen som i slutet. Vår slutledningsförmåga konstaterar då att talens största gemensamma delaren helt enkelt är variabeln B:s slutvärde.

Vi har alltså konstaterat dels att algoritmen terminerar, men också att den gör så med rätt svar - algoritmen är alltså ***korrekt***.

Den uppmärksamma läsaren kanske ifrågasatte att vi betraktade problemet som löst utan att ta någon som helst hänsyn till *hur lång tid* algoritmen tar innan den terimerar. Vår algoritm kanske är så ineffektiv att det tar flera dagar att beräkna svaret när A och B är stora. Under nästa lektion kommer vi behandla de verktyg man använder för att beskriva hur effektiv en algoritm är.

Övning 1. Formulera problemet **Polynomrötter**, som går ut på att finna alla reella nollställen till ett polynom.

Övning 2. Bevisa att D är en gemensam delare till A och B **om och endast om** D är en gemensam delare till talen A och $B - A$.

Övning 3. Bevisa att Euklides algoritm terminerar efter högst $A + B$ iterationer av loopen.

Kattisövning (ID: gcd)**.** Implementera Euklides algoritm.

Implementationsproblem

I den "enklaste" sortens problem är problemlydelsen så pass specifik att det svåra inte är att komma på lösningen, utan att faktiskt implementera den. Ofta handlar implementationsproblem om att simulera något tidsförlopp.

Exempel 2. Receptet (Programmeringsolympiadens skolkval 2011)

> Du har bestämt dig för att laga mat. För att laga maten behöver du N ingredienser. För varje ingrediens vet du hur mycket du redan har hemma, hur mycket du behöver totalt samt kostnaden för ingrediensen om du måste köpa den. Du skall alltså köpa den mängd av varje ingrediens som du saknar. Uppgiften är att beräkna kostnaden för att laga maten.

Kattisövning (ID: receptet)**.** Lös Receptet.

Problemet är inte särskilt svårt: för varje ingrediens beräknar hur mycket av varje ingrediens vi måste köpa. Sedan multiplicerar vi detta med priset för ingrediensen, Till slut summerar vi kostnaden för varje ingrediens.

Bara för att lösningen till ett implementationsproblem är uppenbar behöver den absolut inte vara enkel! Att implementera en algoritm kan vara extremt invecklat och öppnar upp för många buggar i ditt program.

Bruteforce

Många problem går ut på att prova en stor mängd möjligheter i lösningen. Om uppgiften är att man ska skriva ut antalet heltalslösningar till en ekvation kan det tänkta tillvägagångssättet helt enkelt vara att prova alla möjliga värden på ekvationens variabler och se om dessa uppfyller ekvationen. Detta kallas för ***bruteforce***. Man använder här att en dator är väldigt snabb, så att man behöver tänka mindre i själva lösningen.

Ibland är användningen av bruteforce inte så uppenbar. Det kan t.ex. vara en del i ett problem. Ekvationen vi letar heltalslösningar till kanske blir mycket lättare att lösa så länge vi känner till *en* av ekvationens variabler för att de övriga faller ut på ett naturligt sätt.

Exempel 3. Finn alla heltal $0 < x < 10^9$ som uppfyller

$$x = a \cdot s(x)^b + c$$

där

$$|a| \leq 10\,000$$
$$1 \leq b \leq 5$$
$$|c| \leq 10\,000$$

är givna heltal, och $s(x)$ är siffersumman av talet x.

För att lösa ekvationen enklast använder man bruteforce. Det vi gör bruteforce över är inte x, som man skulle kunna tro. x kan nämligen anta alldeles för många värden för att programmet ska bli snabbt nog (en dator klarar av ungefär 10^8 "små" operatoner per sekund). Istället inser vi att ett tal med högst 9 siffror inte kan ha en större siffersumma än 81, nämligen då alla siffror är 9. Det vi

ska bruteforca över är alltså siffersumman $s(x)$. Givet en viss siffersumma är x entydigt bestämt eftersom vi kan beräkna högerledet. Då återstår bara att kontrollera om x faktiskt har samma siffersumma som vi antog att talet hade.

Att lösa ett problem med ren bruteforce, det vill säga utan att kombinera med någon mer avancerad teknik eller ytterligare insikter (som ovan, där det svåra var att komma fram till rätt variabel) brukar oftast bara vara möjligt på enklare problem. Generellt blir programmet alldeles för långsamt.

Lösningar till övningarna

Övning 1:

Indata: Koefficienterna a_i till ett polynom $p(x) = a_n x^n + a_{n-1}x^{n-1} + ... + a_0$.

Utdata: Alla reella tal x sådana att $p(x) = 0$.

Övning 2: Om D är en delare i A och B så kan vi skriva $A = DN$ och $B = DM$ för några heltal N och M. $B - A = D(M - N)$, vilket såklart är delbart med D.

Åt andra hållet, om D är en delare i B och $B - A$ kan vi skriva $B = DN$ och $B - A = DM$. Detta ger $A = B - DM = DN - DM = D(N - M)$, vilket även det är delbart med D.

Övning 3: Varje iteration av loopen minskar summan $A + B$. Detta kan bara hända $A + B$ gånger, eftersom summan är icke-negativ. Händer det fler gånger måste minst ett av talen bli 0, och då kommer loopen ha terminerat.

Kattisövning gcd:

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

int gcd(int A, int B){
    while(A > 0){
        B = B - A;
        if(A > B){
            swap(A, B); // swap finns i <algorithm>
        }
    }
    return B;
}

int main(){
    int A, B;
    cin >> A >> B;
    cout << gcd(A, B) << endl;
}
```

Inlämningsproblem

Kattisinlämning 1 (ID: bijektion)**.** 1/0/0

Kattisinlämning 2 (ID: fullsatt)**.** 2/0/0

Kattisinlämning 3 (ID: kalkylator)**.** 0/1/0

Kattisinlämning 4 (ID: schackmatt)**.** 0/0/1

Teoriuppgift 1. 2/1/0

a) Formulera **Sorteringsproblemet** som går ut på att sortera en lista av heltal.

b) Skriv en lösning till **Sorteringsproblemet** i pseudokod.

c) Bevisa att din lösning är korrekt och terminerar.

d) Nämn några tillämpningar av **Sorteringsproblemet**.

Teoriuppgift 2. 1/1/0

När vi vill bevisa att en algoritm inte är korrekt försöker vi hitta ett ***motexempel*** till algoritmen. Det räcker med ett enda fall där algoritmen inte fungerar för att den ska vara felaktig, oavsett hur många exempel den faktiskt klarar.

Följande problem (Biblioteket) kommer från Programmeringsolympiadens onlinekval 2009, i lite modifierad form:

> Du jobbar på ett bibliotek och vill ställa tillbaka ett antal böcker i hyllorna. Hyllorna är placerade längs x-axeln. Givet i vilken hylla varje bok ska stå (en x-koordinat a_i mellan 0 och 1000) och det maximala antalet böcker som du kan bära samtidigt, bestäm den kortaste sträckan du måste gå för att ställa tillbaka alla böcker. Du och böckerna som ska ställas tillbaka befinner sig ursprungligen på position 0. Du behöver inte gå tillbaka efter att ha återställt den sista boken. Det finns total n böcker, och du kan bära k böcker samtidigt.

Mårten föreslår följande lösning:

Börja med att lägga tillbaka de k böcker som ska placeras närmast utgångspunkten. Sedan tar du de nästa k närmsta böckerna, och fortsätter så tills du placerat ut alla böcker.

Visa att Mårtens lösning är felaktig genom att konstruera ett motexempel.