DD2350 Algoritmer, datastrukturer och komplexitet, hösten 2020
Lösningsförslag till mästarprov 2: komplexitet

1. **Trött robot**

Vi bevisar att problemet ligger i NP genom att visa hur man i polynomisk tid kan verifiera en ja-instans givet en beskrivning av robotens väg genom rutnätet. Låt robotens k rörelser genom rutnätet beskrivas av en sträng $movements[1..m]$, där varje rörelse beskrivs av ett tecken: $\vee$ (gå snett ner till vänster), $>$ (gå snett ner till höger), $\wedge$ (backa snett upp till höger), $<$ (backa snett upp till vänster). Verifikatorn måste kontrollera att roboten håller sig inom rutnätet, att den backar högst en gång, att den slutar i ruta (n, n) och att den på sin väg plockar upp exakt m gram osmium.

```
VerifyTiredRobot(n, m, Os[1..n, 1..n], movements) =
    x ← 1; y ← 1
    backings ← 0
    k ← movements.length(); sum ← Os[x, y]
    if k > 2n then return false
    for i ← 1 to k do
        case movements[i]:
            '∨': x ← x + 1
            '>': y ← y + 1
            '∧': x ← x − 1; backings ← backings + 1
            '<': y ← y − 1; backings ← backings + 1
            default: return false
        if x < 1 or x > n or y < 1 or y > n or backings > 1 then return false
        sum ← sum + Os[x, y]; Os[x, y] ← 0
    return x = n and y = n and sum = m
```

Verifieringen går i linjär tid i längden av indata med både enhetskostnad och bitkostnad, eftersom slingan går $O(n)$ varv och algoritmen bara gör konstant arbete i varje varv av slingan, förutom summeringen av tal som tar linjär tid i talens storlek med enhetskostnad.

Vi visar sedan att problemet är NP-svårt genom att reducera problemet delmängdssumma till tröttrobotproblemet.

```
SubsetSum(P[1..N], K) =
    n ← N + 2
    for i ← 1 to n do
        for j ← 1 to n do
            Os[i, j] ← 0
    for i ← 1 to N do
        Os[i + 1, i + 1] ← P[i]
    m ← K
    return TiredRobot(n, m, Os[1..n, 1..n]])
```

Vi lägger alltså delmängdsproblemets ickenegativa tal på diagonalen i osmiummatrisen men lämnar $Os[1, 1]$ och $Os[n, n]$ tomma. Inget osmium finns på något annat ställe i rutnätet.

Reduktionen är uppenbart polynomisk (linjär i indatas längd). Låt oss visa att den är korrekt, det vill säga att det existerar en delmängd med summa K om och endast om det existerar en tillåten väg för roboten som ger exakt $m = K$ gram osmium.

Anta att det finns en delmängd $A \subseteq P$ med summa K. Låt roboten röra sig genom rutnätet så att den passerar dom diagonalrutor som motsvarar talen i A men inte några andra diagonalrutor (förutom start- och slutrutorna). Det är lätt att undvika övriga diagonalrutor, till exempel genom att man rundar övriga diagonalrutor på höger sida. Denna väg uppfyller kraven på en tillåten robotväg som har summa $m = K$ och inte använder någon backning alls.

Åt andra hållet, anta att det finns en tillåten robotväg som ger exakt $m = K$ gram osmium. Eftersom det bara finns osmium på diagonalen så räcker det att hålla koll på vilka diagonalrutor som vägen passerar. Summan av dessa är K, och summan av motsvarande tal i P är förstås också K, varför dessa tal bildar en delmängd med sökt summa.

Därmed är NP-fullständigheten bevisad.

2. **Färggrann kub av tärningar**
Låt KUB vara problemet i uppgiften, och låt KAKEL vara problem 1 från föreläsning 23. Vi vet att KAKEL är ett oavgörbart problem. För att visa att KUB också är oavgörbart reducerar vi KAKEL till KUB med en karpreduktion.

   För varje typ av kakelplatta skapar vi en typtärning som har färgen gul på både framsida och baksida, och samma färger som kakelplattan på de övriga fyra sidorna: uppåt, höger, nedåt och vänster på plattan motsvarar översida, högersida, undersida respektive vänstersida på tärningen.

   Reduktionen går i linjär tid (vilket i högsta grad är ändligt).

   Det går att hitta ett korrekt kakelmöster om och endast om det för varje n går att pussla ihop en $n \times n \times n$-kub med motsvarande tärningar:

   (a) Antag att det finns ett korrekt kakelmönster. Då kan vi pussla ihop varje lager i kuben, från det främre till det bakre, på samma sätt som i kakelmöstret. De olika lagren kan alltid kombineras eftersom alla tärningar är gula på både fram- och baksida.

   (b) Antag att det finns en korrekt kub. Då kan vi göra en korrekt kakling genom att använda mönstret från det främre lagret i kuben.

   Reduktionen är alltså korrekt.

   Låt oss nu visa att det går att verifiera nej-instanser i ändlig tid, givet ett motexempel. Att svaret är nej innebär att det inte går att pussla ihop en $n \times n \times n$-kub för varje n. Det betyder att det existerar (minst) ett specifikt n för vilket det inte går att pussla ihop en $n \times n \times n$-kub av typtärningarna. Låt detta värde n utgöra vårt motexempel. Det vi ska verifiera är då att det inte är möjligt att pussla ihop en $n \times n \times n$-kub av typtärningarna.

   Verifikatorn testar alla möjliga tärningskombinationer som ger en kub av storlek $n \times n \times n$. Om någon av dessa kombinationer bildar en korrekt kub (som överensstämmer i färg på alla sidor som ligger mot varandra) så returneras falskt. Om inte någon av kombinationerna bildar en korrekt kub returneras sant.

   Eftersom det finns m stycken olika tärningar att testa på varje position i kuben så blir tidskomplexiteten $O(m^{(n^3)})$, vilket är ändlig tid.

   Därmed har vi visat att KUB är co-R.E.-fullständigt.

3. **Konstruktion av väg för trött robot**
Vi antar att det finns en algoritm CanGetExactAmount$(n, m, Os[1..n, 1..n])$ som löser (det NP-fullständiga) beslutsproblemet.

   Vi konstruerar en lösning i två steg. Först tar vi reda på vilka rutor i rutnätet som roboten besöker under sin vandring från ruta (1,1) till ruta (n, n). Därefter följer vi robotens vandring uppifrån och ned och noterar hur roboten rört sig.

För att avgöra vilka rutor i rutnätet som roboten besöker under sin vandring prövar vi oss fram till vilka rutor som roboten *inte* behöver besöka. Vi går igenom rutorna i tur och ordning och prövar att sätta rutans osmiuminnehåll till $m+1$. En robot som på sin vandring plockar upp totalt m gram osmium kan aldrig besöka en ruta där det ligger $m+1$ gram osmium. Med CanGetExactAmount kollar vi om det fortfarande finns en lösning med den modifierade osmiummatrisen. Om det inte finns det vet vi att rutan besöks i den vandring vi söker och vi återställer rutans osmiuminnehåll. Men om det fortfarande finns en lösning med den modifierade osmiummatrisen så vet vi att rutan inte behöver besökas och vi behåller den stora osmiummängden i rutan.

När alla rutor i rutnätet gåtts igenom vet vi precis vilka rutor som roboten behöver besöka.

Vi kan nu följa robotens vandring genom dessa rutor.

Om roboten gjort en backning tillbaka till ruta (x, y) kommer det att synas genom att både $(x+1, y)$ och $(x, y+1)$ besöks av roboten. Vilken av dessa rutor som besöks sist kan vi se genom att titta ytterligare ett steg fram: om $(x+2, y)$ besöks så måste roboten gå till $(x, y+1)$ först, backa till (x, y) och sedan gå till $(x+1, y)$ och $(x+2, y)$; om inte $(x+2, y)$ besöks så kan roboten gå till $(x+1, y)$ först, backa till (x, y) och sedan gå till $(x, y+1)$ och vidare till antingen $(x+1, y+1)$ eller $(x, y+2)$. Någon ytterligare möjlighet finns inte.

Om roboten inte gjort en backning tillbaka till ruta (x, y) så är det bara att kika på om ruta $(x+1, y)$ besöks av roboten, för då måste roboten gå nedåt – annars måste den gå åt höger.

```
ConstructExactAmount(n, m, Os[1..n, 1..n])=
    if not CanGetExactAmount(n, m, Os[1..n, 1..n]) then return "" // ingen lösning finns
    for x ← 1 to n do
        for y ← 1 to n do
            oldOsValue ← Os[x, y]
            Os[x, y] ← m + 1
            if not CanGetExactAmount(n, m, Os[1..n, 1..n]) then
                Os[x, y] ← oldOsValue
    assert I en lösning besöker Roboten rutorna i Os som inte har m + 1 gram osmium
    x ← 1; y ← 1
    movements ← ""
    while x < n ∧ y < n do
            inv Lösningsstigen fram till (x, y) beskrivs av movements
            if x < n ∧ y < n ∧ Os[x + 1, y] ≤ m ∧ Os[x, y + 1] ≤ m then
                if x + 2 ≤ n ∧ Os[x + 2, y] ≤ m then
                    x ← x + 1 movements.append(">")
                else
                    y ← y + 1; movements.append("∨")
            else
                if x < n ∧ Os[x + 1, y] ≤ m then
                    x ← x + 1; movements.append("∨")
                else
                    y ← y + 1; movements.append(">")
    assert Lösningsstigen fram till (n, n) beskrivs av movements
    return movements
```

Att konstruktionen av *movement* är korrekt visas enkelt med hjälp av resonemanget ovan tillsammans med assertsatserna och invarianten i whileslingan.

Algoritmen gör $n^2 - 2$ anrop till CanGetExactAmount (för vi behöver inte anropa den för rutorna (1,1) och (n, n) eftersom dessa ingår i varje vandring). Algoritmens övriga arbete är linjärt i indatas storlek, både med enhetskostnad och bitkostnad.