Föreläsning 11: PSPACE-problem

Minneskomplexitet: Om en algoritm A löser ett problem X genom att använda $O(f(n))$ bitar med minne där n är indatas storlek säger man att $X \in$ SPACE$(f(n))$.

Klassen PSPACE

Def: $X \in$ PSPACE om och endast om $X \in$ SPACE(n^k) för något k.

PSPACE-problem är intressanta bl.a. av följande skäl:

- Det är den första intressanta problemklassen som potentiellt är större än NP.
- Problemen att hitta vinnande strategier i olika spel ligger i PSPACE.

P $\subseteq$ PSPACE

Antag att $X \in$ P och att det finns en Turingmaskin som avgör X i tid $O(n^k)$. Denna algoritm kan högst använda $O(n^k)$ minnesceller. Alltså gäller att $X \in$ P $\Rightarrow X \in$ PSPACE.

En omvänd uppskattning

Antag att $Y \in$ PSPACE och att en Turingmaskin M använder cn^k minnesceller. Om det finns 3 symboler $(0, 1, \#)$ som kan användas så finns det 3^{cn^k} stycken möjliga innehåll på arbetsbandet och cn^k möjliga positioner som arbetshuvudet kan stå på. Ingen kombination plats/innehåll kan upprepas (då skulle maskinen loopa). Det visar att beräkningen måste stanna efter $O(n^k 3^{cn^k})$ steg. Det ger att tidskomplexiteten inte är värre än exponentiell, d.v.s. $Y \in$ EXPTIME.

NP $\subseteq$ PSPACE

Vi vet att 3-SAT är **NP**-fullständigt. Det räcker att visa att 3-SAT $\in$ PSPACE.

Givet ϕ med n variabler går vi igenom alla 2^n möjliga variabeltilldelningar i tur och ordning. Minnesutrymmet som krävs är $\log 2^n = n$ för att hålla räkningen på tilldelning $+k$ extraminne. Det ger $O(n)$ i minnesåtgång.

Olika komplexitetsklasser

Vi har nu klasserna

$$\text{P} \subseteq \text{NP} \subseteq \text{PSPACE} \subseteq \text{EXPTIME}$$

där EXPTIME är de problem som kan lösas i TIME(c^{n^k}) för några tal c, k. Det går att visa att P $\neq$ EXPTIME. Inga andra olikheter som t.ex. P $\neq$ NP eller NP $\neq$ PSPACE är bevisade.

PSPACE-kompletta problem

Ett problem A är PSPACE-komplett om

1. $A \in$ PSPACE
2. Varje problem $B \in$ PSPACE kan reduceras till A, d.v.s. $B \leq_P A$.

Vi kommer snart att se att PSPACE-kompletta problem finns.

Problemet QSAT

En QSAT-formel har formen

$$\exists x_1 \forall x_2 \exists x_3 \ldots \forall x_{n-1} \exists x_n \phi(x_1, \ldots, x_n)$$

där ϕ är på 3-SAT-form.

Möjliga värden på variablerna är $\{0, 1\}$.

$\exists x_1 \forall x_2 \phi(x_1, x_2)$ betyder att det finns värde på x_1 (0 eller 1) så att $\phi(x_1, x_2)$ är sann för alla värden på x_2 (0 och 1).

Vi vill nu avgöra om formler av detta slag är *logiskt giltiga* eller inte.

QSAT:

Indata: En QSAT-formel.

Mål: Avgör om formeln är logiskt giltig eller inte.

Obs: SAT är problemet att avgöra om en formel på formen

$$\exists x_1 \exists x_2 \exists x_3 \ldots \exists x_{n-1} \exists x_n \phi(x_1, \ldots, x_n)$$

är logiskt giltig eller inte.

QSAT $\in$ PSPACE

Använd följande algoritm:

Låt formlerna vi arbetar med ha formen $Q_i x_i Q_{i+1} x_{i+1} \ldots Q_n x_n \phi_i(x_i, \ldots, x_n)$.

QSAT-REK(ϕ)
(1) **if** första kvantifikatorn är $\exists x_i$
(2) **if** QSAT-REK($Q_{i+1} \ldots \phi(0, x_{i+1}, \ldots, x_n)$) = 1 eller QSAT-REK($Q_{i+1} \ldots \phi(1, x_{i+1}, \ldots, x_n)$) = 1
(3) Nollställ allt rekursivt aktivt minne
(4) **return** 1
(5) **if** första kvantifikatorn är $\forall x_i$
(6) **if** QSAT-REK($Q_{i+1} \ldots \phi(0, x_{i+1}, \ldots x_n)$) = 1 och QSAT-REK($Q_{i+1} \ldots \phi(1, x_{i+1}, \ldots x_n)$) = 1
(7) Nollställ allt rekursivt aktivt minne
(8) **return** 1
(9) **if** ϕ inte innehåller någon kvantifikator
(10) Beräkna värdet på ϕ och returnera det

För en formel med k variabler används $p(k)$ minnesceller per variabel. Det ger totalt $p(n) + p(n-1) + \ldots p(1) \leq np(n)$ minnesceller som används. Det visar att QSAT $\in$ PSPACE.

Planeringsproblemet

Vi har en uppsättning *tillståndsvariabler* $c_1, c_2, \ldots, c_n$ som har värde 0 eller 1. Värdena på $c_1, c_2, \ldots, c_n$ bestämmer vilket tillstånd vi befinner oss i. Vi har *operatorer* $O_1, O_2, \ldots O_k$ som ändrar tillståndsvariablerna.

Indata: Listor $c_1, c_2, \ldots, c_n$ och $O_1, O_2, \ldots O_k$. Ett starttillstånd C_0 och ett måltillstånd C^*.
Mål: Finns det en följd $O_{i_1}, O_{i2}, \ldots O_{i_j}$ som transformerar C_0 till C^*?

Savitchs sats

Givet en graf G med n noder och två givna noder a, b så finns det en algoritm som med minneskomplexitet $O((\log n)^2)$ avgör om det finns en väg mellan a och b.

Vi definierar

```
PATH(x, y, L)
(1)   if L = 1 and x = y or (x, y) ∈ E(G)
(2)       return 1
(3)   if L > 1
(4)       Numrera alla noder med räknare med minne
          log n
(5)       foreach z ∈ V(G)
(6)           Beräkna Path(x, z, ⌈L/2⌉). Radera an-
              vänt minne och spara returnerat värde
(7)           Beräkna Path(z, y, ⌈L/2⌉). Radera an-
              vänt minne och spara returnerat värde
(8)           if båda beräkningarna ger värde 1
(9)               return 1
(10)  return 0
```

Kör $Path(a, b, n)$. Svaret 1 visar att det finns väg $a \to b$.

I varje rekursivt steg lagras information x, y, L. Den informationen kräver minne $3 \log n$. Rekursionsdjupet blir högst $\log n$. Det ger minnesåtgång $O((\log n)^2)$.

Planning ∈ PSPACE

Vi använder Savitchs sats. Det finns potentiellt 2^n möjliga tillstånd i ett Planningproblem. Vi vill veta om det finns en väg $C_0 \to C^*$. En sådan väg har längd $\leq 2^n - 1$. Använd algoritmen i Savitchs sats. Den kräver då $O(n)$ i minnesutrymme.

Är NP = PSPACE?

Ingen har lyckats avgöra det. Men flera PSPACE-problem verklar sakna *certifikat*. Som exempel:

- QSAT: Om en QSAT-formel ϕ är logiskt giltig verkar enda sättet att visa det vara att ”gå igenom alla variabeltilldelningar”. De är 2^n stycken. Ett sådant certifikat är exponentiellt i indatas storlek.
- Planning: Ett certifikat skulle vara en väg. Men en sådan väg kan ha längd $2^n - 1$. Den är därför exponentiell i indatas storlek.

NSPACE

En ickedeterministisk algoritm avgör ett språk (problem) L om

- $A(x) =$ Ja med sannolikhet $> 0 \Leftrightarrow x \in L$.
- $A(x) =$ Nej med sannolikhet $1 \Leftrightarrow x \notin L$.

TIME($f(n)$) står för problem som kan avgöras i tid $O(f(n))$ med deterministisk algoritm.

NTIME($f(n)$) står för problem som kan avgöras i tid $O(f(n))$ med ickedeterministisk algoritm.

Det går att visa att $A \in \text{NTIME}(f(n)) \Rightarrow A \in \text{TIME}(c^{f(n)})$

$A \in \text{P} \Leftrightarrow A \in \text{TIME}(n^k)$ för något k.

$A \in \text{NP} \Leftrightarrow A \in \text{NTIME}(n^k)$ för något k

På samma sätt kan vi definiera NPSPACE genom

$A \in \text{NPSPACE} \Leftrightarrow A \in \text{NSPACE}(n^k)$ för något k

PSPACE = NPSPACE

Bevisskiss:

Låt X vara problem i NPSPACE. Låt M vara en ickedeterministisk Turingmaskin som avgör X i minne $O(n^k)$. Beräkningsgrafen innehåller maximalt $O(c^{n^k})$ noder.

Algoritmen i Savitchs sats hittar accepterande beräkning (om den finns) med användande av minne av storlek $O((\log c^{n^k})^2) = O(n^{2k})$.

Så $X \in$ PSPACE.

QSAT är PSPACE-komplett

Bevisskiss:
Låt $A \in$ PSPACE. Vi vill göra en reduktion $A \leq_P$ QSAT.

Det finns en Turingmaskin M som avgör A med polynomiell minnesåtgång. För att avgöra A letar vi efter accepterande vägar i beräkningsgrafen. Som tidigare gäller att en sådan väg har längd $O(c^{n^k})$.

Vi översätter problemet att hitta vägar till ett logikproblem genom att definierar formler $\psi_{v_1,v_2,L}$ som anger att det finns en väg från v_1 till v_2 av längd $\leq L$.

Formeln definieras rekursivt ur formler av typen $\psi_{v_1,v_3,\lceil \frac{L}{2} \rceil}$ och $\psi_{v_3,v_2,\lceil \frac{L}{2} \rceil}$.

Svårigheten är att definiera en rekursion som inte gör att formlerna blir exponentiellt stora. Det kan dock ordnas genom ett listigt användande av kvantifikatorer. Resultatet blir en formel ψ på QSAT-form som är logiskt giltig om och endast om det finns accepterande väg i beräkningsgrafen.

Spelet (Generaliserad) GEOGRAFI

Låt G vara en riktad graf med en startnod v.

Vi har två spelare I och II.

I börjar spelet. Spelarna turas om att göra *drag*.

Tillåtna drag är att flytta från en nod x till en grannod y *som inte har besökts tidigare*.

Den spelare som först inte kan göra ett drag förlorar.

Indata: En graf G och en startnod v.

Mål: Finns det en vinnande strategi för spelare I?

GEOGRAFI $\in$ PSPACE

Vi ger en skiss på en algoritm som avgör om I har en vinnande strategi.

Givet startkonfigurationen $< G, v >$ låter vi G_1 vara G med v och kanter tillfrån v borttagna.

Låt $v_1, v_2, \ldots, v_k$ vara v:s grannar i G.

Undersök $< G_1, v_1 >, < G_1, v_2 >, \cdots < G_1, v_k >$ rekursivt. Om någon av dessa problem saknar vinnande strategi (för II) så returnerar vi ”Ja”, annars returnerar vi ”Nej”.

Det går lätt att se att denna algoritm kan implementeras så att den bara använder polynomiellt stort minnesutrymme.

GEOGRAFI är PSPACE-komplett

Vi vet att GEOGRAFI $\in$ PSPACE.

Det går att göra en reduktion QSAT $\leq_P$ GEOGRAFI.