DD1352 Algoritmer, datastrukturer och komplexitet, hösten 2015
Lösningsförslag till mästarprov 2: komplexitet

1. **Flyktingplaceringsproblemet**
   Vi inför ett mål M i indata i formuleringen av maximeringsproblemet som beslutsproblem:

   Indata: ett heltal n som anger antal platser på flyktingboendet, ett mål M, en lista $\{(a_i, p_i)\}$ med m stycken talpar som representerar flyktinggrupper. Flyktinggrupp i består av a_i stycken personer och deras prioritering (på en skala från 1 till 5) av den ort som det aktuella flyktingboendet ligger i anges av p_i.

   Fråga: Finns det en delmängd D av grupperna så att $\sum_{(a,p)\in D} a = n$ och $\sum_{(a,p)\in D} ap \geq M$?

   Först visar vi att flyktingplaceringsproblemet ligger i NP. För varje ja-instans ska en lösning kunna verifieras i polynomisk tid. Låt lösningen vara delmängden av grupperna, till exempel representerad som en delmängd $D \subseteq \{1, \ldots, m\}$. Då behöver verifieringsalgoritmen bara kolla att antalet personer i delmängderna är n och att målfunktionens värde är minst M.

   VerifyPlacement$(n, M, \{(a_i, p_i)\}_1^m, D)$ =
   ```
   psum ← 0
   gsum ← 0
   foreach i ∈ D do
       psum ← psum + a_i
       gsum ← gsum + a_i p_i
   if psum ≠ n or gsum < M then return false
   return true
   ```

   Eftersom D har högst m element blir tidskomplexiteten $O(m)$ med enhetskostnad, så villkoret om att verifieringen ska gå i polynomisk tid är uppfyllt.

   Vi visar nu att problemet är NP-svårt med en reduktion av delmängdssumma. Idén är att vi bygger en flyktinggrupp för varje tal i mängden, med lika många personer som talet anger. Vi låter alla prioriteringar vara ett, antal platser på flyktingboendet vara lika med den sökta summan och och målet lika med antalet platser på flyktingboendet.

   Delmängdssumma$(\{b_i\}_1^m, K)$ =
   ```
   return PlacementDecision(K, K, {(b_i, 1)}_1^m)
   ```

   Om det finns en delmängd som har summan K så väljer vi motsvarande flyktinggrupper till boendet. Eftersom delmängden har summan K så kommer boendet att bli precis fyllt av dessa grupper, och målfunktionen får också värdet K, vilket är lika med målet. Detta är alltså en ja-instans till flyktingplaceringsproblemet.

   Anta nu istället att vi har en ja-instans till flyktingplaceringsproblemet som skapats av reduktionen och vill konstruera en ja-instans till delmängdsproblemet. Eftersom antalet personer i dom valda grupperna är precis antalet platser i boendet, dvs K, så har dom motsvarande talen i delmängdsproblemet också att ha summan K, vilket är den sökta summan.

   Reduktionen ovan tar linjär tid i indatas storlek, vilket innebär att reduktionen är en polynomisk Karpreduktion. Alltså är flyktingplaceringsproblemet NP-fullständigt.

2. **Konspirationsdetektionsproblemet**

   Anta att nätverket består av en mängd personer P och en mängd bekantskapspar A av personer som är bekanta med varandra. Den förmodade spindeln kallas X.

Vi kan verifiera en ja-lösning till beslutsproblemet i polynomisk tid på följande sätt. Låt lösningen vara mängden konspiratörer, $C \subseteq P$. Verifikatorn kollar först att ingen konspiratör är bekant med någon annan konspiratör, sedan att alla konspiratörer är bekanta med spindeln och slutligen att varje person i nätverket som inte redan är konspiratör eller spindeln är bekant med minst en konspiratör.

```
VerifyConspiracy((P, A), X, C) =
    foreach p1 ∈ C do
        foreach p2 ∈ C do
            if (p1, p2) ∈ A then return false // är konspiratörerna bekanta?
    foreach p ∈ C do
        if (X, p) ∉ A then return false // är konspiratörerna bekanta med spindeln?
    foreach p ∈ P − C − X do
        if |{c ∈ C : (c, p) ∈ A}| = 0 then return false // är övriga bekanta med någon konspiratör?
    return true
```

Om vi representerar A som en boolesk $|P| \times |P|$-matris tar verifikationen tid $O(|P|^2)$ som är polynomiskt.

För att visa NP-svårighet kan vi enklast reducera 3CNFSAT. Idén är att vi skapar två personer för varje boolesk variabel, en som representerar sanningsvärdet sant och en som representerar falskt, och dessa är bekanta med varandra. Vi skapar också en person för varje klausul. Alla variabelpersoner är bekanta med spindeln och därmed potentiella konspiratörer. Varje klausulperson är bekanta med dom personer som motsvarar dom literaler som ingår i klausulen.

```
3CNFSAT(φ) =
    Anta att φ = c1 ∧ c2 ∧ ··· ∧ cm över variablerna x1, ... xn
    spider ← new Person
    P ← spider
    A ← ∅
    for i ← 1 to n do
        xpi ← new Person
        xni ← new Person
        P ← P ∪ {xpi, xni}
        A ← A ∪ {(xpi, xni), (spider, xpi), (spider, xni)}
    for j ← 1 to m do
        clj ← new Person
        foreach xi ∈ cj do A ← A ∪ {(xpi, clj)}
        foreach x̄i ∈ cj do A ← A ∪ {(xni, clj)}
    return ConspiracyDecision((P, A), spider)
```

Bevis av reduktionen: Vi ska visa att det finns en konspiration om och endast om ϕ är satisfierbar. Anta att det finns en variabeltilldelning som satisfierar ϕ. Låt konspiratörerna vara dom n personerna som motsvarar den satisfierande variabeltilldelningen, dvs så att om x_k är sann så ska xp_k vara konspiratör och om x_k är falsk så ska xn_k vara konspiratör. Dessa konspiratörer är alla bekanta med spindeln men inte med varandra. Eftersom varje klausul är satisfierad så är motsvarande klausulperson bekant med minst en konspiratör, nämligen den konspiratör som motsvarar den sanna literalen i klausulen. Alltså har vi en konspiration.

Åt andra hållet: Anta att vi har en konspiration. Eftersom xp_k och xn_k är bekanta med varandra och med spindeln men inte med någon annan potentiell konspiratör så måste exakt en av xp_k och xn_k vara med i konspirationen. Om xp_k är konspiratör låter vi motsvarande variabel x_k vara sann och om xn_k är konspiratör låter vi den vara falsk. Detta ger oss en

variabeltilldelning. Att denna variabeltilldelning satisfierar varje klausul c_j beror på att klausulpersonen cl_j (som inte är bekant med spindeln och därför inte själv kan vara konspiratör) måste vara bekant med någon konspiratör, och dom enda konspiratörer som cl_j är bekant med motsvarar tilldelade variabler. Därför satisfierar tilldelningen hela formeln ϕ.

Reduktionen går i polynomisk tid (närmare bestämt linjär tid i nätverkets storlek), så problemet är alltså NP-fullständigt.

3. **Konstruktion av konspiration**
Vi konstruerar en lösning genom att prova att ta bort en bekantskapsrelation i taget från dom personer som är bekanta med spindeln (potentiella konspiratörer). Om problemet då inte längre har någon lösning lägger vi tillbaka bekantskapsrelationen igen. När vi är klara kommer den återstående mängden personer som är bekanta med spindeln att utgöra konspiratörerna i konspirationen.

```
Conspiracy((P, A), X) =
    if not Spider((P, A), X) then return "ingen lösning"
    C ← ∅ // här samlar vi konspiratörer
    R ← ∅ // här samlar vi personer som inte är konspiratörer
    A' ← A // kopia av bekantskapsrelationer som vi ska modifiera
    foreach p : (X, p) ∈ A do
        inv Varje lösning till (P, A'), X är lösning till ursprungsproblemet (P, A), X
        inv Varje person i C är med i varje lösning till (P, A'), X
        if Spider((P, A' − {(X, p)}), X) then
            A' ← A' − {(X, p)}
            R ← R ∪ {p}
        else
            C ← C ∪ {p}
    foreach p ∈ C do print p
```

För att konstruera lösningen görs högst $n + 1$ anrop, där n är antalet personer i det sociala nätverket. Tidskomplexiteten blir därför $O((n + 1)B(n))$.

Korrektheten för algoritmen inses på följande sätt. I varje varv i den stora slingan provar algoritmen att ta bort en bekantskapsrelation (X, p). Om problemet fortfarande har en lösning så måste den lösningen vara en lösning även till det tidigare problemet eftersom p då inte är en konspiratör och bekantskapsrelationer med spindeln inte är relevanta för ickekonspiratörer. Om problemet inte längre har en lösning måste p vara en konspiratör och algoritmen lägger till den till mängden C. Detta bevisar invarianten.

I varje varv i slingan läggs en bekant till spindeln X till antingen C eller R. När vi gått igenom alla bekanta till spindeln finns inte längre några bekantskapsrelationer kvar i A' mellan spindeln och personer i R. Det betyder att dom enda personerna som kan vara konspiratörer är personerna i C. Invarianten säger att varje person i C är konspiratör. Det betyder att mängden konspiratörer måste vara precis C.

I den avslutande slingan skrivs just personerna i C ut. Därför är algoritmen korrekt.