Föreläsning 1

Rekursion

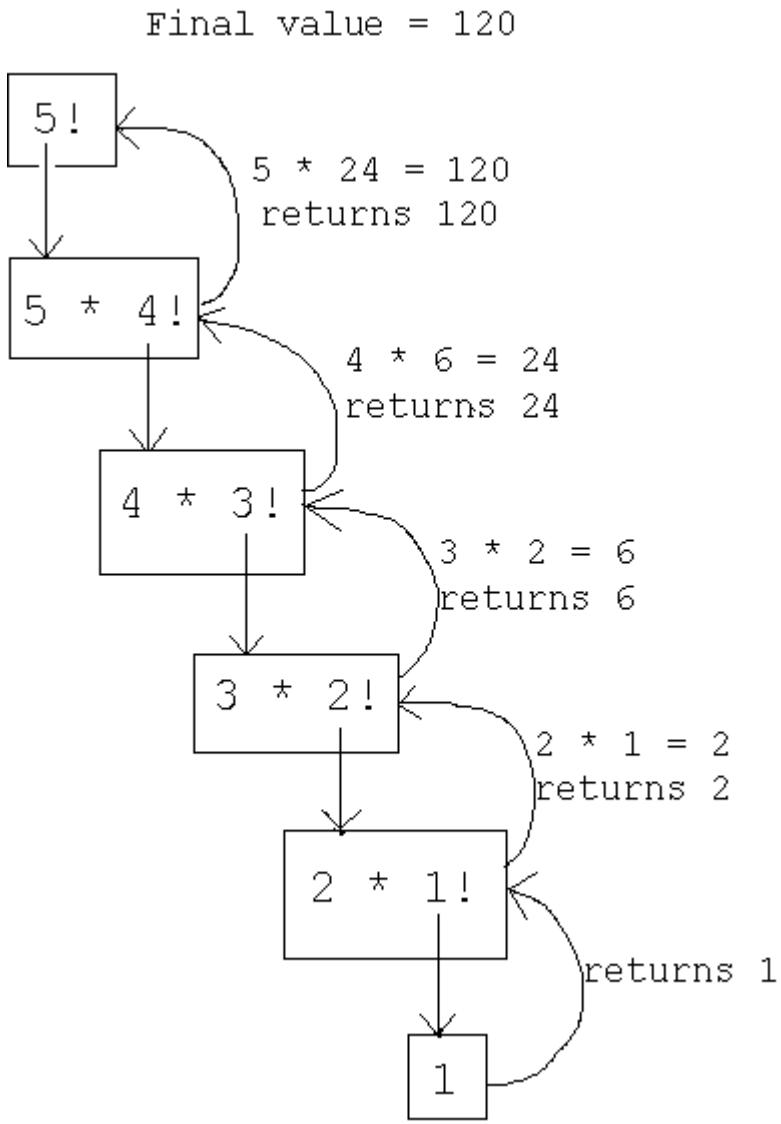
"För att förstå rekursion måste man förstå rekursion." - okänd

Så vad är egentligen rekursion i ett datorprogram?

En subrutin som anropar sig själv

Direkt eller indirekt

Factorial som exempel…



Användningsområde

Iterera

Det finns språk där rekursion är enda möjligheten att upprepa en sekvens med kod. En for-loop skrivs enkelt om som en rekursion.

Lösa vissa typer av problem

Problemet är ofta svårt att lösa men kan successivt reduceras till mindre problem genom rekursiva anrop.

Java-exempel 1

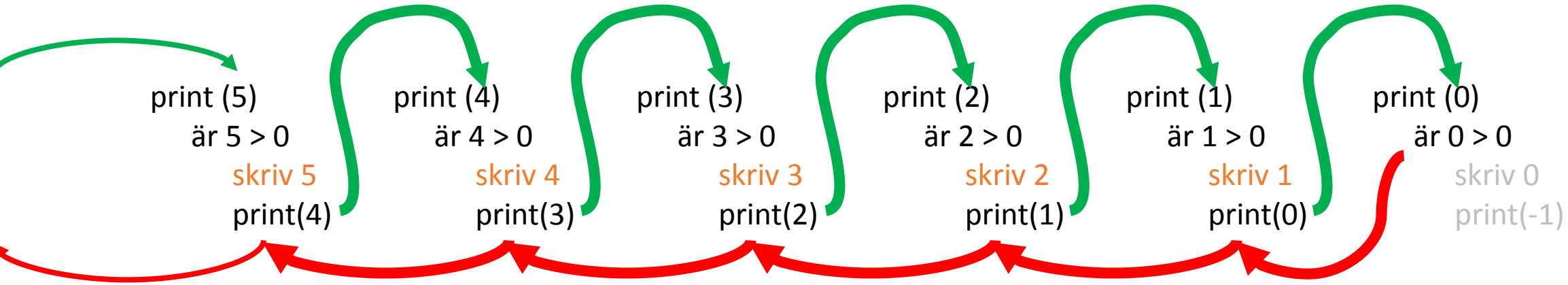
En metod som anropas sig själv kallas en ***rekursiv metod*** som gör ett ***rekursivt anrop***.

Metoden **print** är en rekursiv metod.

Vad blir körresultatet när metoden anropar sig själv så här: **print(5);** ?

```
public void print(int n) {                  ?
    if (n > 0) {                            ?
        System.out.println(n);              ?
        print(n - 1); // rekursivt anrop    ?
    }                                       ?
}
```

Java-exempel 1, svar




```
public void print(int n) {                5
    if (n > 0) {                          4
        System.out.println(n);            3
        print(n - 1); // rekursivt anrop  2
    }                                     1
}
```

Java-exempel 2

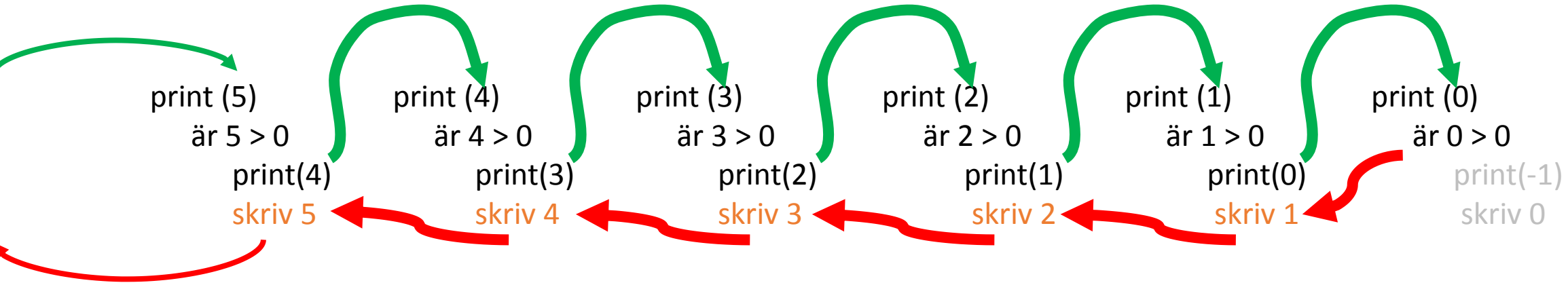
Vad blir körresultatet av **print2(5)**?

Bytt plats på **print(n - 1)** samt **System.out.println(n)**

```
public void print(int n) {                5
    if (n > 0) {                          4
        System.out.println(n);            3
        print(n - 1); // rekursivt anrop  2
    }                                     1
}
```

```
public void print2(int n) {          ?
    if (n > 0) {                     ?
        print2(n - 1);               ?
        System.out.println(n);       ?
    }                                ?
}
```

Java-exempel 2, svar




```
public void print(int n) {              5
    if (n > 0) {                        4
        System.out.println(n);          3
        print(n - 1); // rekursivt anrop 2
    }                                   1
}
```

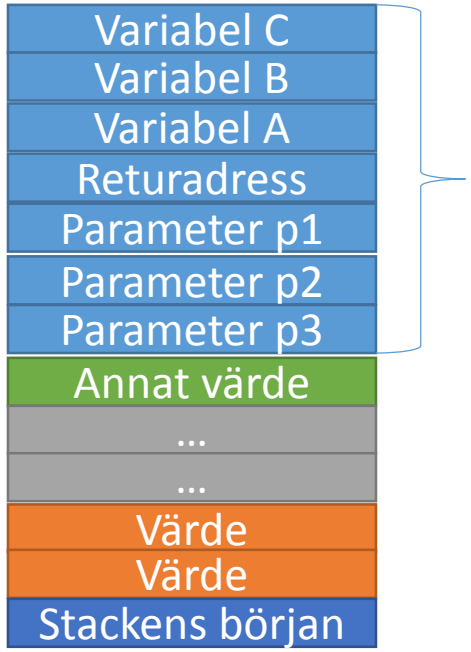
```
public void print2(int n) {             1
    if (n > 0) {                        2
        print2(n - 1);                  3
        System.out.println(n);          4
    }                                   5
}
```


MALMÖ HÖGSKOLA

Stacken vid anrop av metoder

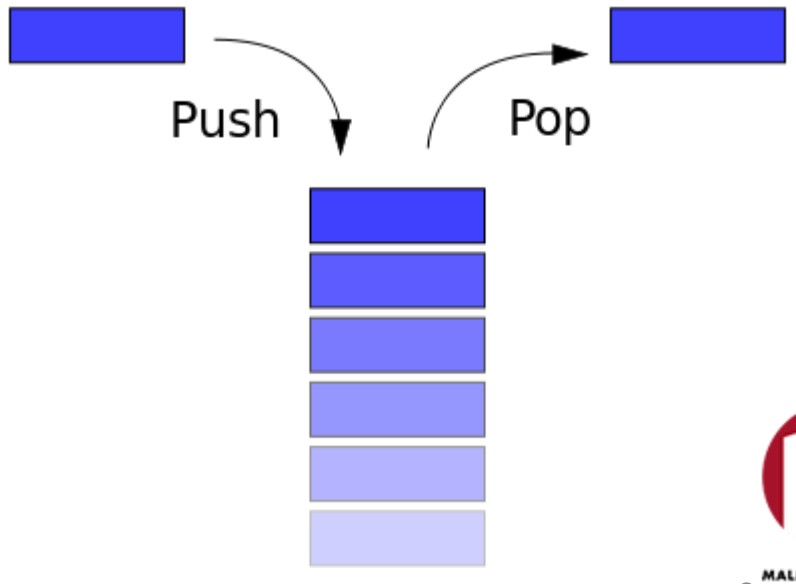
Stacken är ett minnesutrymme som lagrar data vid metodanrop.

- **Platsen i programmet** där anropet till metoden sker (returadress).
- **Parametrarna** i metodens parameterlista.
- **Lokala variabler** som finns i metoden.



Stack frame av func

```
int func(int p1, int p2, int p3)
{
    int A, B, C;
    ...
}
```



Stacken vid metodanrop

Metoden **neverStop** är en rekursiv metod eftersom den anropar sig själv.

Vad blir körresultatet vid anropet **neverStop(10);** ?

```
public void neverStop(int number) {
    System.out.println(number);
    neverStop(number + 1);
}
```

Exception in thread "main" java.lang.StackOverflowError
…

Iterera

Rekursion kan användas för att upprepa en kodsekvens ett antal gånger.

```
public void iter( int n ) {
   if( n > 0 ) {
      // kod att upprepa
      iter( n – 1 );
   }
}
```

Exempel 1: Skriv ett tecken ett antal gånger

```
public void fill( char chr, int n ) {
   if( n > 0 ) {
      System.out.print( chr );
      fill( chr, n – 1);
   }
}

fill( '-', 66 );
```

Exempel 2: Skriv ut tecknen i en sträng med start i en angiven position

```
public void print( String str, int pos ) {
   if( ( pos >= 0 ) && ( pos < str.length() ) ) {
      System.out.print( str.charAt( pos ) );
      print( str, pos + 1 );
   }
}
```

Skillnad?

```
public void print( String str, … ) {
   if( ( pos >= 0 ) && … ) {
      print( str, pos + 1 );
      System.out.print( str … );
   }
}
```

Iterera genom en array

Rekursion kan användas för att gå igenom en array element för element.
Men det behövs en parameter för att hålla reda på positionen,

```
public void positive( int[] array , int pos ) {
   if( ( pos  >= 0 ) && ( pos < array.length ) ) {  // giltig position
      if( array[ pos ] > 0 )
         System.out.print( array[ pos ] + ” ” );
      positive( array, pos + 1 );
   }
}
```

eller

```
public void positive( int[] array ) {
   positive( array, 0);
}

private void positive( int[] array, int pos ) {
   if( pos < array.length) {
      if( array[ pos ] > 0 )
         System.out.print( array[ pos ] + ” ” );
      positive( array, pos + 1 );
   }
}
```

Rekursion för att lösa problem

Problem: Summera talen 5, 4, 3, 2, 1, dvs från 5 och nedåt

Lösning:

Steg 1: Addera 5 med summan av övriga tal (1, 2, 3, 4)
Steg 2: Addera 4 med summan av övriga tal (1, 2, 3)
Steg 3: Addera 3 med summan av övriga tal (1, 2)
Steg 4: Addera 2 med summan av övriga tal (1)
Steg 5: Summan är 1
Forts steg 4: Summan är 2 + 1 = 3
Forts steg 3: Summan är 3 + 3 = 6
Forts steg 2: Summan är 4 + 6 = 10
Fort steg 1: Summan är 5 + 10 = 15

Java: (anrop sum(5);)

```
public int sum( int n ) {
   if( n == 1 )
      return 1;
   else
      return n + sum( n – 1 );
}
```

Rekursion för att lösa problem

Java: (anrop sum(5);)

```
public int sum( int n ) {
   if( n == 1 )
      return 1;
   else
      return n + sum( n – 1 );
}
```

På problemet **sum(5)** får vi:

Svaret är **5 + sum(4)**

På problemet sum(4) får vi:

Svaret är **4 + sum(3)**

På problemet sum(3) får vi:

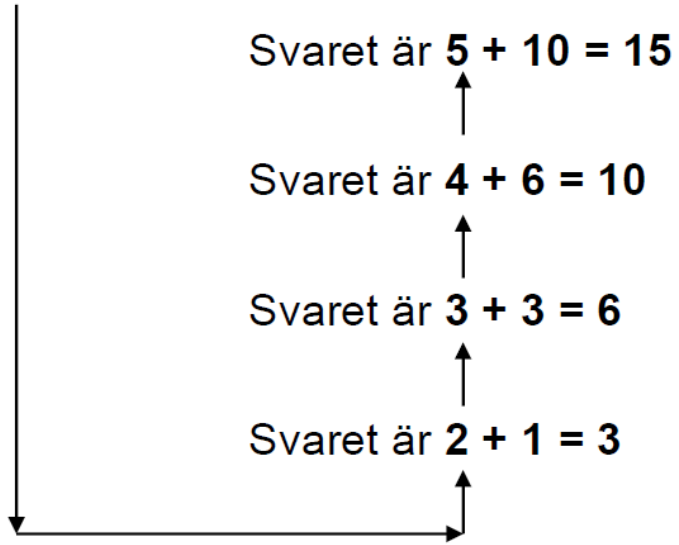
Svaret är **3 + sum(2)**

På problemet sum(2) får vi:

Svaret är **2 + sum(1)**

På problemet sum(1) får vi:

Svaret är 1 och nu kan programmet slutföra ovanstående beräkningar



Rekursion för att lösa problem

Java:

```
public int sum( int n ) {
   if( n == 1 )
      return 1;
   else
      return n + sum( n – 1 );
}
```

Metoden summera innehåller 2 fall där en if-else-sats avgör vilket fall som ska exekveras.

1. Basfallet – det enkla problemet

Då villkoret är sant, n == 1, så är problemet enkelt, summan av talet 1 är ju 1. Metoden returnerar därmed ett svar direkt, nämligen 1.

Inget nytt rekursivt anrop sker.

2. Rekursiva fallet - att förenkla problemet

Om n > 1, så är inte problemet enkelt och därför sker ett rekursivt anrop med ett problem som är lite enklare att lösa.

Alla rekursioner måste innehålla en selektion som avgör om det rekursiva fallet ska användas.

Rekursion för att lösa problem

Problem: Talen 17, 11, 9, 16 lagras i arrayen a. Summera talen.

Lösning:

Steg 1: Addera 17 med summan av övriga tal (11, 9, 16)

Steg 2: Addera 11 med summan av övriga tal (9, 16)

Steg 3: Addera 9 med summa av övriga tal

Steg 4: 16 (det sista talet i arrayen)

Forts steg 3: 9 + 16 = 25

Forts steg 2: 11 + 25 = 36

Fort steg 1: 17 + 36 = 53

Java: (summaArray(a, 0);)

```
public int sumArray(int[] array, int pos) {
   if( pos == array.length – 1 )
      return array[ pos ];
   else
      return array[ pos ] + sumArray(array, pos + 1);
}
```

Rekursion för att lösa problem

Problem: Finns ett speciellt tecken i en sträng?

Lösning:

Om strängen är tom
  returnera false
Om första tecknet i strängen är det sökta tecknet så
  returnera true
Annars
  returnera svaret vid kontroll av resten av strängen

Java:

```
public boolean member( char chr, String str ) {
   if( str.length() == 0)
      return false;
   else if( chr == str.charAt(0) )
      return true;
   else
      return member( chr, str.substring(1));
}
```

Självliknande figurer

Figuren till höger är uppbyggd av:

1 grön rektangel

4 ljusblå rektanglar

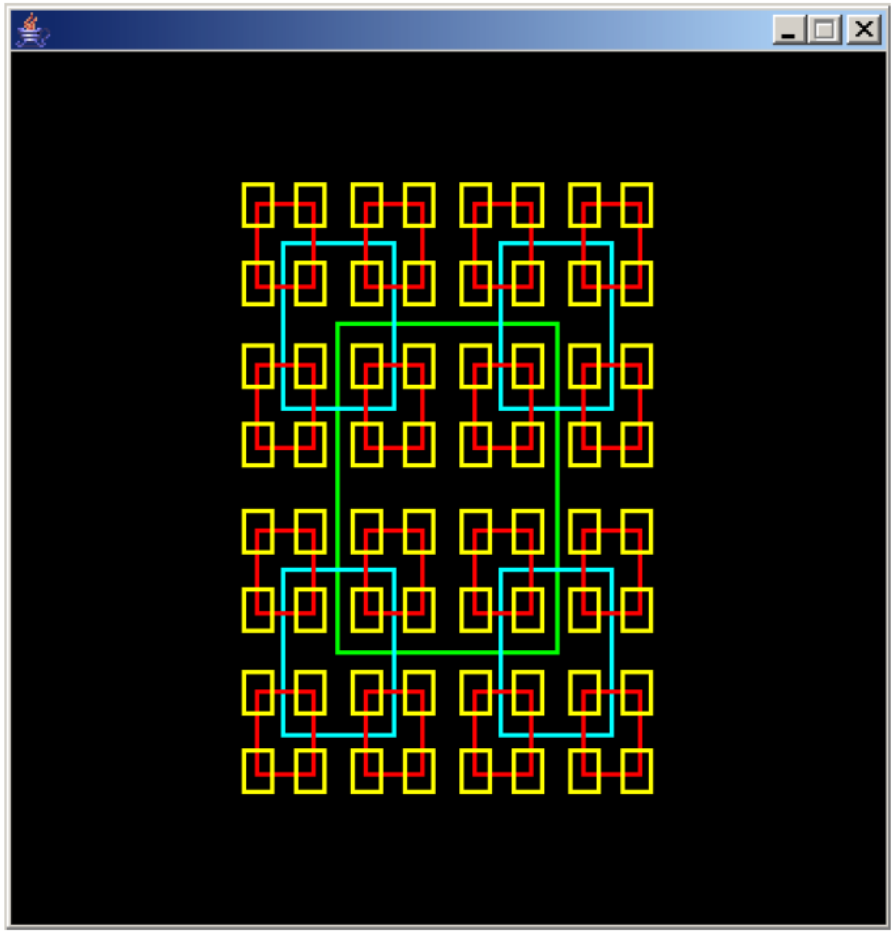
16 röda rektanglar

64 gula rektanglar

En rekursiv metod ritar figuren, **paintRectangles(…),**

Så här är metoden uppbyggd:

Om fler rektanglar ska ritas så
- Rita en rektangel
- Rekursivt anrop
- Rekursivt anrop
- Rekursivt anrop
- Rekursivt anrop

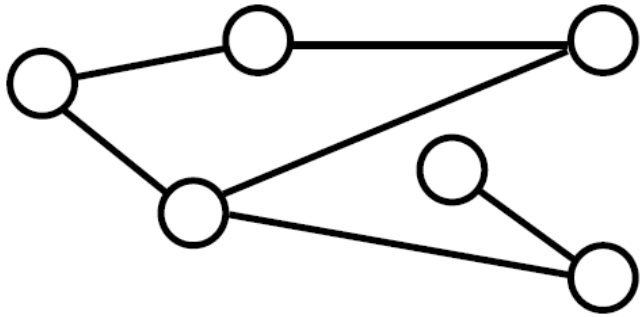


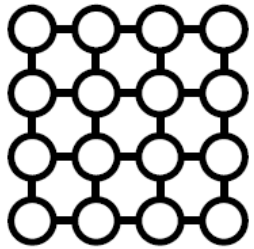
Vid varje rekursivt anrop ritas en mindre rektangel. Dessa ges olika positioner, nämligen runt hörnen.

Den gröna rektangeln ritas, sedan sker fyra rekursiva anrop varvid de ljusblå rektanglarna ritas. De ljusblå genererar de röda osv.

Rekursion - sökning

Med hjälp av rekursion kan man söka i olika datastrukturer, t.ex. i grafer:





Exempel: Beräkna minsta summan från övre vänstra hörnet till nedre högra hörnet. Det är endast tillåtet att gå nedåt och åt höger.

I figuren till höger är den lägsta summan beräknad och en tänkbar väg rödmarkerad.

Frågor?