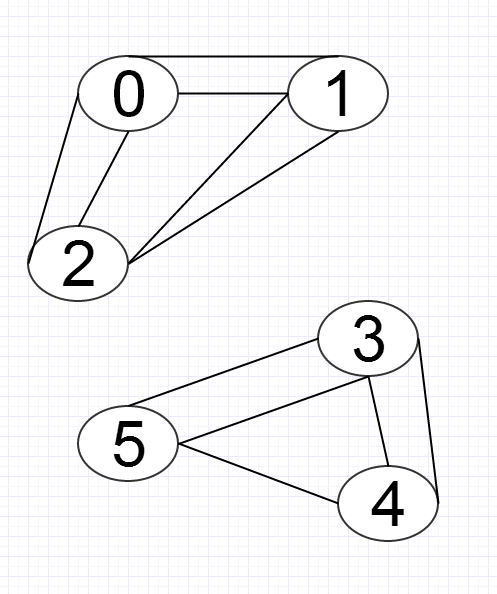
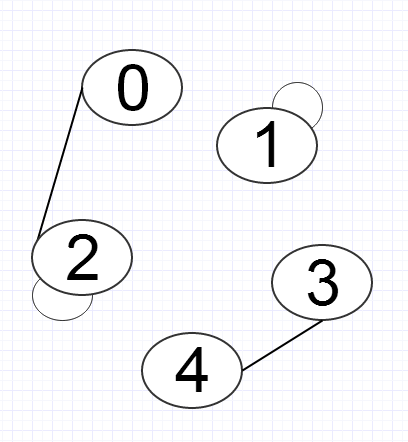
## Rita en oriktad graf med 6 hörn, 10 kanter och 2 sammanhängande komponenter. Går det att rita en graf med 5 hörn, 4 kanter och 3 komponenter?

En version på en graf med 6 hörn, 10 kanter och 2 sammanhängande komponenter.

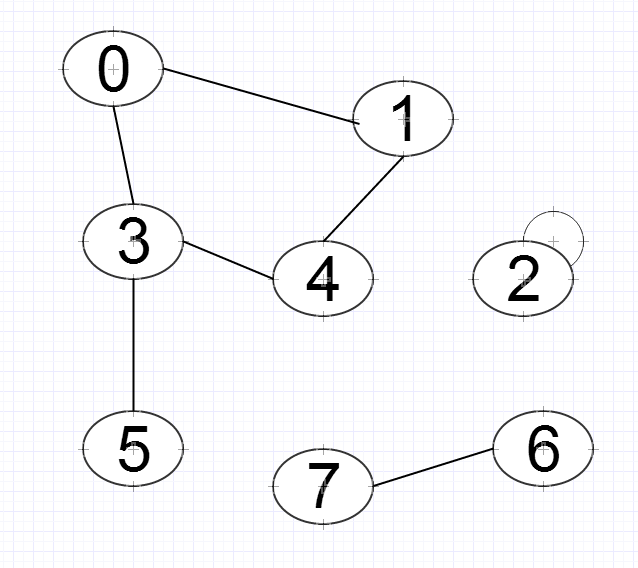


En version på en 5 hörn, 4 kanter och 3 sammanhängande komponenter.

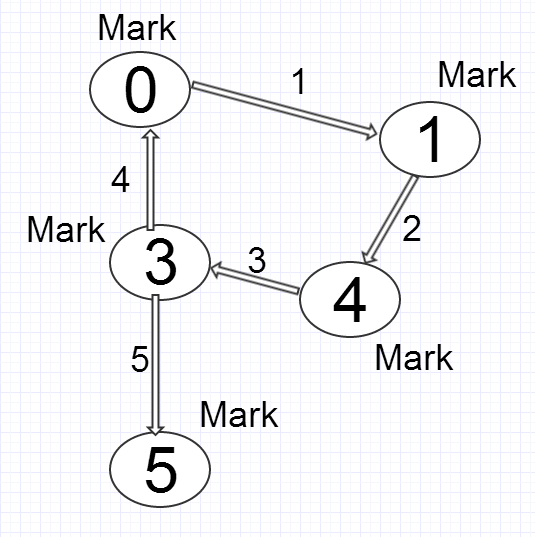


## Låt G vara en oriktad graf som består av 8 hörn numrerade från 0 till 7 och kantmängden {(0,1), (0,3), (1,4), (2,2), (3,4), (3,5), (6,7)}.

### Rita G.

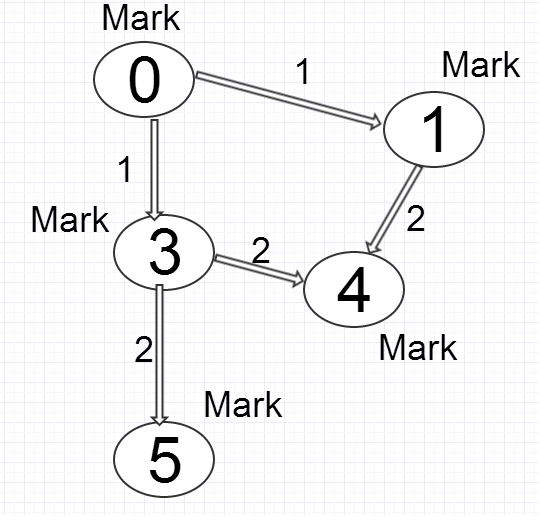


### Ange ordningen som hörnen besöks vid en Djupetförstsökning (DFS) med start i hörn 0.



Vid steg 4 och 5 så ges ett return för 0 är redan ”marked” och 5 har inte flera grannar i komponenten.

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



Vid steg 2 så ges ett return för att 4 och 5 har inga flera grannar i komponenten.

## 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.

Till detta exempel så skulle jag ha närhetsmatriser. Detta är för att det inte är så mycket med data. Samt så är det ganska så sparsamt med minne men tanke på att du hjälp kan bestämma hur stora vektorerna ska vara.

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

Till detta exempel så skulle jag använda närhetslistor. Detta är för att närhetslistor är har något bättre prestanda än närhetsmatriser. Samt så är det lättare att hålla kolla på alla kanter i list form.

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

Detta göra lättas med närhetsmatris, då du behöver hitta de två hörnen och kolla om dem blir en etta.

## Förklara varför DFS tar Θ(n2) tid för en sammanhängande graf med n hörn om grafen representeras med en närhetsmatris.

Oavsett om det är en riktad eller oriktad så ges DFS alltid Θ(n2) för att du behöver på igenom n\*n element (n2) i en riktad graf och (n\*n)/2 i element(n2/2) i en oriktad graf.

I det riktade fallet så blir det: Θ(n2) och i det oriktade fallet Θ(n2/2) = Θ(n2)

## Ange antalet komponenter och den största komponentens storlek när n = 1000.

Efter att ha kört programmet med n = 1000 ett par gånger.

Så blir antalet komponenter ca 500 och största komponenten ca 100. Men båda det sakerna varierar väldigt mycket.

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

I detta fall så är närhetsmatriser inte så bra då dess tidskomplexitet alltid är Θ(n2). Men i fallen för närhetslistor är dess tidskomplexitet mindre.

Listan måste gå igenom n element i listan och för varje index i listan gå igenom varje nycken i mappen.

n1 + (nycklar) + n2 + (nycklar) + .... + nn + (nycklar)

(n1 + n2 + ... + nn) + ((nycklar för n1) + (nycklar för n2) + ... + (nycklar för nn))

n + m

Detta gör att tidskomplexiteten blir O(n + m) där n är för alla hörn i listan och m är för alla nyckar i mapparna.