**Algoritmer och Komplexitet ht 09. Övning 7**

**Oavgörbarhet m.m.**

**Oavgörbarhet 1** Finns det något explicit program $P$ så att det givet $y$ är avgörbart huruvida $P$ stannar på indata $y$?

---

**Oavgörbarhet 2** Finns det något explicit program $P$ så att det givet $y$ är oavgörbart huruvida $P$ stannar på indata $y$?

---

**Oavgörbarhet 3** Om programmet $x$ stannar på tomt indata så låter vi $f(x)$ vara antalet steg innan det stannar. Annars sätter vi $f(x) = 0$. Definiera nu $MR(y)$, den maximala körtiden över alla program vars binära kodning är mindre än $y$.

$$MR(y) = \max_{x \leq y} f(x),$$

Är $MR$ beräkningsbar?

---

**Oavgörbarhet 4** Visa att funktionen $MR$ i föregående uppgift växer snabbare än varje rekursiv funktion. Visa närmare bestämt att för varje rekursiv funktion $g$ finns ett $y$ så att $MR(y) > g(y)$.

---

**Oavgörbarhet 5** Anta att en turingmaskin $M$, indata $X$ (som står på bandet från början) och en heltalskonstant $K$ är givna. Är följande problem avgörbart eller oavgörbart?

Stannar $M$ på indata $X$ efter att ha använt högst $K$ rutor på bandet (en använd ruta får skrivas och läsas flera gånger)?

---

**Konstruera kappsäckslösning** Kappsäcksproblemet är ett välkänt NP-fullständigt problem. Indata är en mängd $P$ med prylar med vikt $w_i$ och värde $v_i$, en kappsäcksstorlek $S$ och ett mål $K$. Frågan i kappsäcksproblemet är ifall det går att välja ut prylar av sammanlagt värde minst $K$ så att deras sammanlagda vikt är högst $S$. Alla tal i indata är ickenegativa heltal.

Anta nu att vi har en algoritm $A$ som löser ovanstående beslutsproblem. Konstruera en algoritm som med hjälp av anrop till $A$ löser det konstruktiva kappsäcksproblemet, det vill säga med samma indata som $A$ dels besvarar kappsäcksproblemet och dels, ifall svaret är ja, talar om precis vilka prylar som ska packas ned i kappsäcken. Algoritmen får anropa $A$ $O(|P|)$ gånger men får i övrigt inte ta mer än polynomisk tid.

Du ska alltså konstruera och analysera en turingreduktion av det konstruktiva kappsäcksproblemet till det vanliga kappsäcksproblemet (som är ett beslutsproblem).

---

**Tillförlitlighet hos Internet** Det händer alltför ofta att datorer eller enskilda förbindelser mellan datorer är trasiga. För att detta inte ska störa möjligheten att kommunicera inom resten av Internet vill man att det ska finnas alternativa vägar mellan varje par av datorer i nätet. Om det till exempel finns tre olika vägar genom Internet från datorn A till datorn B och dessa vägar dessutom är helt disjunkta (utom ändpunkterna) så kan två stycken datorer, vilka som helst, försvinna utan att möjligheten att kommunicera mellan A och B försvinner.

I det här problemet tänker vi oss Internet som en oriktad graf där hörnen motsvarar datorerna i Internet och varje kant $(X, Y)$ motsvarar en *möjlig direktförbindelse* mellan datorerna $X$ och $Y$. För varje par av datorer $(X, Y)$ har vi också en önskad tillförlitlighet $T(X, Y)$ som anger hur många disjunkta vägar det ska finnas i Internet mellan $X$ och $Y$. Till sist finns det en budget $B$ som anger hur många direktförbindelser man har råd att konstruera.

Frågan är: går det att plocka ut en delgraf bestående av $B$ kanter så att det för alla par av hörn $(X, Y)$ finns minst $T(X, Y)$ disjunkta stigar i delgrafen mellan $X$ och $Y$.

Visa att detta problem är NP-fullständigt!

---

# Lösningar

**Lösning till Oavgörbarhet 1**
Ja, det finns massor av sådana program. Ta till exempel det enkla programmet $P(y) =$ **return**. Det är lätt att se att $P$ stannar på alla indata. □

---

**Lösning till Oavgörbarhet 2**
Ja. Låt till exempel $P$ vara en interpretator och indata $y$ vara ett program $x$ följt av indata $y'$ till det programmet. Det betyder att $P(y)$ beter sig precis som programmet $x$ skulle göra på indata $y'$, och det är ju oavgörbart huruvida ett program $x$ stannar på ett visst indata $y'$. □

---

**Lösning till Oavgörbarhet 3**
Nej, *MR* är inte beräkningsbar. Vi vet att det är oavgörbart huruvida ett program stannar på blankt indata (tomma strängen). Reducera därför detta problem till *MR*. Notera att $MR(y)$ är det maximala antalet steg som varje program av "storlek" högst $y$ behöver innan det stannar om det startas med blankt indata.

```
StopBlank(y) =
        s ← MR(y)
        if s = 0 then return false
        else
            simulera y på blankt indata i s steg (eller tills y stannar)
            if y stannade then return true
            else return false
```

Om inte $y$ har stannat inom $s$ steg så vet vi att den aldrig stannar. Eftersom StoppBlank inte är beräkningsbar så kan inte *MR* vara det heller. □

---

**Lösning till Oavgörbarhet 4**
Anta motsatsen, dvs att det finns en rekursiv funktion $g$ så att $g(y) \geq MR(y)$ för alla $y$. Då skulle vi kunna byta ut första satsen $s \leftarrow MR(y)$ i programmet i lösningen till föregående uppgift mot satsen $s \leftarrow g(y)$ och programmet skulle ändå fungera. Dessutom skulle programmet då bli

beräkningsbart vilket vi vet att det inte kan vara. Därför har vi en motsägelse och vårt antagande var alltså falskt. □

---

**Lösning till Oavgörbarhet 5**
Problemet är avgörbart. Eftersom turingmaskinen måste hålla sig inom $K$ rutor på bandet så finns det bara ett ändligt antal konfigurationer som turingmaskinen kan befinna sig i. Om maskinen har $m$ tillstånd så är antalet konfigurationer $m \cdot K \cdot 3^K$ (antalet tillstånd gånger antalet möjliga positioner för läs- och skrivhuvudet gånger antalet möjliga bandkonfigurationer). Simulera därför turingmaskinen i $m \cdot K \cdot 3^K + 1$ steg och kontrollera hela tiden att den inte rör sig utanför dom $K$ rutorna på bandet. Om turingmaskinen stannar inom denna tid svarar vi *ja*. Om turingmaskinen inte stannat efter denna tid så måste den ha återkommit till en konfiguration som den tidigare varit i, och därmed är den inne i en oändlig slinga och vi kan tryggt svara *nej*. □

---

**Lösning till Konstruera kappsäckslösning**

```
Kappsäck(P,S,K)=
  if not A(P,S,K) then return "Ingen lösning"
  foreach p in P do
    if A(P-p,S,K) then P:=P-{p}
  return P
```

Algoritmen anropar $A$ högst $|P| + 1$ gånger. Den returnerade mängden P är en lösning för den uppfyller följande två kriterier.

- *P innehåller inga överflödiga element* eftersom forslingan löper över alla element i P, och i varje varv kollas om elementet p är överflödigt; om det är det plockas det bort ur P.
- *P innehåller en lösning* eftersom detta är en invariant för slingan. Om man kommer till slingan är detta uppfyllt, och efter varje varv i slingan är det uppfyllt.

□

---

**Ledning till Tillförlitlighet hos Internet**
Reducera problemet *Hamiltonsk krets*.

---