Algoritmer, datastrukturer och komplexitet

Övning 3

Anton Grensjö
grensjo@csc.kth.se

20 september 2015

Kursplanering

F8: Dekomposition

F9: Dynamisk programmering

Ö3: Dekomposition och DP

Bonusdatum Labb 1

F10: Dynamisk programmering

F11: Korrekthet

F12: Grafer: MST och Dijkstra

Ö4: Dynamisk programmering

Idag

- Från förra veckan:
  - Exempel på BFS och DFS.
- Dekomposition
- Dynamisk programmering

Exempel på BFS och DFS

Pseudokod för BFS

function BFS(G, v)
    Låt Q vara en kö
    $Q.push(v)$
    while Q inte är tom **do**
        $v = Q.pop()$
        Gör saker med v.
        for varje granne w till v i G **do**
            if w inte är upptäckt än **then**
                Markera w som upptäckt
                $Q.push(w)$

Exempel på BFS och DFS

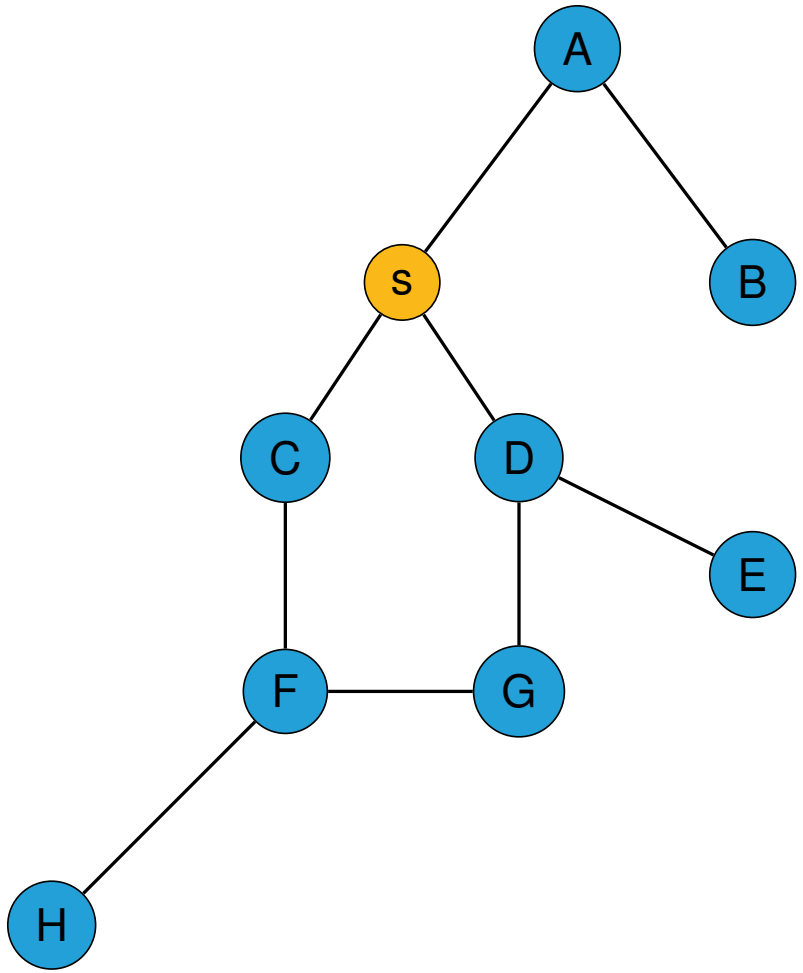
Pseudokod för DFS

function DFS(G, v)
    Låt S vara en stack
    $S.push(v)$
    while S inte är tom **do**
        $v = S.pop()$
        if v inte redan är besökt **then**
            Markera v som besökt.
            Gör saker med v.
            for varje obesökt granne w till v i G **do**
                $S.push(w)$

Exempel på BFS och DFS

BFS

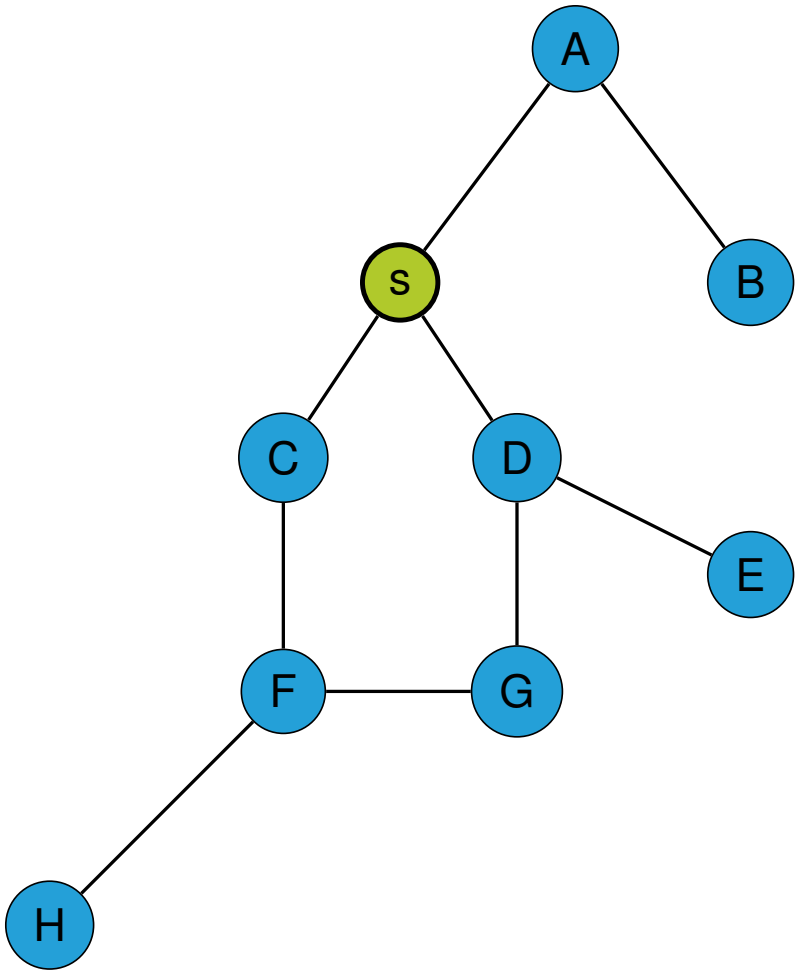
s



Exempel på BFS och DFS

BFS

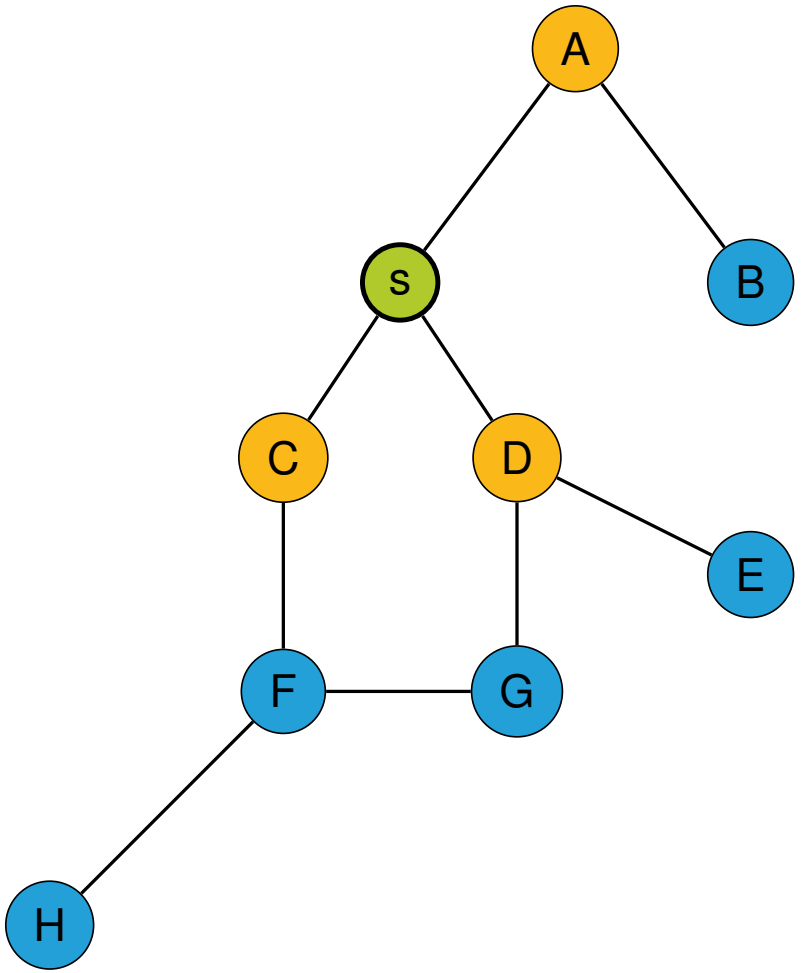
□



Exempel på BFS och DFS

BFS

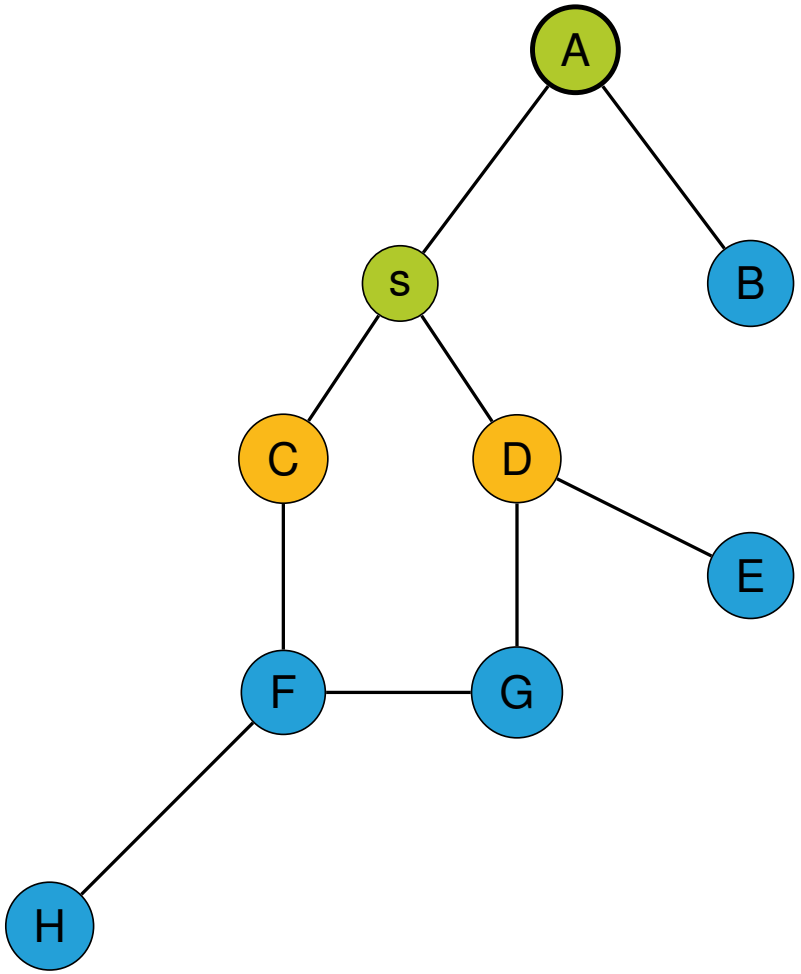
A
C
D



Exempel på BFS och DFS

BFS

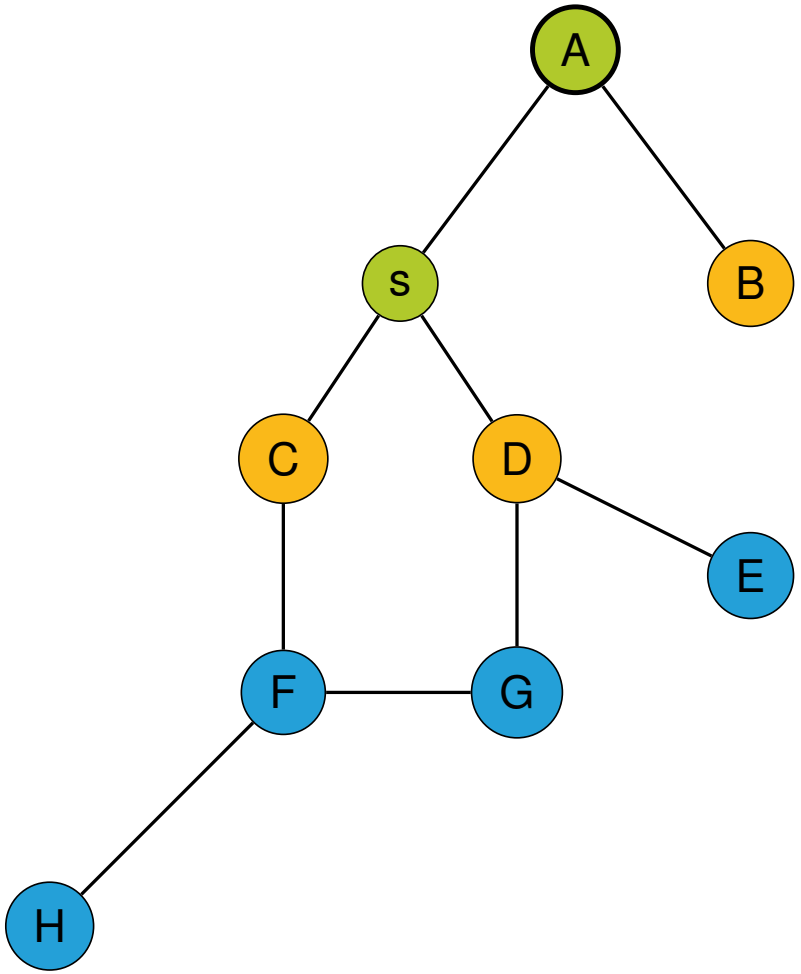
C
D



Exempel på BFS och DFS

BFS

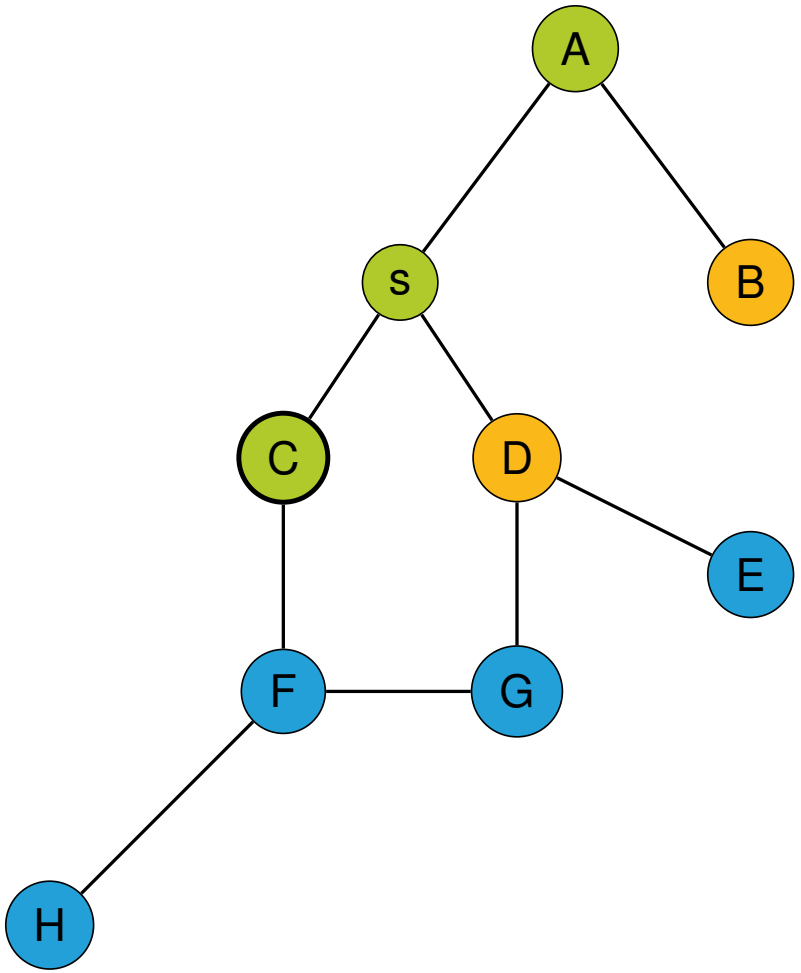
C
D
B



Exempel på BFS och DFS

BFS

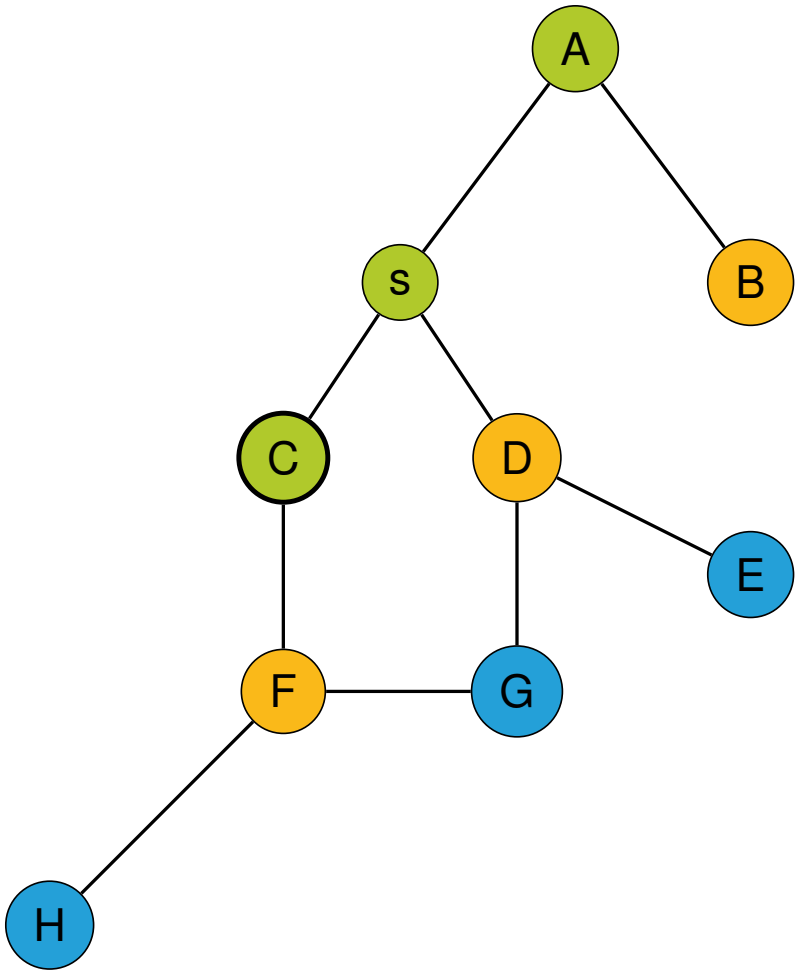
D
B



Exempel på BFS och DFS

BFS

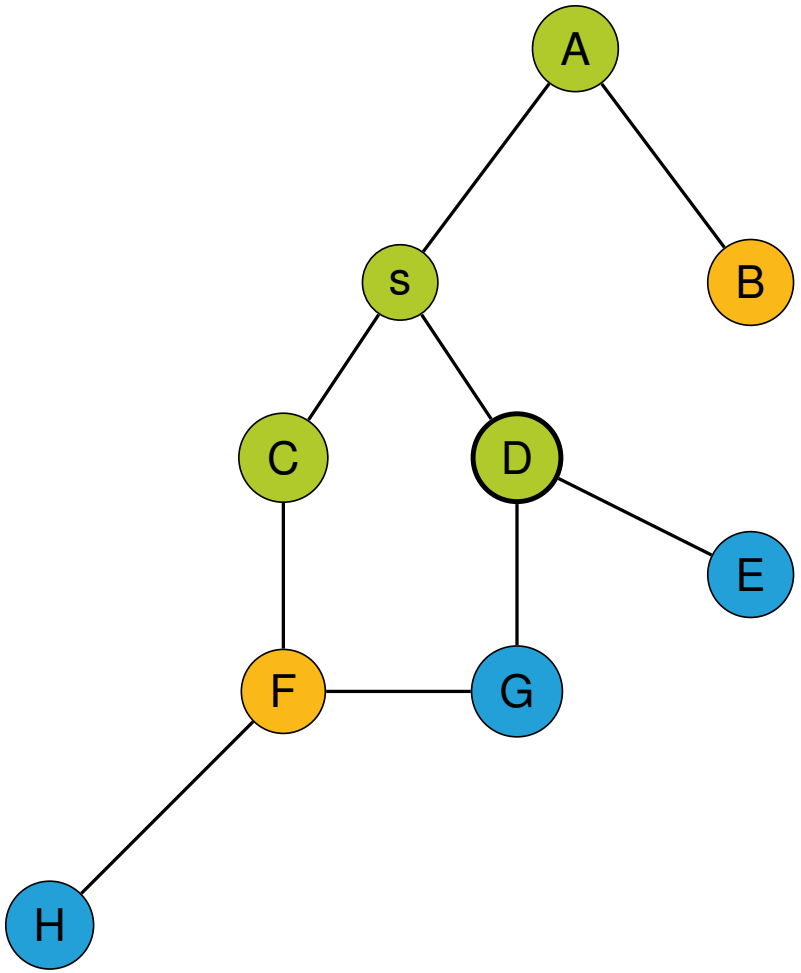
D
B
F



Exempel på BFS och DFS

BFS

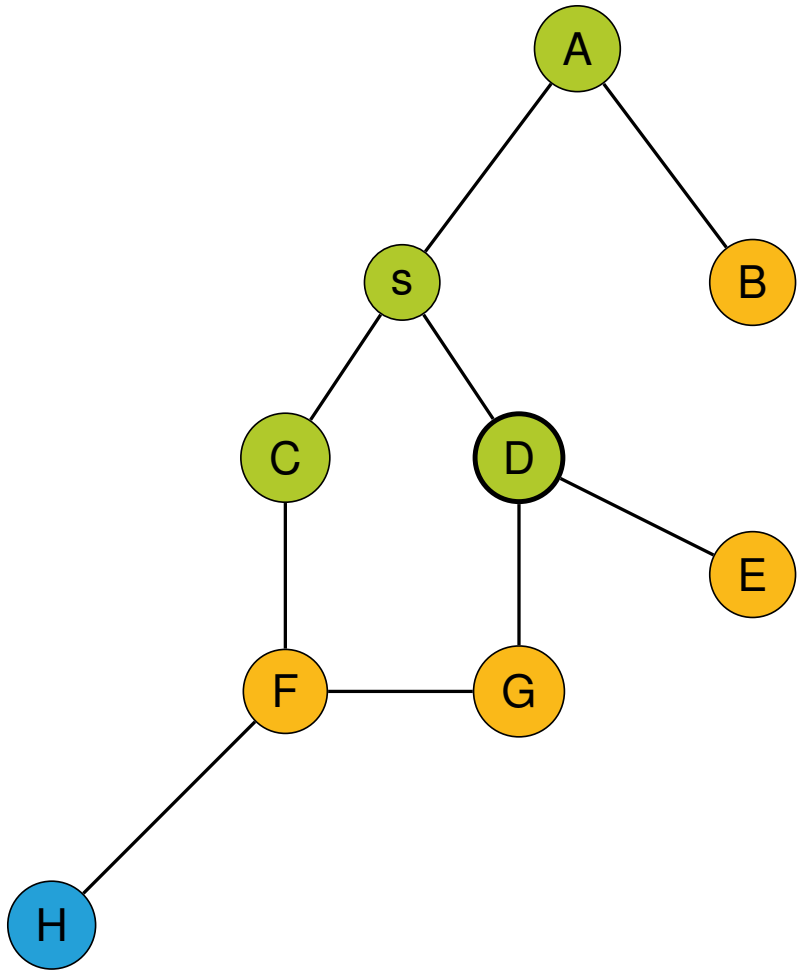
B
F



Exempel på BFS och DFS

BFS

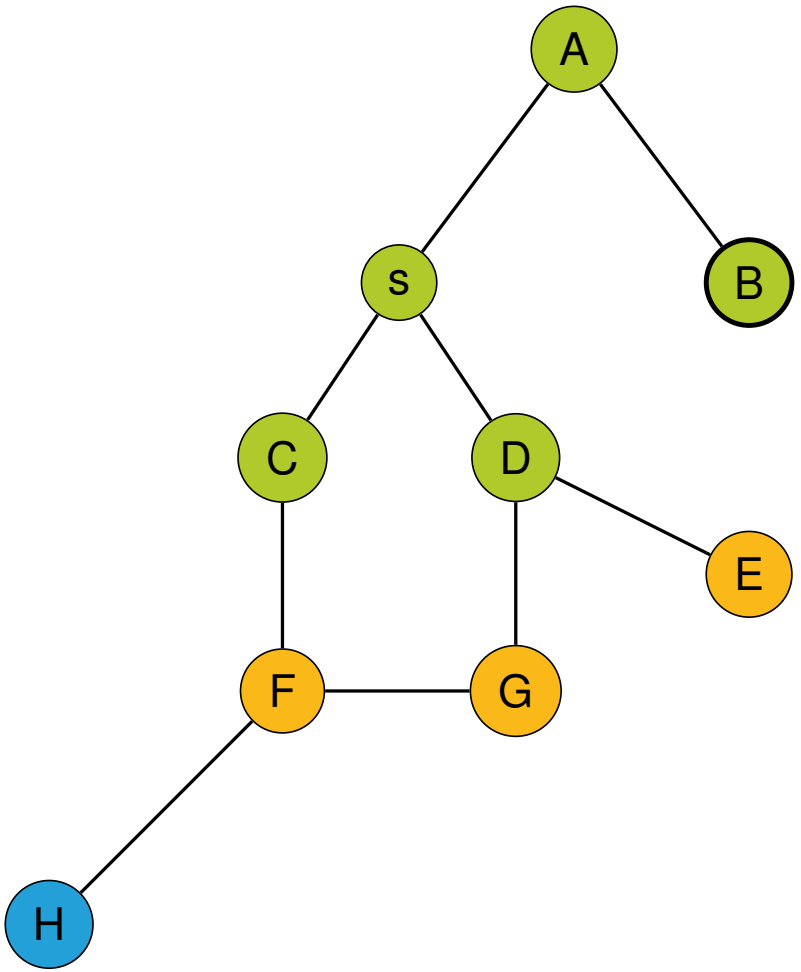
B
F
E
G



Exempel på BFS och DFS

BFS

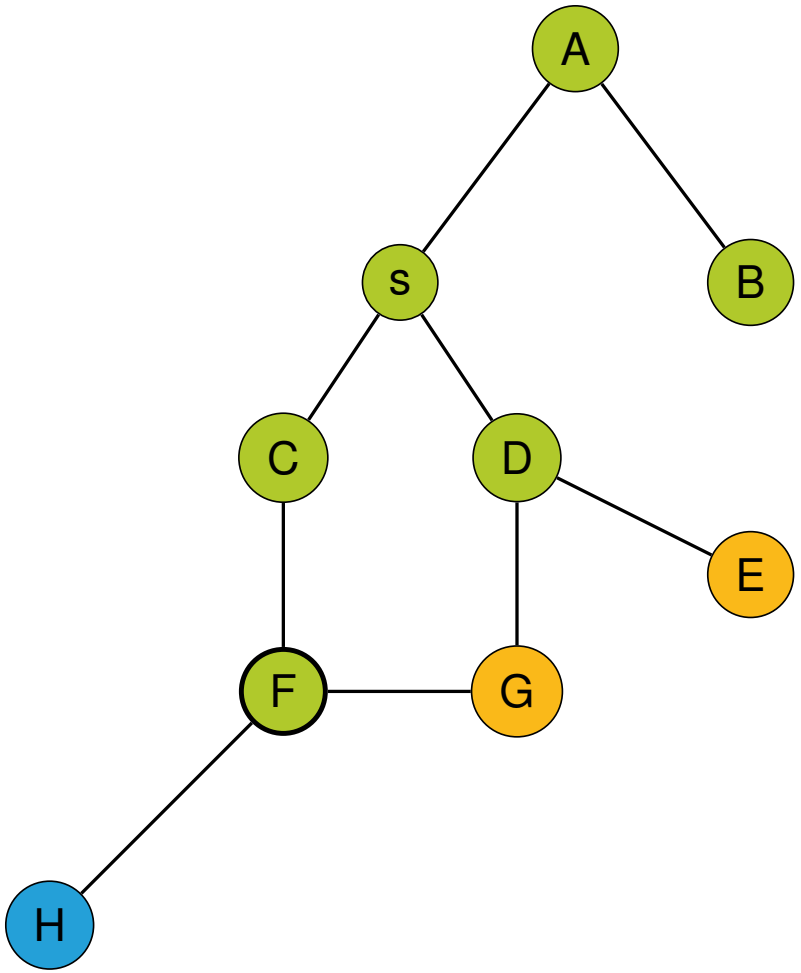
F
E
G



Exempel på BFS och DFS

BFS

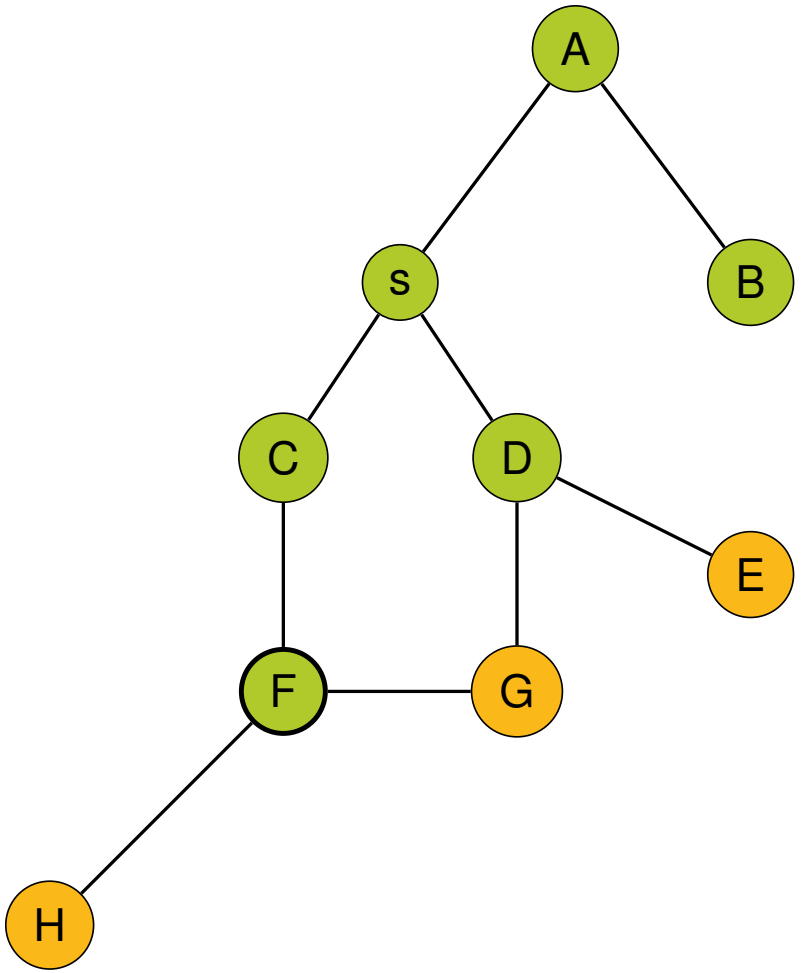
E
G



Exempel på BFS och DFS

BFS

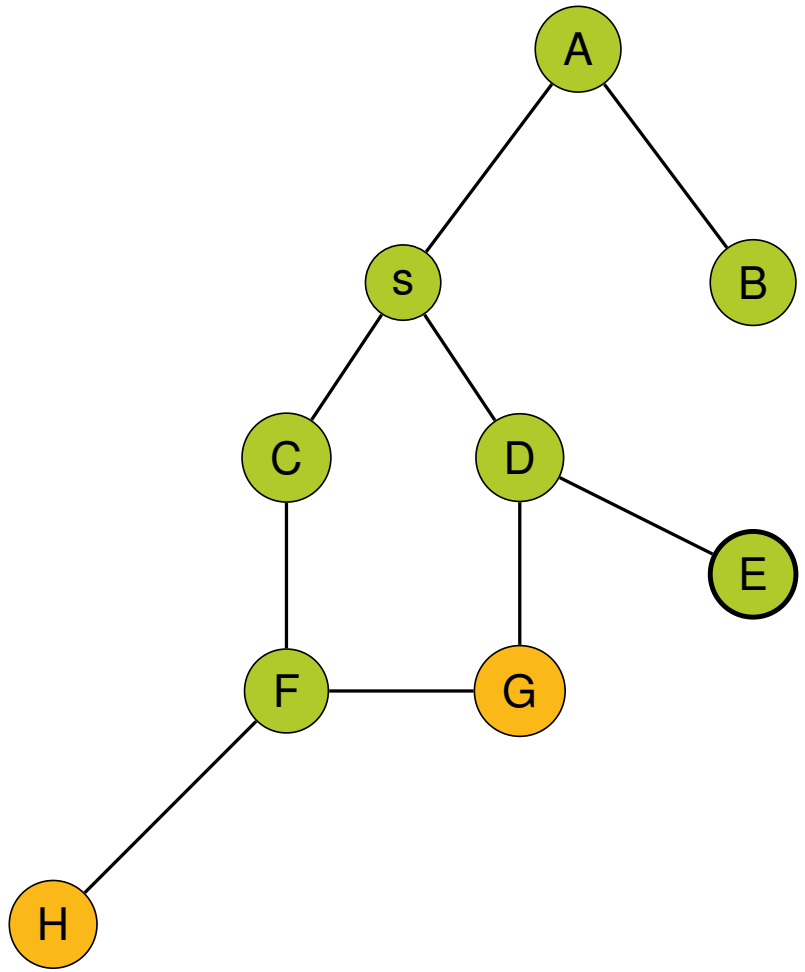
E
G
H



Exempel på BFS och DFS

BFS

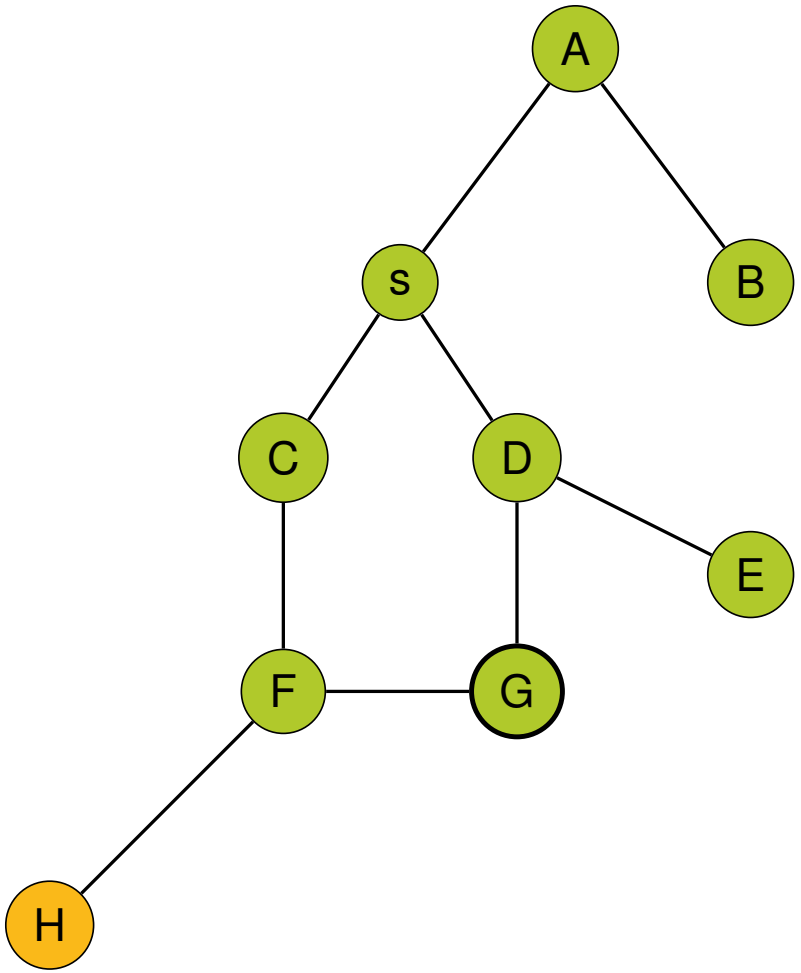
G
H



Exempel på BFS och DFS

BFS

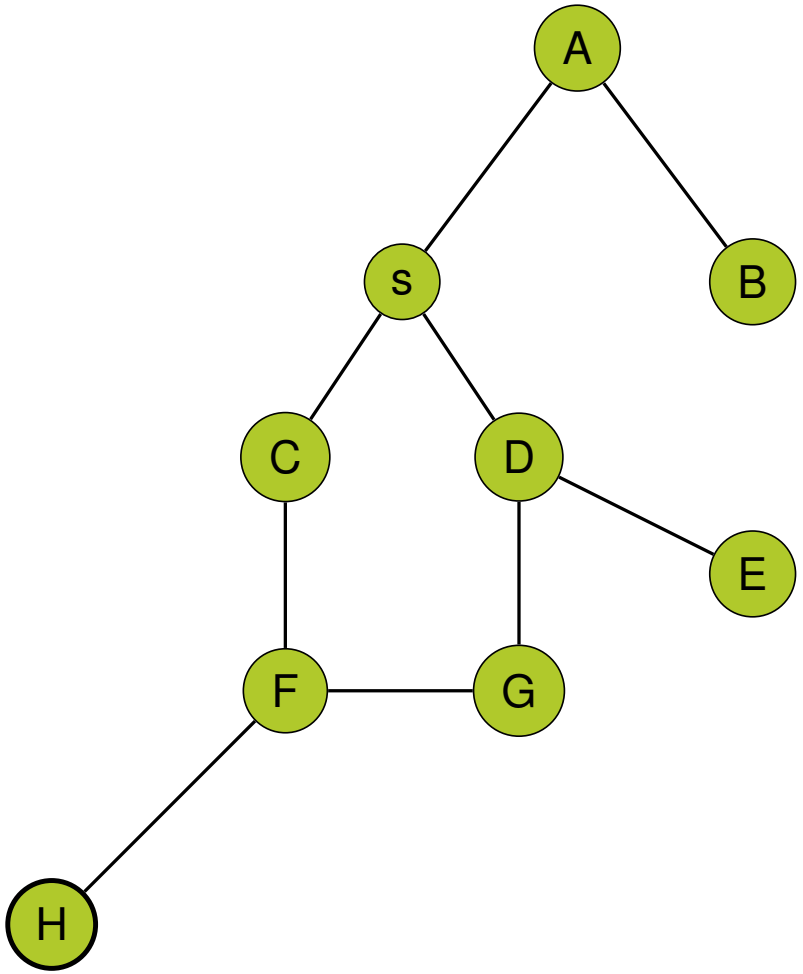
H



Exempel på BFS och DFS

BFS

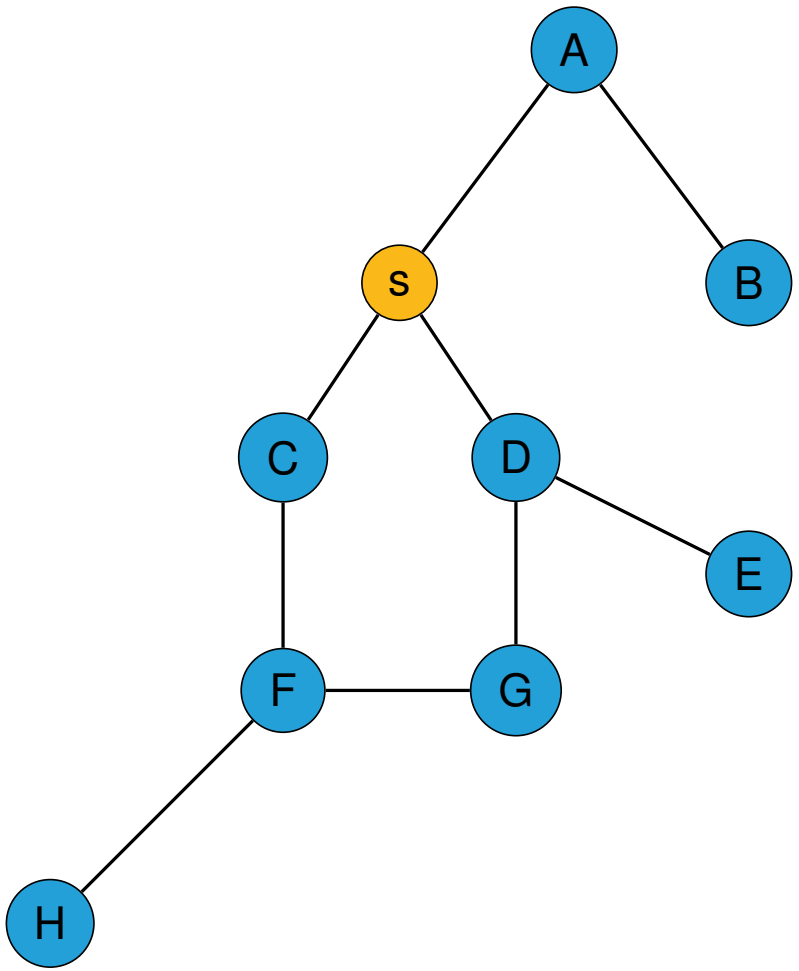
□



Exempel på BFS och DFS

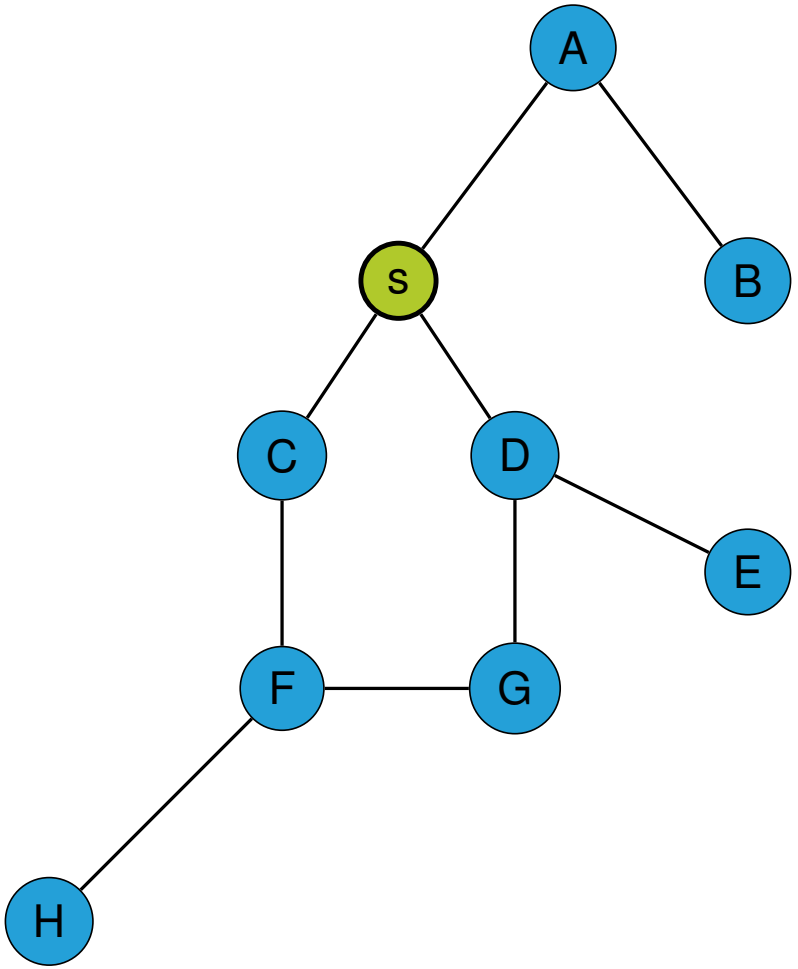
DFS

s



Exempel på BFS och DFS

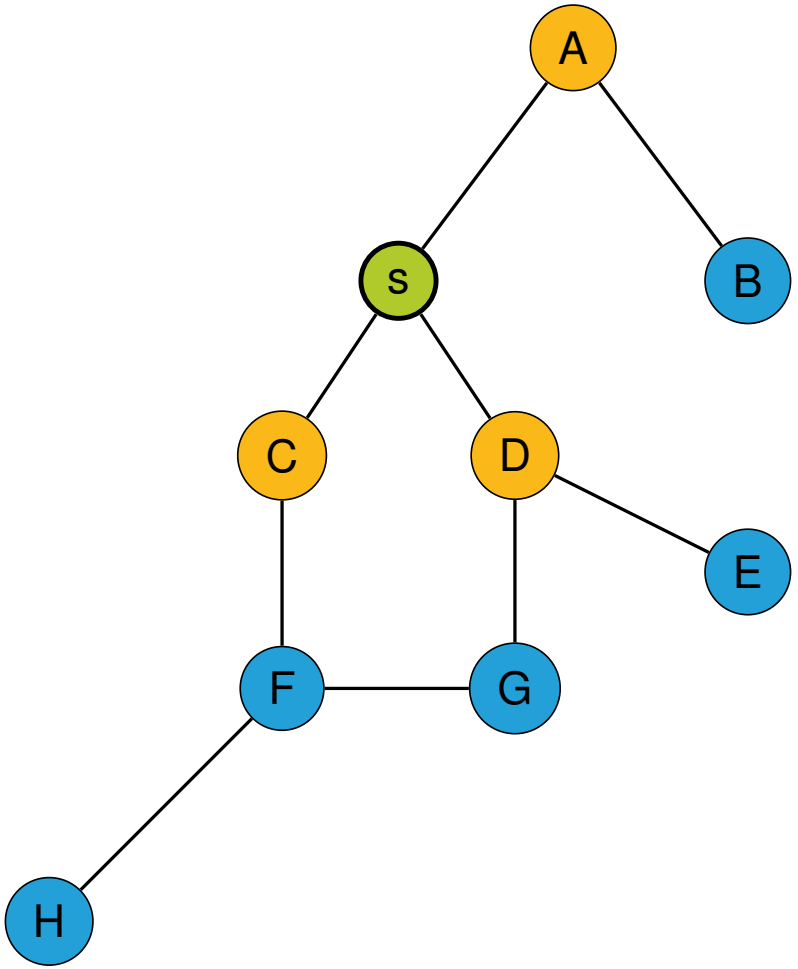
DFS



Exempel på BFS och DFS

DFS

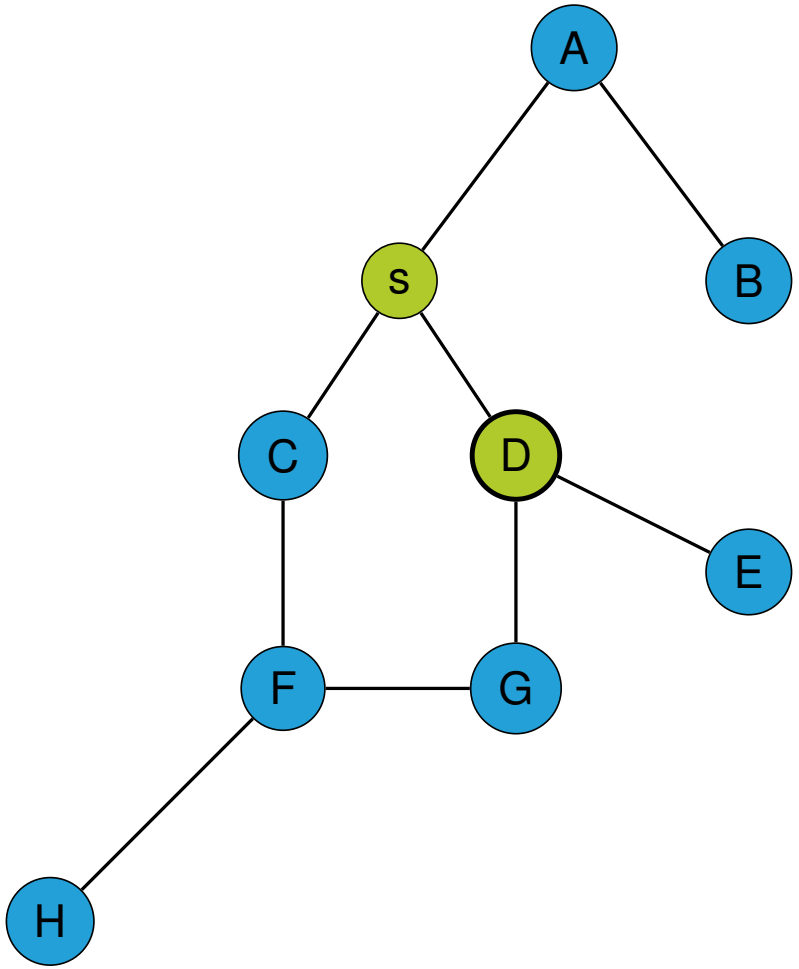
A
C
D



Exempel på BFS och DFS

DFS

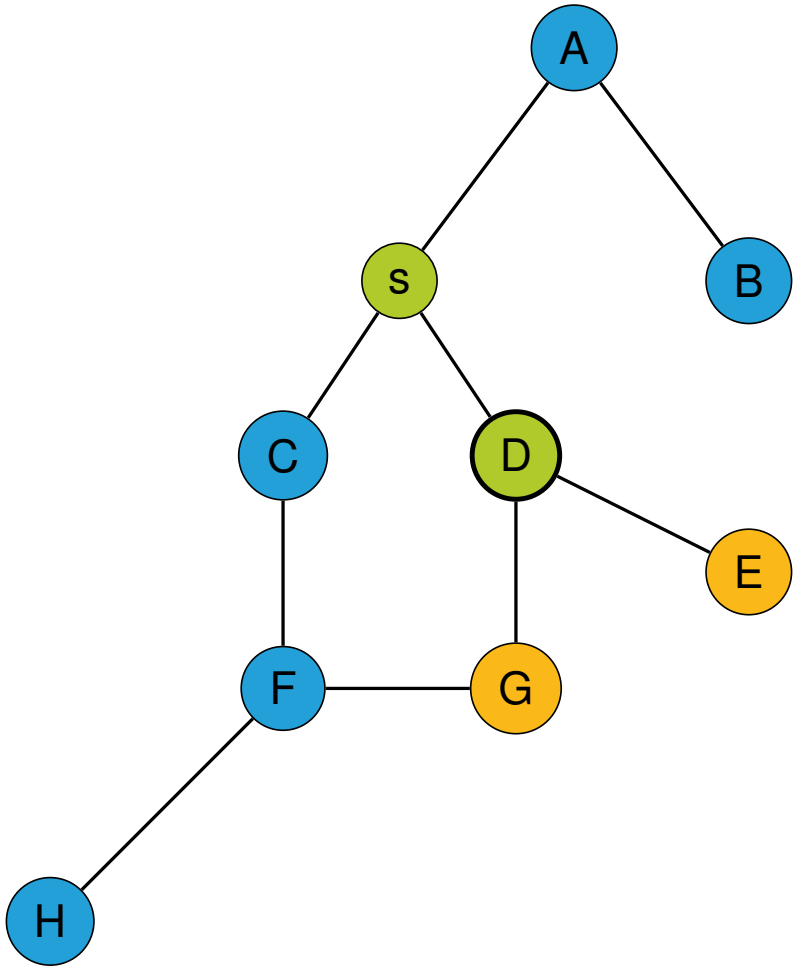
A
C



Exempel på BFS och DFS

DFS

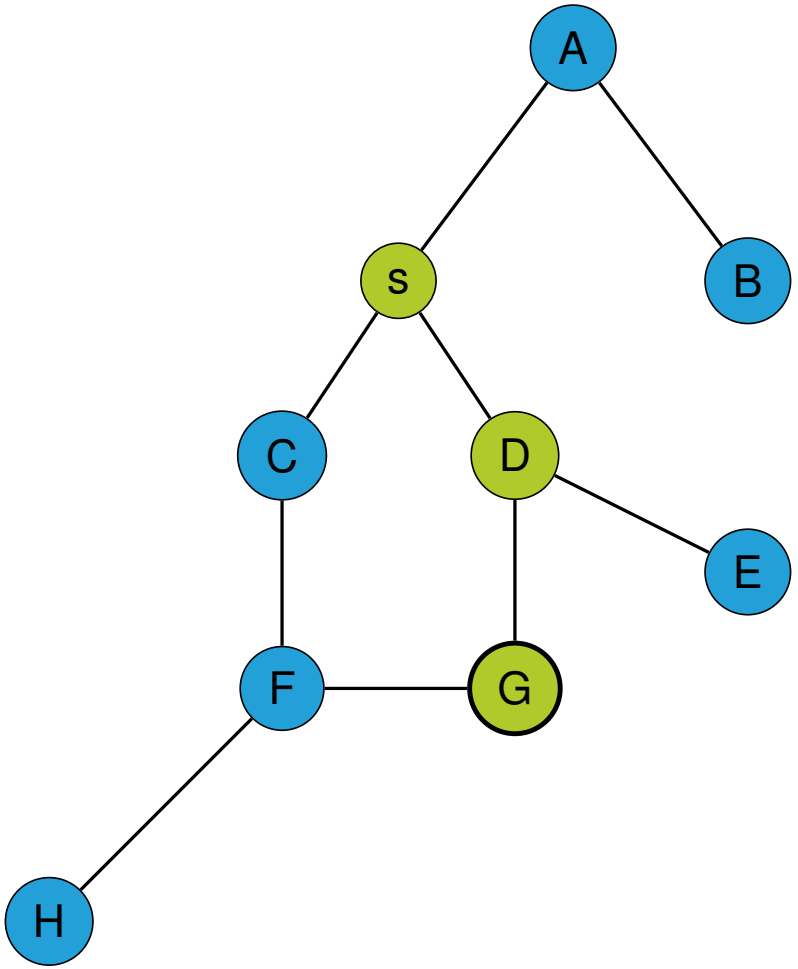
A
C
E
G



Exempel på BFS och DFS

DFS

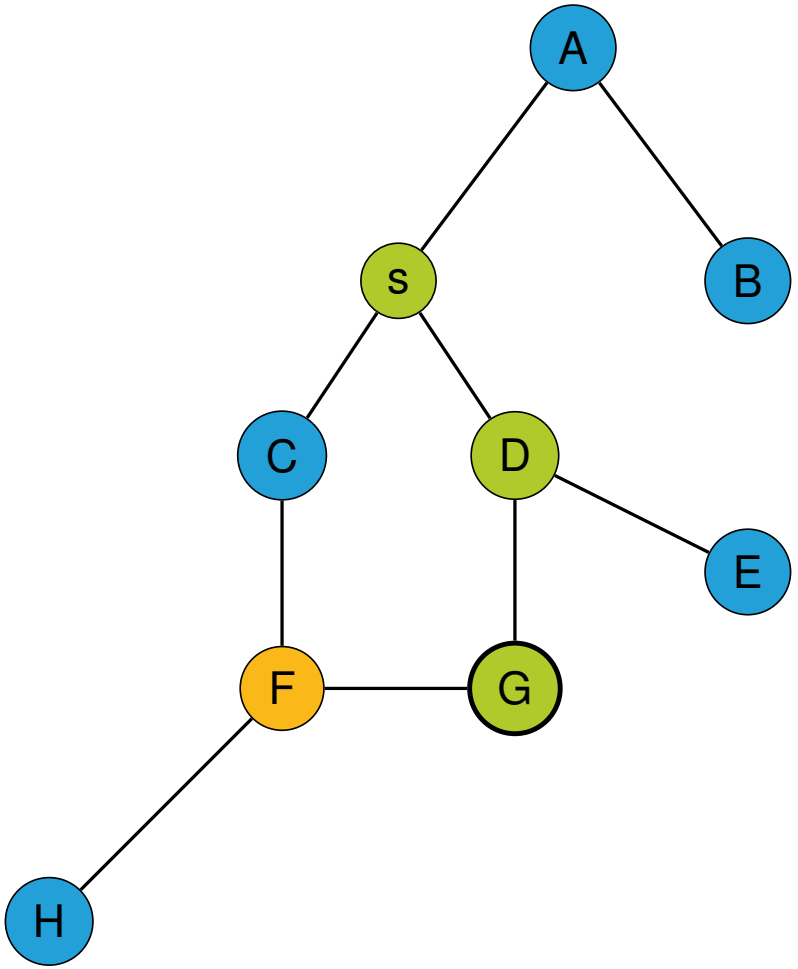
A
C
E



Exempel på BFS och DFS

DFS

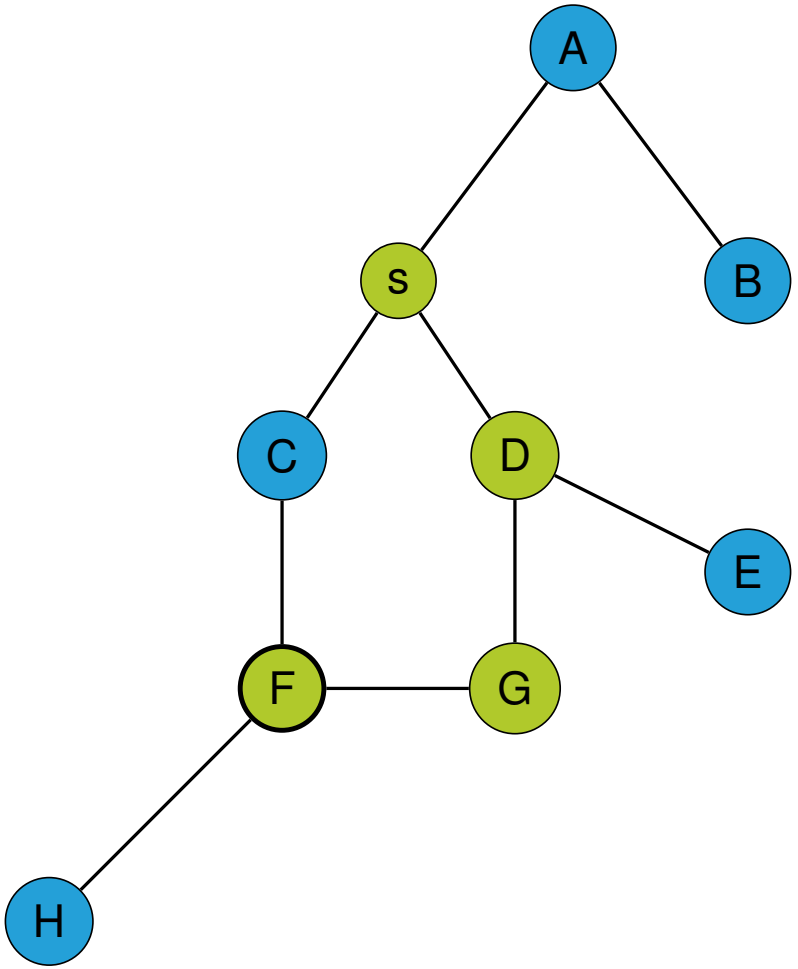
A
C
E
F



Exempel på BFS och DFS

DFS

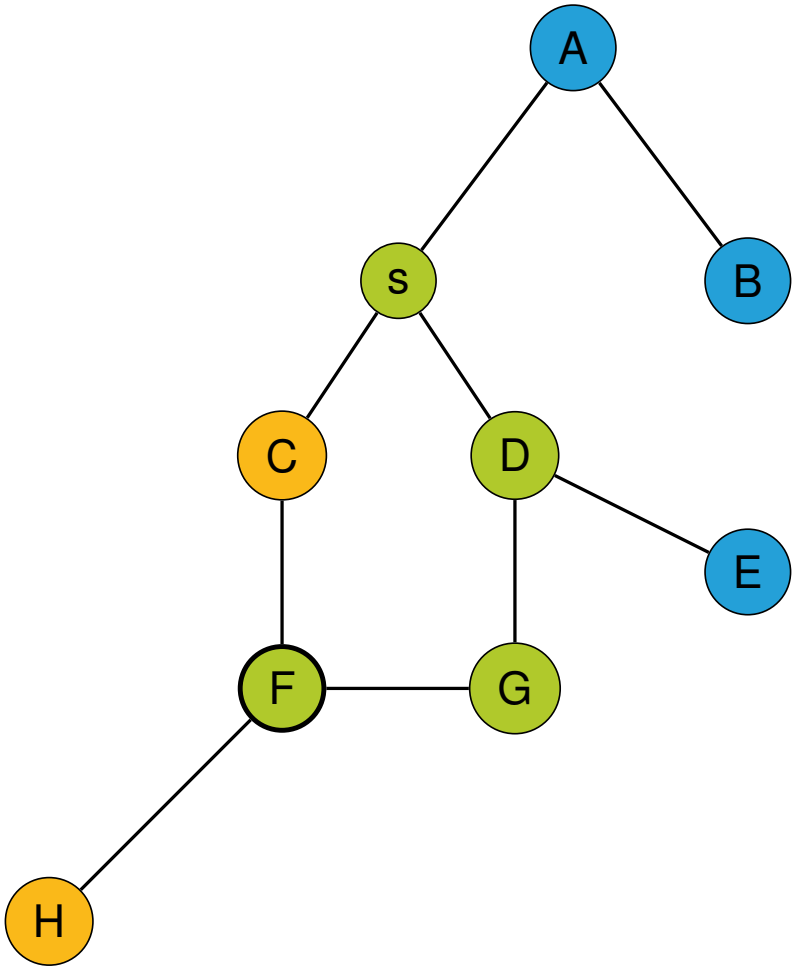
A
C
E



Exempel på BFS och DFS

DFS

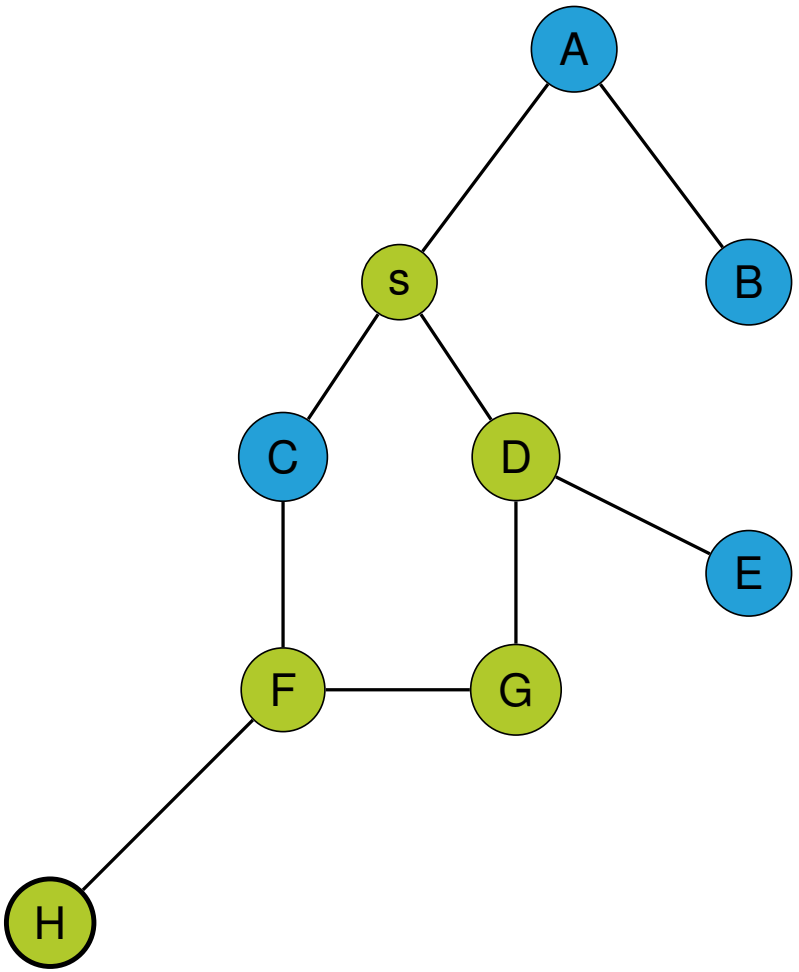
A
C
E
C
H



Exempel på BFS och DFS

DFS

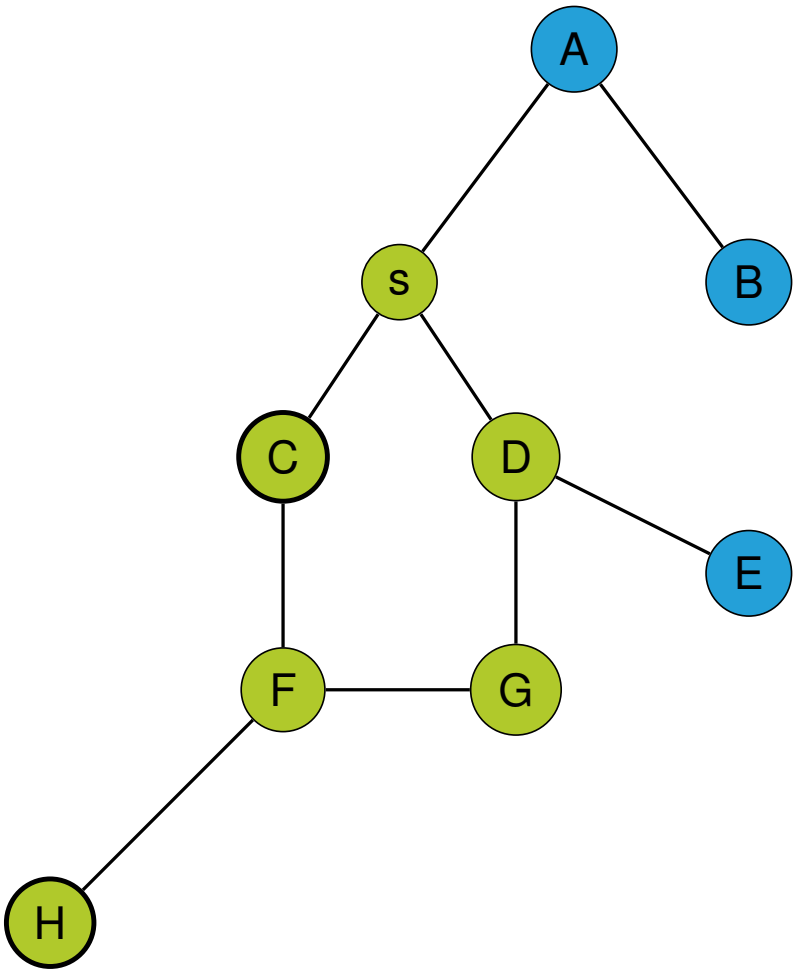
A
C
E
C



Exempel på BFS och DFS

DFS

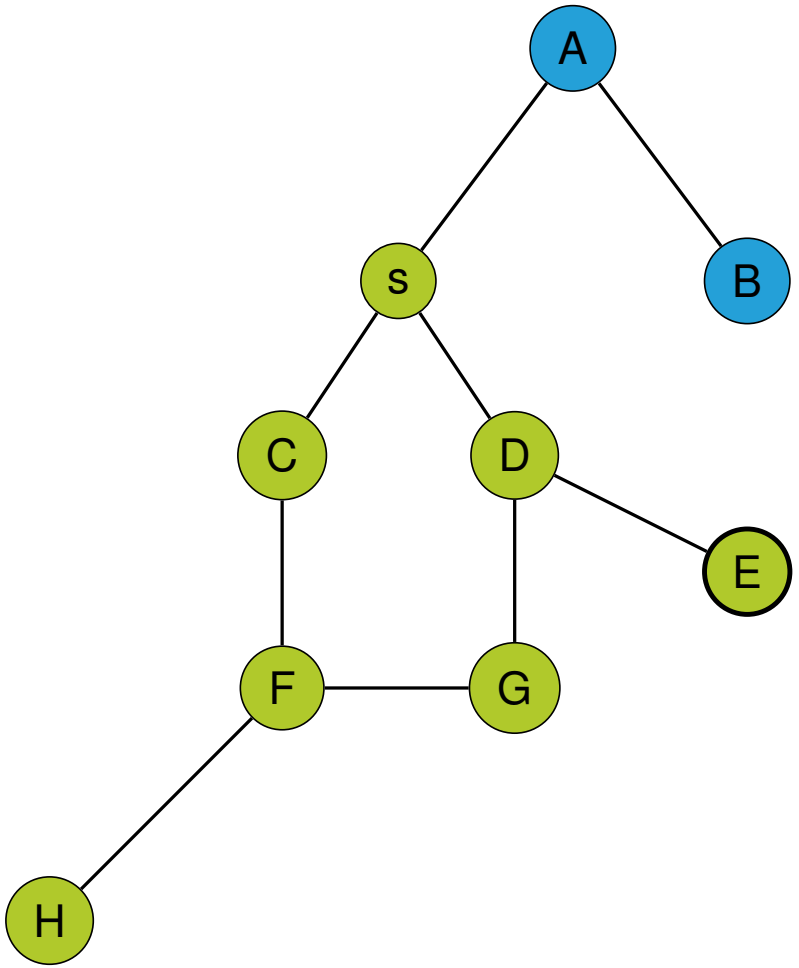
A
C
E



Exempel på BFS och DFS

DFS

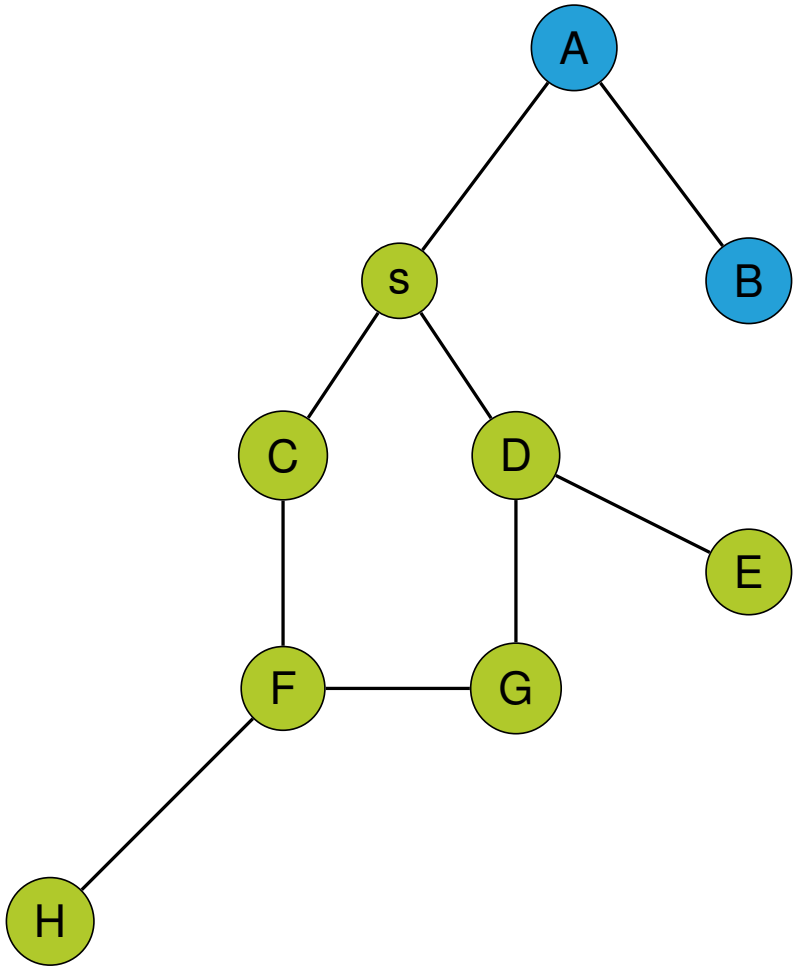
A
C



Exempel på BFS och DFS

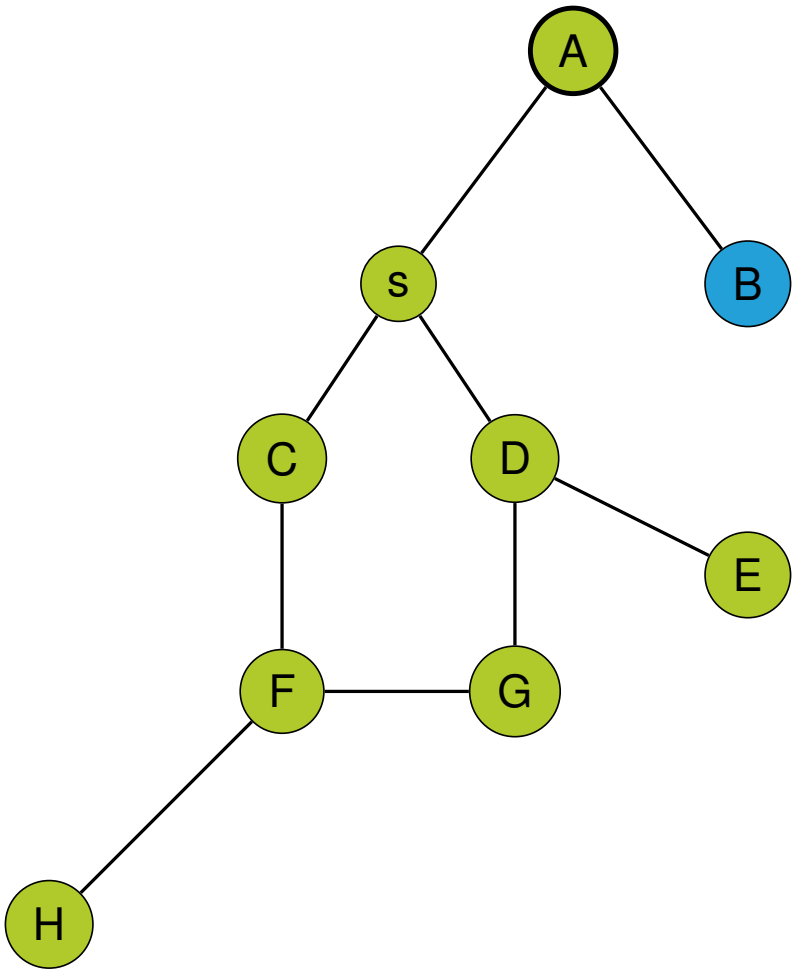
DFS

A



Exempel på BFS och DFS

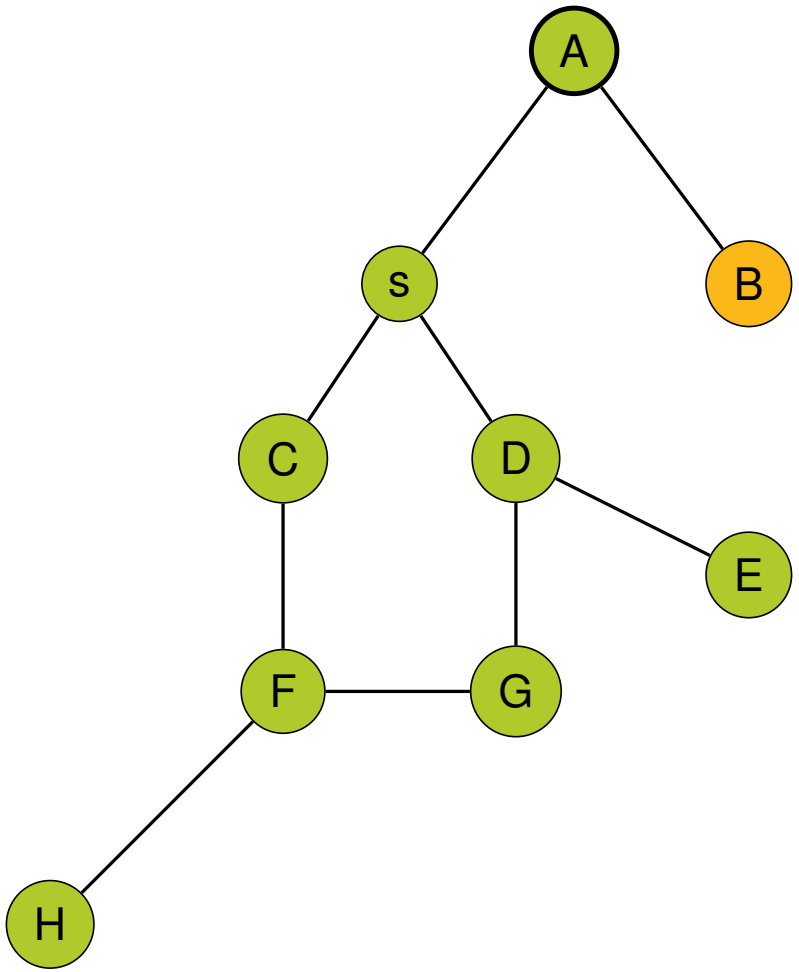
DFS



Exempel på BFS och DFS

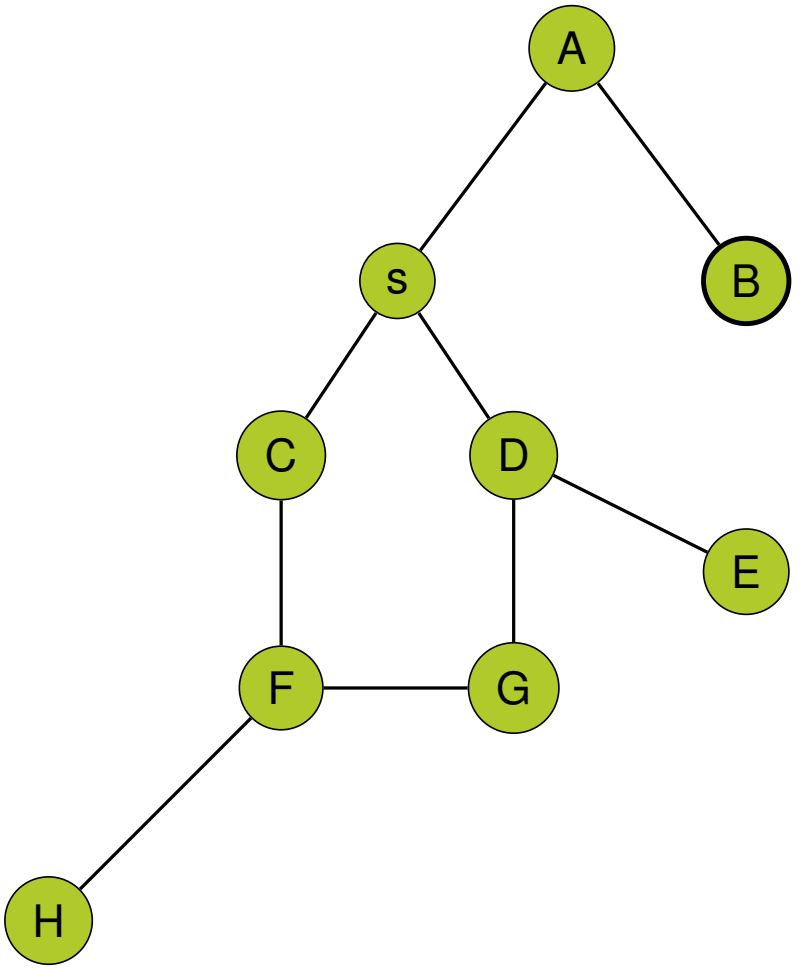
DFS

B



Exempel på BFS och DFS

DFS



Rekursion

Rekursion handlar om saker som refererar till eller beror av sig själva.

Exempel:

- “Om du inte förstår rekursion, läs den här meningen igen.”
- Rekursiva akronymer:
  - GNU = GNU’s Not Unix
  - Wine = Wine Is Not an Emulator
- Testa googla på “recursion”.

Rekursion

Matematik

- En rekursiv funktion är en funktion som definieras i termer av sig själv.
- Exempel:

$$n! = \begin{cases} n \cdot (n-1)! & n > 0 \\ 1 & n = 0 \end{cases}$$

$$F_n = \begin{cases} F_{n-1} + F_{n-2} & n > 1 \\ 1 & n = 1 \\ 0 & n = 0 \end{cases}$$

Viktigt!

Se till att ha heltäckande basfall, och att varje funktionsevaluering kan redueras till dessa.

Rekursion

Datalogi

- En rekursiv algoritm är en algoritm som anropar sig själv.
- Exempel:
  - Mergesort
  - Euklides algoritm (gcd)
  - Beräkna fakultet eller fibonacci-tal
- Även här viktigt med heltäckande basfall.

Example

```
1: function FAKULTET(n)
2:     if n == 0 then
3:         return 1
4:     else
5:         return n · FAKULTET(n − 1)
```

Dekomposition

- Engelska: divide and conquer.
- Metod för algoritmkonstruktion, som bygger på rekursion.
- Idé:
  1. Dela upp i mindre problem.
  2. Lös delproblemen rekursivt.
  3. Kombinera resultaten.
- Glöm inte basfall!
- Exempel: mergesort.

Mästarsatsen

När man vill beräkna tidskomplexiteten för en dekompositionsalgoritm så måste man kunna hantera rekursionsformler. Mästarsatsen är ett bra verktyg för detta.

Sats

Låt $a \geq 1, b > 1, d > 0$.

$$\begin{cases} T(n) = aT\left(\frac{n}{b}\right) + f(n) \\ T(1) = d \end{cases}$$

har den asymptotiska lösningen

- $T(n) \in \Theta\left(n^{\log_b a}\right)$ om $f(n) \in \mathcal{O}\left(n^{\log_b a - \epsilon}\right),\ \epsilon > 0$
- $T(n) \in \Theta\left(n^{\log_b a} \log n\right)$ om $f(n) \in \Theta\left(n^{\log_b a}\right)$
- $T(n) \in \Theta(f(n))$ om $f(n) \in \Omega\left(n^{\log_b a + \epsilon}\right), \epsilon > 0$

Uppgift 1: Max och min med dekomposition

- I vektorn $v[1..n]$ ligger n tal.
- Konstruera en dekompositionsalgoritm som hittar max och min i v.
- Algoritmen får använda högst $\lceil 3n/2 \rceil - 2$ jämförelser.
- Antalet tal i v behöver inte vara en tvåpotens.

Uppgift 1: Max och min med dekomposition

1. Dela upp i mindre problem.
   - Dela listan i två lika stora delar.
2. Lös delproblemen rekursivt.
   - Anropa algoritmen på varje halva.
3. Kombinera resultaten.
   - Vi har nu max och min från de bägge halvorna.
   - Det totala max-värdet är det största av de båda max-värdena, och analogt för min.

Uppgift 1: Max och min med dekomposition

Pseudokod

```
MinMax(v,i,j)=
    if i=j then return (v[i],v[i])
    else if i+1=j then
        if v[i]<v[j] then return (v[i],v[j])
        else return (v[j],v[i])
     else
        m:=Floor((j-i)/2)
        if Odd(m) then m:=m+1;
        (min1,max1):=MinMax(v,i,i+m-1);
        (min2,max2):=MinMax(v,i+m,j);
        min:=(if min1<min2 then min1 else min2);
        max:=(if max1>max2 then max1 else max2);
        return (min,max);
```

Uppgift 1: Max och min med dekomposition

Tidskomplexitet

Sats

$$\begin{cases} T(n) = aT\left(\frac{n}{b}\right) + f(n) \\ T(1) = d \end{cases}$$

- $T(n) \in \Theta\left(n^{\log_b a}\right)$ om $f(n) \in \mathcal{O}\left(n^{\log_b a - \epsilon}\right),\ \epsilon > 0$
- $T(n) \in \Theta\left(n^{\log_b a} \log n\right)$ om $f(n) \in \Theta\left(n^{\log_b a}\right)$
- $T(n) \in \Theta(f(n))$ om $f(n) \in \Omega\left(n^{\log_b a + \epsilon}\right),\ \epsilon > 0$

$$\begin{cases} T(n) = 2 \cdot T(\frac{n}{2}) + \mathcal{O}(1) \\ T(1) = \mathcal{O}(1) \end{cases} \implies T(n) \in \mathcal{O}(n)$$

Uppgift 2: Matrismultiplikation

- Strassens algoritm
  - Multiplicerar två $n \times n$-matriser (beräknar $C = AB$) på tiden $\mathcal{O}(n^{2.808})$.
  - Dela in A och B i $2 \cdot 2 = 4$ blockmatriser av storlek $\frac{n}{2} \times \frac{n}{2}$.
  - Konstruera C genom att kombinera produkter av dessa mindre matriser.
  - Går snabbare än $\mathcal{O}(n^3)$ för att den lyckas konstruera C med endast med 7 multiplikationer av $\frac{n}{2} \times \frac{n}{2}$-matriser istället för 8.
- Idé: kan vi lyckas bättre genom att dela upp matriserna i $3 \cdot 3 = 9$ mindre blockmatriser av storlek $\frac{n}{3} \times \frac{n}{3}$ istället?
  - Viggo har försökt hitta det minimala antalet multiplikationer av $\frac{n}{3} \times \frac{n}{3}$-matriser som krävs för att multiplicera en $n \times n$-matris.
  - Han lyckades nästan komma fram till 22 multiplikationer.
  - Om han hade lyckats, vilken tidskomplexitet hade det gett för multiplikation av två $n \times n$-matriser?

Uppgift 2: Matrismultiplikation

- Låt $T(n)$ vara tiden det tar att multiplicera två $n \times n$-matriser.
- Algoritmen gör 22 rekursiva anrop som varje tar tiden $T(n/3)$.
- Att kombinera dessa resultat går på linjär tid i antalet element, dvs $\mathcal{O}(n^2)$.

$$\begin{cases} T(n) = 22 \cdot T(n/3) + \mathcal{O}(n^2) \\ T(1) = \mathcal{O}(1) \end{cases}$$

Uppgift 2: Matrismultiplikation

Sats

$$\begin{cases} T(n) = aT\left(\frac{n}{b}\right) + f(n) \\ T(1) = d \end{cases}$$

- $T(n) \in \Theta\left(n^{\log_b a}\right)$ om $f(n) \in \mathcal{O}\left(n^{\log_b a - \epsilon}\right),\ \epsilon > 0$
- $T(n) \in \Theta\left(n^{\log_b a} \log n\right)$ om $f(n) \in \Theta\left(n^{\log_b a}\right)$
- $T(n) \in \Theta(f(n))$ om $f(n) \in \Omega\left(n^{\log_b a + \epsilon}\right),\ \epsilon > 0$

$$\begin{cases} T(n) = 22 \cdot T\left(\frac{n}{3}\right) + \mathcal{O}(n^2) \\ T(1) = \mathcal{O}(1) \end{cases} \implies T(n) \in \mathcal{O}(n^{\log_3 22}) = \mathcal{O}(n^{2.814})$$

Uppgift 3: Majoritet med dekomposition

- Input: array A med n element.
- Konstruera en algoritm som tar reda på om något element i arrayen A är i majoritet, och i så fall returnerar det.
  - Ett element är i majoritet om det förekommer fler än $n/2$ gånger (dvs minst $\lfloor \frac{n}{2} \rfloor + 1$ gånger).
- Använd dekomposition. Tidskomplexiteten ska vara $\mathcal{O}(n \log n)$.
- Den enda tillåtna jämförelseoperationen är $=$.
- *Det finns alltså ingen ordningsrelation mellan elementen.*

Uppgift 3: Majoritet med dekomposition

- Insikt: Om det existerar ett majoritetselement så måste det vara i majoritet åtminstone i ena halvan av arrayen.
  - Varför?
- Rekursiv tanke:
  - Kolla majoritet rekursivt i vänstra och högra halvan.
  - Från de rekursiva anropen får vi två möjliga kandidater till majoritetselement i hela arrayen.
  - Gå igenom arrayen och räkna antalet förekomster av denna.
  - Om något element är i total majoritet, returnera detta.

Uppgift 3: Majoritet med dekomposition

Pseudokod

Majority($A[1..n]$) =
    if $n = 1$ **then return** $A[1]$
    mid$\leftarrow \lceil (n+1)/2 \rceil$
    majInLeft$\leftarrow$ Majority($A[1..\text{mid}-1]$)
    majInRight$\leftarrow$ Majority($A[\text{mid}..n]$)
    if majInleft=majInRight **then return** majInLeft
    noOfMajInLeft$\leftarrow 0$
    noOfMajInRight$\leftarrow 0$
    for $i \leftarrow 1$ **to** n **do**
        if $A[i]$ =majInLeft **then** noOfMajInLeft$\leftarrow$noOfMajInLeft$+1$
        else if $A[i]$ =majInRight **then** noOfMajInRight$\leftarrow$noOfMajInRight$+1$
    if noOfMajInLeft$\geq mid$ **then return** majInLeft
    if noOfMajInRight$\geq mid$ **then return** majInRight
    else return NULL

Uppgift 3: Majoritet med dekomposition

Tidskomplexitet

Sats

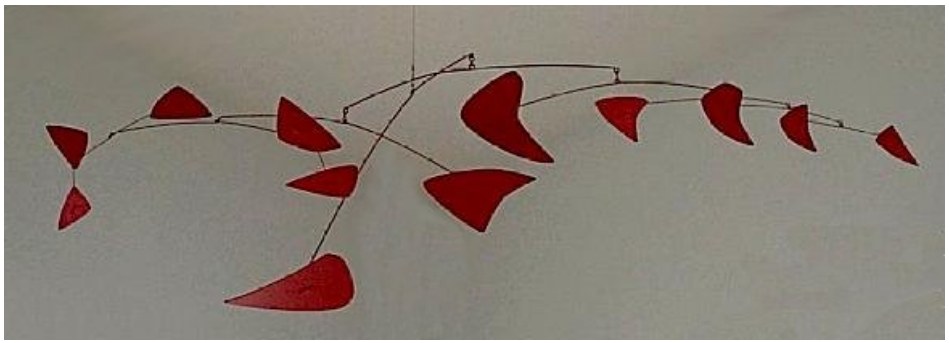
$$\begin{cases} T(n) = aT\left(\frac{n}{b}\right) + f(n) \\ T(1) = d \end{cases}$$

- $T(n) \in \Theta\left(n^{\log_b a}\right)$ om $f(n) \in \mathcal{O}\left(n^{\log_b a - \epsilon}\right),\ \epsilon > 0$
- $T(n) \in \Theta\left(n^{\log_b a} \log n\right)$ om $f(n) \in \Theta\left(n^{\log_b a}\right)$
- $T(n) \in \Theta(f(n))$ om $f(n) \in \Omega\left(n^{\log_b a + \epsilon}\right),\ \epsilon > 0$

$$\begin{cases} T(n) = 2 \cdot T(\frac{n}{2}) + \mathcal{O}(n) \\ T(1) = \mathcal{O}(1) \end{cases} \implies T(n) \in \mathcal{O}(n \log n)$$

Uppgift 4: Mobil

- Konstruera en algoritm som balanserar en mobil!



Red Mobile, di en:Alexander Calder. 1956. Lamiera verniciata e bacchette di metallo. Montreal Museum of Fine Arts. Foto di Montrealais.

- Mobilen beskrivs som ett binärträd:
  - Löven motsvarar kulor som hänger underst.
  - Inre noder motsvarar horisontella pinnar, i vars ändar trådar är fastknutna, där barnen hänger.
- Indata: binärträd med vikter i noderna
  - Vikten i varje löv: kulans massa.
  - Vikten i varje inre nod: pinnens längd.
- Utdata: samma binärträd, men i varje inre nod ska stå hur långt från pinnens vänstra kant den ska fästas i föräldern.

Uppgift 4: Mobil

- Vi vill att mobilen ska vara i jämvikt. Dags att komma ihåg lite gymnasiefysik!
- Betrakta en pinne av längd l i mobilen.
- Säg att vikten v hänger i vänstra änden och vikten w i den högra.
- Låt x vara avståndet från pinnens vänstra ände till tråden som fäster pinnen i sin förälder.
- Momentjämvikt kräver:

$$xv = (l - x)w \iff xv + xw = lw \iff x = \frac{lw}{v + w}$$

- Låt den rekursiva funktionen returnera den totala vikten av subträdet den anropas på.

Uppgift 4: Mobil

Pseudokod

Balance(p)=
    if $p.left =$**NIL then return** $p.num$
    $left \leftarrow$ *Balance*$(p.left)$
    $right \leftarrow$ *Balance*$(p.right)$
    $p.x \leftarrow p.num \cdot right/(left + right)$
    return $left + right$

- Tidskomplexitet: $\mathcal{O}(n)$

Dynamisk programmering

Betrakta följande algoritm för att beräkna Fibonacci-tal:

function FIB(n)
    if $n = 0$ **then**
        return 0
    else if $n = 1$ **then**
        return 1
    else if **then**
        FIB($n - 2$) + FIB($n - 1$)

- Exponentiell tidskomplexitet!
- Vi anropar funktionen flera gånger med samma argument. Onödigt!
- Istället kan vi spara undan delresultaten och bara beräkna varje resultat en gång.

Dynamisk programmering

- Recept för dynamisk programmering:
  1. Hitta en matematisk rekursionsformel som beskriver lösningen på problemet.
  2. Beskriv basfallen.
  3. Hitta en lämplig beräkningsordning.
- När detta är klart är man redo att implementera lösningen.

Uppgift 5: Talföljder

Anta att du har fått rekursionen för en talföljd presenterad för dig. Skriv pseudokoden för en dynamisk programmeringsalgoritm för att beräkna S_n om

a)

$$S_n = \begin{cases} 1 & \text{om } n = 0, \\ 2(S_{n-1}) + 1 & \text{annars.} \end{cases}$$

Skriv ned de 7 första talen. Vad behöver man spara under beräkningens gång?

Uppgift 5: Talföljder

Pseudokod för a)

$$S_n = \begin{cases} 1 & \text{om } n = 0, \\ 2(S_{n-1}) + 1 & \text{annars.} \end{cases}$$

```
Talfoljda(n) =
    v = 1
    for i = 1 to n do
      v = 2*v+1
    return v
```

Uppgift 5: Talföljder

Anta att du har fått rekursionen för en talföljd presenterad för dig. Skriv pseudokoden för en dynamisk programmeringsalgoritm för att beräkna S_n om

b)

$$S_n = \begin{cases} 4 & \text{om } n = 1, \\ 5 & \text{om } n = 2, \\ \max\left(S_{n-1}, S_{n-2}, S_{n-1} - S_{n-2} + 7\right) & \text{annars.} \end{cases}$$

Skriv ned de 7 första talen. Vad behöver man spara under beräkningens gång?

Uppgift 5: Talföljder

Pseudokod för b)

$$S_n = \begin{cases} 4 & \text{om } n = 1, \\ 5 & \text{om } n = 2, \\ \max\left(S_{n-1}, S_{n-2}, S_{n-1} - S_{n-2} + 7\right) & \text{annars.} \end{cases}$$

```
Talfoljdb(n) =
  if n = 1 then return 4
  elif n = 2 then return 5
  else
    v0 = 4
    v1 = 5
    for i = 2 to n do
      v = max(v1, v0, v1-v0+7)
      v0 = v1
      v1 = v
    return v
```

Uppgift 5: Talföljder

Anta att du har fått rekursionen för en talföljd presenterad för dig. Skriv pseudokoden för en dynamisk programmeringsalgoritm för att beräkna S_n om

c)

$$S_n = \begin{cases} 4 & \text{om } n = 1, \\ 5 & \text{om } n = 2, \\ \max\left(S_{n-2}, S_{n-1} - S_{n-2} + 7\right) & \text{annars.} \end{cases}$$

Skriv ned de 7 första talen. Vad behöver man spara under beräkningens gång?

Uppgift 5: Talföljder

Pseudokod för c)

$$S_n = \begin{cases} 4 & \text{om } n = 1, \\ 5 & \text{om } n = 2, \\ \max\left(S_{n-2}, S_{n-1} - S_{n-2} + 7\right) & \text{annars.} \end{cases}$$

```
Talfoljdc(n) =
  if n = 1 then return 4
  elif n = 2 then return 5
  else
    v0 = 4
    v1 = 5
    for i = 2 to n do
      v = max(v0, v1-v0+7)
      v0 = v1
      v1 = v
    return v
```

Uppgift 6: Generaliserade talföljder

a) Tvådimensionell rekursion utan indata

Givet följande rekursion, konstruera en algoritm som beräknar $M[i,j]$ med dynamisk programmering. Argumentera för att algoritmens beräkningsordning fungerar.

$$M[i,j] = \begin{cases} 0 & \text{om } i = 0 \text{ eller } j = 0, \\ M[i-1,j-1] + i & \text{annars.} \end{cases}$$

Uppgift 6: Generaliserade talföljder

$$M[i,j] = \begin{cases} 0 & \text{om } i = 0 \text{ eller } j = 0, \\ M[i-1,j-1] + i & \text{annars.} \end{cases}$$

- Vi vill beräkna hela M, upp till $j = n$, $i = m$.
- Beräkningsordning?
  - Radvis från vänster till höger fungerar fint.
  - Hela översta raden och vänstra kolumnen är redan ifylld (basfallet).
  - När vi kommer till en position kommer positionen ett steg snett uppåt redan vara ifylld.

for $j \leftarrow 0$ **to** n
    $M[0,j] \leftarrow 0$
for $i \leftarrow 1$ **to** m
    $M[i,0] \leftarrow 0$
    for $j \leftarrow 1$ **to** n
        $M[i,j] \leftarrow M[i-1,j-1] + i$
return M

- Tidskomplexitet? $\Theta(nm)$

Uppgift 6: Generaliserade talföljder

b) 2D-rekursion med indata

- Låt a och b vara två strängar av längd n resp. m.
- Låt a_i vara i:te tecknet i a och b_j det j:te tecknet i b.
- Hitta en beräkningsordning och skriv en algoritm som utför följande rekursion med hjälp av dynamisk programmering.

$$M[i,j] = \begin{cases} 0 & \text{om } i = 0 \text{ eller } j = 0, \\ M[i-1, j-1] + 1 & \text{om } a_i = b_j, \\ 0 & \text{annars.} \end{cases}$$

Uppgift 6: Generaliserade talföljder

- Beräkningsordning?
  - Samma som förra gången.
  - Radvis från vänster till höger fungerar fint.

for $j \leftarrow 0$ **to** n
    $M[0,j] \leftarrow 0$
for $i \leftarrow 1$ **to** m
    $M[i,0] \leftarrow 0$
    for $j \leftarrow 1$ **to** n
        if $a_i = b_j$ **then**
            $M[i,j] \leftarrow M[i-1,j-1]+1$
        else $M[i,j] \leftarrow 0$
return M

- Tidskomplexitet? $\Theta(nm)$

Nästa vecka

- Mer dynamisk programmering!
  - Från konkret problem till DP-lösning
- Labbteori 2