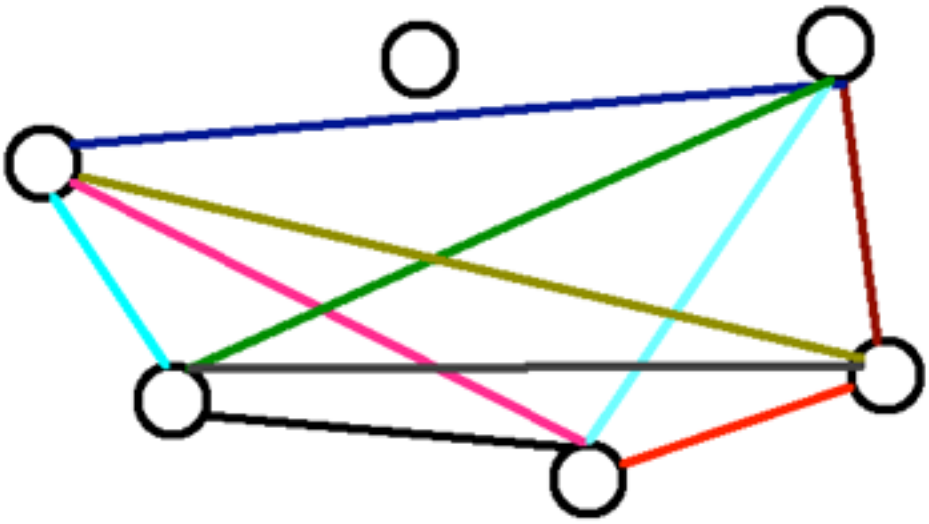
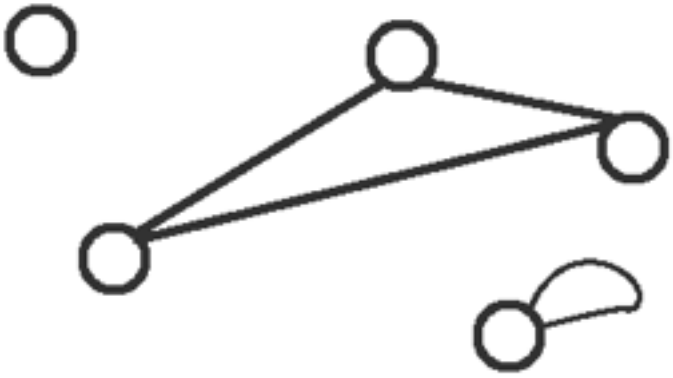


Rita en oriktad graf med 6 hörn, 10 kanter och 2 sammanhängande komponenter.

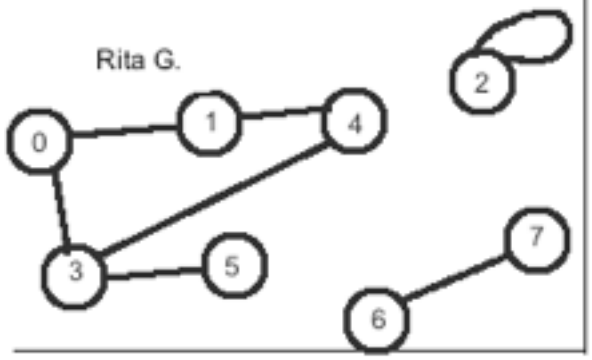


Går det att rita en graf med 5 hörn, 4 kanter och 3 komponenter?

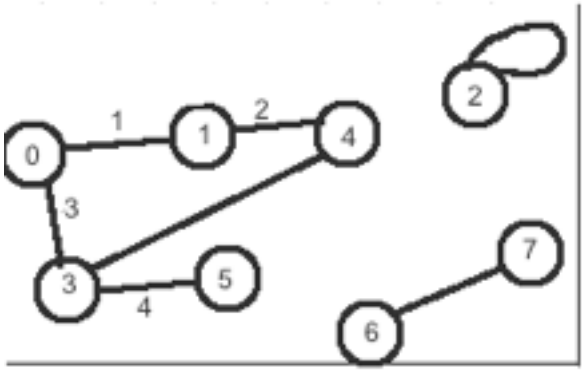
JA

Låt G vara en oriktad graf som består av 8 hörn numrerade från 0 till 7 och kantmängden {(0,1), (0,3), (1,4), (2,2), (3,4), (3,5), (6,7)}.

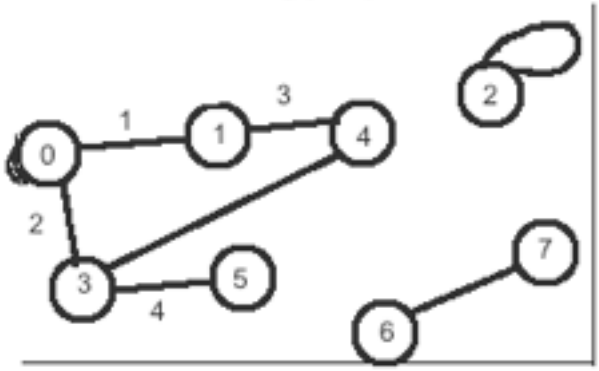
Rita G.



Ange ordningen som hörnen besöks vid en djupetförstsökning (DFS) med start i hörn 0.



Ange ordningen som hörnen besöks vid en breddenförstsökning (BFS) med start i hörn 0.



- **Skulle du representera en graf med hjälp av en närhetsmatris eller med hjälp av närhetslistor i följande fall? Motivera dina svar.**
  - **Grafen har 1000 hörn och 2000 kanter och det är viktigt att vara sparsam med minnet.**

    Jag skulle välja en närhetslista.

    Vi räknar på det:

    Matrisen skulle bli 1000 x 1000 stor. Vi säger att varje element i matrisen tar 1 bit eftersom man bara lagrar 1 eller 0. Då får vi att matrisen tar upp 1.000.000 bits. Detta är oberoende av antalet kanter.

    Listan kommer ha 1000 sub-listor.

    Nu vet jag inte detta exakt och bara gissar att “initiera”/“indexera” en sub-lista kostar 1 bit.

    Då har vi 1000x1 bits för att initiera alla sub-listor.

    Sedan har vi 2000 kanter som ligger lite överallt i sub-listorna. En kant måste kunna lagra ett tal upp till 1000. För att göra det krävs 10 bits (2^10 = 1024 vilket är större än 1000). Alltså kräver varje kant 10 bits.

    Nu har vi (1000 x 1) + (2000 x 10) = 21.000 bits.

    21.000 är mycket mindre än 1.000.000. Alltså är listan bättre.

    En lista tar mindre minne ju färre kanter vi har. I just detta exemplet har vi få kanter och därför är listan bättre. Matrisen blir fylld med massa nollor för att symbolisera att det inte finns koppling medan listan inte behöver symbolisera att det inte finns någon koppling.

    Varje sub-lista i listan kommer vara väldigt kort vilket är fördelaktigt.

  - **Grafen har 1000 hörn och 50000 kanter och det är viktigt att vara sparsam med minnet.**

    Jag skulle välja en närhetslista.

    Vi räknar på samma sätt som i exemplet ovan:

    Matrisen skulle ta 1000 x 1000 = 1.000.000 bits.

    Listan skulle ta upp 1000*1+50.000*10= 501.000 bits.

Den teoretiska förklaring här är samma som ovan. En matris tar konstant minne oavsett hur många kanter grafen har. En lista lagrar bara dem kanter som finns. Även i detta fall tog en lista mindre minne jämfört med en matris.

- **Det är viktigt att snabbt (på konstant tid) kunna avgöra om två hörn är grannar. Om möjligt vill du också vara sparsam med minnet.**

  Jag skulle föredra en matris.

  En en matris tar sökningen konstant tid att kolla upp om två hörn är grannar. Det är som att slå upp värden i en array. Tiden är konstant.

  I en lista tar sökningen längre tid. Och ju fler kanter desto längre tid kommer det att ta. Medan matrisen kommer vara lika snabb oberoende av antalet kanter.

- **Förklara varför DFS tar $\Theta(n^2)$ tid för en sammanhängande graf med n hörn om grafen representeras med en närhetsmatris.**

Första steget i algoritmen är att hitta alla grannar till hörnet från vilket vi börjar. I en matris måste vi då gå igenom hela kolunnen. Det tar O(n) tid. Sedan kollar vi vilka grannar dem har - O(n) och sedan vilka grannar den grannen har - O(n) osv osv. För varje gång går vi genom en kolonn i matrisen. Till slut har vi kollat igenom alla n kolonner. Och varje kolonn tar O(n). Alltså blir det O(n^2)

- Ange antalet komponenter och den största komponentents storlek när $n = 1000$.

  När jag körde min funktion som räknar ut antalet komponenter och största komponentens storlek fick jag detta:

```
Largest component size in Hash:  59
Number of components in Hash  479
Largest component size in Matrix:  59
Number of components in Matrix  479
```

  Dessa siffror skiljer sig lite från gång till gång eftersom man slumpmässigt genererar grafer varje gång.

  Här är några andra outputs:

```
Largest component size in Hash:  79
Number of components in Hash  478
Largest component size in Matrix:  79
Number of components in Matrix  478
```

```
Largest component size in Hash:  178
Number of components in Hash  513
Largest component size in Matrix:  178
Number of components in Matrix  513
```

- **Vilken datastruktur är bäst i det här fallet? Varför? Förklara genom att beräkna tidskomplexiteten för DFS med närhetsmatris samt för DFS med närhetslistor.**

  Tidskomplexiteten för matrisen blir n^2.

  Tanken är att man ska gå igenom alla grannar till start noden. För att göra det måste man gå igenom hela den kolonnen. Sedan kollar man vilka grannar den grannen har och då måste man söka igenom grannens kolonn och det tar O(n). Sedan kollar man grannens grannar osv osv.. Till slut har vi kollat igenom n kolonner och det tar O(n) att kolla igenom varje kolonn. Alltså blir det O(n^2) sammanlagt.

  Tidskomplexiteten för listan blir O(n+antalet kanter).
  Så här funkar det:

  Man går först till det elementet man börjar med i listan. Det tar O(1). Sedan kollar vi i sublistan och går igenom elementets grannar O(1)*antal_grannar_elementet_har. Sen går vi till en granne i taget och utför proceduren på nytt.

  Till slut har vi gått igenom alla element som kommer ta O(n). Och sen har vi kollat deras grannar som tar O(antal_kanter) för att kolla igenom allas grannar.

  Alltså får vi O(n+antalet_kanter) sammanlagt.

  Närhetslistan är bättre i det här fallet eftersom tidskomplexiteten för matrisen med 1000 noder och kopplingar blir O(1.000.000) och för lista bara O(2000).

  Ju större n vi använder desto mer skillnad blir det mellan matrisen och listan. Matrisen är mycket långsammare eftersom tidskomplexiteten växer kvadratiskt. Jag har provat att jämföra prestandan på väldigt små n (runt 50) och även där körs listan snabbare.