Algoritmer, datastrukturer och komplexitet, hösten 2011

Uppgifter till övning 12

Komplexitetsklasser och repetition

Övningens ena timme ägnas åt genomgång av mästarprov 2.

Uppgifter på komplexitetsklasser

co-NP-fullständighet Ett diskret tekniskt diagnosproblem kan modelleras på följande sätt: komponenttillstånd, systemtillstånd och omgivningstillstånd representeras av booleska variabler och själva systemet definieras med en boolesk formel φ. Man vet att i alla *möjliga* (fungerande) världar kommer systemvariablerna att ha värden så att φ är sann.

Anta att komponenttillstånden alla är tvåvärda och att en komponent c därför representeras av en boolesk variabel x_c (där sant betyder att komponenten fungerar och falskt att den är trasig). Vi vet att en komponent c är trasig om den formel som vi får om vi i φ sätter in värden för alla variabler som representerar kända komponent-, system- och omgivningstillstånd samt sätter x_c till sann inte är satisfierbar.

Visa att det är co-NP-fullständigt att avgöra ifall c är trasig.

Komplexitetsklassrelation PSPACE är komplexitetsklassen som består av alla språk för vilka det existerar en *deterministisk* turingmaskin som känner igen språket i polynomiskt *minne*. EXPTIME består av alla språk för vilka det existerar en *deterministisk* turingmaskin som känner igen språket i exponentiell tid. Visa att PSPACE$\subseteq$EXPTIME.

Blandade uppgifter från gamla tentor

Första uppgiften är en typisk första uppgift på teoritentan. Övriga uppgifter är typiska uppgifter på muntliga tentan.

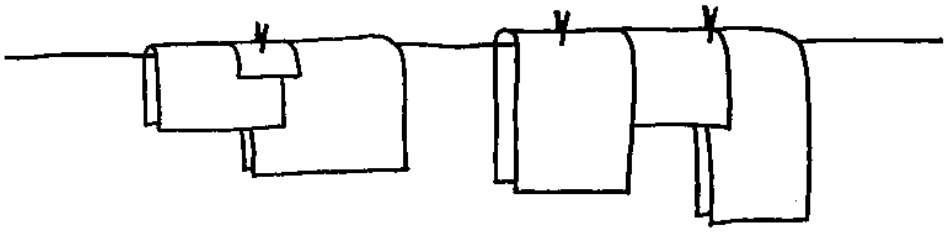
1. Är följande påståenden sanna eller falska? För varje deluppgift ger riktigt svar 1 poäng och ett övertygande bevisat riktigt svar 2 poäng,

   a) Problemet att avgöra ifall ett tal med n siffror är ett primtal ligger i komplexitetsklassen co-NP.

   b) Det finns en konstant $c > 1$ så att $n^3 \in O(c^{\log n})$.

   c) Binära träd implementerar man vanligen genom att införa två pekare i varje post (`left` och `right`). När man implementerar ternära träd (där varje nod har tre söner) går det inte att klara sig med mindre än tre pekare i varje post.

2. (algoritmkonstruktion, betyg C) På ett klädstreck har Viggo hängt n stycken handdukar av olika storlek. Viggo vill att handdukarna ska göra ett harmoniskt intryck för betraktaren. Därför var han mycket noga vid utplaceringen och har tagit mått på exakt var handdukarna hänger längs strecket, så att ingen ska flytta på dom. För att inte vinden ska störa ordningen vill Viggo säkra handdukarna med klädnypor, men han vill inte använda fler klädnypor än absolut nödvändigt. Det räcker ju att varje handduk kläms åt av en nypa, och två eller flera handdukar som hänger över varandra kan dela på samma nypa.



Beskriv en algoritm (i ord eller med pseudokod) som så snabbt som möjligt beräknar var på strecket man ska sätta nyporna för att det ska gå åt så få nypor som möjligt. Motivera att algoritmen är korrekt och analysera med enhetskostnad.

Indata är n stycken par av tal (v_1, h_1), (v_2, h_2), ..., (v_n, h_n) där varje par talar om var vänsterkanten respektive högerkanten på en handduk är. Alla mått avser avståndet från början av klädstrecket till en handdukskant. Paren är inte sorterade på något sätt. Utdata ska vara ett antal tal som anger var man bör sätta klädnyporna (en klädnypa antas ha bredd 0).

För att du ska få maximal poäng måste din algoritm gå i tid $O(n \log n)$.

3. (komplexitet, betyg C) I en stor organisation som Teknis finns det många grupper av personer, t ex lärare på Nada, lärare på F, elever på kursen Algoritmer, datastrukturer och komplexitet, medlemmar i Teknologkören, etc. Varje individ är med i minst en grupp, men kan vara med i många grupper. Nu vill rektor skapa en grupp av informatörer som snabbt ska kunna föra ut information till alla individer på Teknis. Han vill att varje grupp ska vara representerad i informatörsgruppen (dvs minst en medlem i varje grupp ska vara med i informatörsgruppen), men han vill samtidigt att informatörsgruppen ska vara så liten som möjligt.

   Detta är ett exempel på ett allmänt problem där man givet en uppsättning grupper vill hitta en så liten skara av representanter som möjligt.

   a) Formulera problemet matematiskt som ett mängdproblem och beskriv det samtidigt som ett beslutsproblem.

   b) Visa att problemet är NP-fullständigt.

4. (algoritmkonstruktion, betyg A; komplexitet, betyg C; svår uppgift!) Tumstocksproblemet kan beskrivas på följande sätt. En tumstock består av en kedja av träbitar som är fastsatta med gångjärn i ändarna. Det är möjligt (men inte nödvändigt) att vika tumstocken vid varje gångjärn. Det kan vara så att träbitarna som tumstocken är gjord av har olika längd. Då är det inte längre självklart hur man ska vika ihop tumstocken så att den blir så kort som möjligt (och går ner i snickarens ficka). Problemet är alltså att, givet en tumstock, vika ihop den så att den blir så kort som möjligt.

   a) Formulera tumstocksproblemet matematiskt som ett beslutsproblem.

   b) Visa att tumstocksproblemet är NP-fullständigt. Reducera till exempel mängdpartitioneringsproblemet till tumstocksproblemet.

   c) Anta att träbitarna har heltalslängder och att den längsta träbiten är $17n$, där n är antalet träbitar som tumstocken är sammansatt av. Bestäm komplexiteten för detta problem.

5. (komplexitet, betyg C) MAX 2$\wedge$SAT är ett optimeringsproblem som definieras som beslutsproblem på följande sätt.

INMATNING: Ett positivt heltal K mellan 1 och n samt n klausuler där varje klausul består av en enda literal eller två literaler kombinerade av operatorn $\wedge$. Exempel: $x_1 \wedge \overline{x}_3$, $x_2 \wedge x_3$.

PROBLEM: Finns det en variabeltilldelning som satisfierar minst K klausuler?

Visa att denna beslutsproblemsversion av MAX 2$\wedge$SAT är NP-fullständig. Du kan exempelvis använda dej av att MAX 2SAT – det motsvarande problemet med operatorn $\vee$ istället för $\wedge$ – är NP-fullständigt.

Lösningar till uppgifter på komplexitetsklasser

Lösning till co-NP-fullständighet
För att visa att problemet ligger i co-NP så ska vi visa att vi i polynomisk tid kan verifiera en nej-lösning, det vill säga att c inte är trasig. Vi vet att c inte är trasig om och endast om det finns en variabeltilldelning som satisfierar formeln. Om vi gissar variabelvärden behöver vi alltså bara verifiera att formeln med denna variabeltilldelning är sann. Detta går att göra i polynomisk tid.

För att visa att problemet är co-NP-svårt reducerar vi co-SAT, alltså problemet att avgöra ifall en given boolesk formel ϕ *inte* är satisfierbar. Eftersom SAT är NP-fullständigt så är per definition co-SAT co-NP-fullständigt.

Givet ϕ, konstruera ett system som innehåller den extra komponenten c (representerad av x_c) och som definieras av formeln $\varphi = \phi \vee \neg x_c$.

Notera nu att om x_c sätts till sann blir $\varphi = \phi$ så problemet att avgöra ifall c är trasig är precis samma som problemet att avgöra ifall ϕ inte är satisfierbar. Alltså är reduktionen korrekt. □

Lösning till Komplexitetsklassrelation
Om en turingmaskin använder polynomiskt minne finns det en konstant k så att antalet använda rutor på bandet är $O(n^k)$ där n är indatas längd. Om alfabetet består av tre tecken (0, 1, blank) så är antalet olika möjliga konfigurationer på bandet begränsat av $3^{O(n^k)}$, antalet möjliga platser för läs/skrivhuvudet är $O(n^k)$ och antalet möjliga tillstånd i turingmaskinen är ändligt, dvs $O(1)$. Det totala antalet konfigurationer för en turingmaskin som använder polynomiskt minne är alltså $O(n^k) \cdot 3^{O(n^k)}$, dvs exponentiellt i n. Eftersom turingmaskinen inte kan återkomma till samma konfiguration flera gånger (då skulle den gå i en oändlig slinga) så är detta också en övre gräns på tiden. Alltså kan varje problem som kan lösas med polynomiskt minne lösas i exponentiell tid. □

Lösningar till blandade uppgifter från gamla tentor

1. a) *Sant.* Problemet ligger i co-NP om komplementproblemet ligger i NP. Komplementproblemet är i detta fall att avgöra ifall ett tal med n siffror kan faktoriseras i minst två faktorer (större än 1). Detta problem ligger i NP eftersom en lösning (dvs en faktorisering av talet) kan verifieras i polynomisk tid (genom att man multiplicerar ihop faktorerna och kollar att produkten blir det givna talet).

   b) *Sant.* Om vi antar att $\log n$ är logaritmen i basen 2 så vet vi att $c^{\log n} = 2^{\log c^{\log n}} = 2^{\log n \log c} = n^{\log c}$. Om vi väljer $c \geq 8$ så är $\log c \geq 3$ och $n^3 \in O(n^{\log c}) = O(c^{\log n})$.

   c) *Falskt.* För allmänna träd räcker det med två pekare i varje post (`firstson` och `next`).

2. En girig algoritm löser problemet i tid $O(n \log n)$.

   1. Sortera alla paren dels med avseende på vänsterkanten och dels med avseende på högerkanten. Lagra resultaten i två listor, *Vsort* och *Hsort*. Håll under sorteringen reda på var varje post i den ena listan hamnar i den andra listan. Med heapsort tar detta tid $O(n \log n)$.

2. Så länge det finns några par kvar i listan *Hsort*:

   2.1 Ta det första återstående paret, säg (v, h). Eftersom paren har sorterats efter högerkanterna är (v, h) den handduk vars högerkant ligger längst till vänster.

   2.2 Sätt en klädnypa längst till höger på denna handduk, dvs precis till vänster om h.

   2.3 Ta bort alla par vars vänsterkant är till vänster om h, dvs ta bort alla handdukar som den just ditsatta klädnypan nyper fast. Detta gör man genom att plocka par från början av *Vsort* som ju är sorterad i stigande vänsterkantsordning. För varje par som plockas bort från *Vsort* ska samma par plockas bort från *Hsort*.

Till slut har alla par plockats bort och algoritmen avslutas. I steg 2 behandlas varje par bara en gång. Tiden för steg 2 blir alltså $O(n)$. Hela algoritmen tar därför tid $O(n \log n)$.

Korrekthet: Eftersom bara par som fått en klädnypa i sig blir borttagna så kommer alla par att sitta fast när algoritmen har genomförts. Nu ska vi bara visa att antalet använda klädnypor är minimalt. Den handduk som har den vänstraste högerkanten (dvs den som är allra först i *Hsort*) måste få en klädnypa i sig. Om man sätter fast handduken i punkten p så kommer alla handdukar med vänsterkant mindre än p att sitta fast, eftersom det inte finns någon handduk som har sin högerkant till vänster om p. För att sätta fast så många handdukar som möjligt så ska man välja p så stort som möjligt, dvs så nära handdukens högerkant som möjligt. Samma resonemang tillämpas sedan på dom övriga handdukarna.

3. a) Låt k vara ett positivt heltal, S vara mängden av personer och $C = \{C_1, \ldots, C_m\}$ vara dom m grupperna. Problemet är att hitta en delmängd $S' \subseteq S$ med högst k element så att $S' \cap C_i \neq \emptyset$ för $1 \leq i \leq m$. Detta problem har på engelska namnet *hitting set*.

   b) Problemet ligger i NP eftersom man kan gissa vilka k element som ska ligga i S' och verifiera att $S' \cap C_i \neq \emptyset$ för $1 \leq i \leq m$ i polynomisk tid.

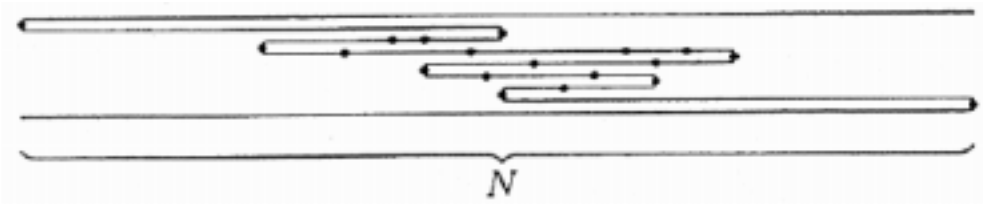
   Problemet är NP-svårt eftersom det är en generalisering av hörntäckningsproblemet som är känt NP-fullständigt. Givet en graf $G = (V, E)$, låt $S = V$ och $C = E$. En hörntäckning av storlek k motsvarar precis en delmängd $S' \subseteq S$ av storlek k som innehåller minst ett element från varje C_i.

4. a) Indata är n positiva tal $l_1, \ldots, l_n$, som i tur och ordning anger längderna av dom n träbitarna som ingår i tumstocken, samt ett positivt tal K som anger längden av snickarens ficka. Frågan är om det går att välja riktningar $r_1, \ldots, r_n$, där $r_i \in \{-1, 1\}$, för dom n träbitarna så att om man viker tumstocken enligt riktningarna så blir dess längd högst K, dvs $\max_{j \in [0..n]} \left\{ \sum_{i=1}^{j} r_i l_i \right\} - \min_{j \in [0..n]} \left\{ \sum_{i=1}^{j} r_i l_i \right\} \leq K$.

   b) Tumstocksproblemet tillhör NP eftersom man kan gissa värdena på $r_1, \ldots, r_n$ och verifiera att tumstockens längd högst är K i polynomisk tid.

   Vi visar att det är NP-svårt genom att reducera partitioneringsproblemet som är välkänt NP-fullständigt. Indata till partitioneringsproblemet är n positiva tal $t_1 \ldots, t_n$ och frågan är om talen kan delas upp i två grupper så att deras summa är lika.

   Givet $t_1 \ldots, t_n$, konstruera indata till tumstocksproblemet på följande sätt: $l_1 = N$, $l_2 = N/2$, $l_{i+2} = t_i$ för $1 \leq i \leq n$, $l_{n+3} = N/2$, $l_{n+4} = N$ och $K = N$, där $N = \sum_{i=1}^{n} t_i$. Visa nu att det finns en jämn partitionering om och endast om det finns en vikning av tumstocken så att längden blir högst N.



Om vi har en partitionering så kan vi vika tumstocken så att träbitar som motsvarar tal i den första gruppen viks åt vänster (riktning -1), bitar som motsvarar tal i den andra gruppen viks åt höger (riktning $+1$), $r_1 = -1$, $r_2 = 1$, $r_{n+3} = 1$, $r_{n+4} = -1$. Det är lätt att kontrollera att längden av tumstocken blir precis N, se figuren.

Omvänt, om vi har en vikning som gör att tumstocken ryms i fickan av storlek N, så kan vi anta att $r_1 = 1$. Om $r_1 = -1$ så tar vi först och byter tecken på alla riktningar, dvs vänder på tumstocken, som förstås fortfarande har samma längd. Eftersom längden av den första träbiten är N så måste nästa bit ha riktning $r_2 = -1$ och föra tillbaka till en position mitt på den första träbiten. På samma sätt ser vi att den sista och den näst sista träbiten måste vara riktade åt olika håll och att den näst sista träbiten börjar i en position som är mitt på den sista träbiten. För att längden av tumstocken ska vara högst N så måste uppenbarligen den andra träbitens ändes position överensstämma med den näst sista träbitens början. Detta innebär att dom mellanliggande bitarna måste börja och sluta på samma position, dvs $\sum_{i=3}^{n+2} r_i l_i = 0$, vilket kan skrivas som $\sum_{i:3\le i\le n+2 \wedge r_i=1} l_i - \sum_{i:3\le i\le n+2\wedge r_i=-1} l_i = 0$. Eftersom dessa träbitars längder precis motsvarar tal i partitioneringsproblemet så får vi en lösning till detta problem genom att vi partitionerar efter hur dom motsvarande träbitarna är riktade.

c) Problemet kan lösas i polynomisk tid med dynamisk programmering genom att man håller reda på alla delmängdssummor som kan uppkomma. Om den längsta träbiten är $17n$ vet vi att det för alla delsummor måste gälla att $-17n^2 \le \sum_{i=1}^{j} r_i l_i \le 17n^2$. Skapa en boolesk array $b[0..n, -17n^2..17n^2]$ där alla element från början är falska. Algoritmen ska fylla arrayen så att $b[j,k]$ är sant omm det finns värden på r_i så att $\sum_{i=1}^{j} r_i l_i = k$. Rekursionsekvationen blir $b[j,k] = b[j-1, k+l_j] \vee b[j-1, k-l_j]$. Skapa också en boolesk array $ends[0..n, -17n^2..17n^2, -17n^2..0, 0..17n^2]$ där $ends[j,k,l,r]$ är sant om det finns någon tumstocksvikning av dom j första träbitarna som slutar i position k, har sin vänstraste punkt i position l och sin högraste punkt i position r.

$b[0,0] \leftarrow$ sant
for $j \leftarrow 1$ **to** n **do for** $k \leftarrow -17n^2$ **to** $17n^2$ **do**
  if $b[j-1,k]$ **then**
    $b[j, k-l_j] \leftarrow b[j, k+l_j] \leftarrow$ sant
    for $l \leftarrow -17n^2$ **to** 0 **do for** $r \leftarrow 0$ **to** $17n^2$ **do**
      if $ends[j-1,k,l,r]$ **then**
        $ends[j, k-l_j, min(l, k-l_j), r] \leftarrow ends[j, k+l_j, l, max(r, k+l_j)] \leftarrow$ sant
$minlength \leftarrow 17n^2$
for $k \leftarrow -17n^2$ **to** $17n^2$ **do**
  if $b[n,k]$ **then**
    for $l \leftarrow -17n^2$ **to** 0 **do for** $r \leftarrow 0$ **to** $17n^2$ **do**
      if $ends[j,k,l,r] \wedge (r - l < minlength)$ **then** $minlength \leftarrow r - l$
return $(minlength)$

Det är lätt att verifiera att algoritmen returnerar den minsta tumstockslängd som kan uppnås. Algoritmen tar tid $O(n^7)$, vilket är polynomiskt (men ändå ganska långsamt!).

5. MAX 2∧SAT ligger i NP eftersom det är lätt att verifiera att en variabeltilldelning (en lösning) satisfierar minst K av klausulerna. Vi visar att det är NP-svårt genom att reducera MAX 2-SAT. Varje 2-SAT-klausul med en enda literal flyttar vi oförvanskad över till MAX 2∧SAT-probleminstansen. För varje klausul $l_i \vee l_j$ i MAX 2-SAT-instansen konstruerar vi tre klausuler i MAX 2∧SAT-probleminstansen: $l_i \wedge l_j$, $\overline{l}_i \wedge l_j$ och $l_i \wedge \overline{l}_j$. Vi ser att $l_i \vee l_j$ är sann omm en av dom tre konstruerade klausulerna är sann. Om $l_i \vee l_j$ är falsk så är alla dom tre konstruerade klausulerna falska. Alltså motsvarar antalet satisfierade MAX 2-SAT-klausuler precis antalet satisfierade MAX 2∧SAT-klausuler. Välj alltså målet K för det konstruerade problemet till samma värde som K för MAX 2-SAT-problemet.