PSPACE Problems

Space Complexity: If an algorithm A solves a problem X by using $O(f(n))$ bits of memory where n is the size of the input we say that $X \in$ SPACE($f(n)$).

The Class PSPACE

Def: $X \in$ PSPACE if and only if $X \in$ SPACE(n^k) for some k.

PSPACE Problems are interesting since:

- They form the first interesting class potentially greater than NP.
- The problem of finding winning strategies is in PSPACE.

P $\subseteq$ PSPACE

Assume $X \in$ P and there is a Turing Machine that decides X in time $O(n^k)$. This algorithm can use at most $O(n^k)$ bits of memory. So we get $X \in$ P $\Rightarrow X \in$ PSPACE.

In the other direction

Assume $Y \in$ PSPACE and that a Turing Machine M uses cn^k bits of memory. If we have 3 possible symbols $(0, 1, \#)$ on the input tape there are 3^{cn^k} possible contents on the tape and cn^k possible positions for the head. No possible combination of content/position can be repeated. (Since the machine then would be looping.) This shows that the machine must stop after at most $O(n^k 3^{cn^k})$ steps. So the time complexity cannot be worse than exponential, i.e. $Y \in$ EXPTIME.

NP $\subseteq$ PSPACE

We know that 3-SAT is NP-Complete. So we just have to show that 3-SAT $\in$ PSPACE.

Given ϕ with n variables we run true all 2^n possible value assignments one at a time. The amount of space needed is $\log 2^n = n$ to keep count of the number of the assignment and $+k$ extra bits of memory.. This gives us space complexity $O(n)$.

Different Complexity Classes

We now have the classes

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

where EXPTIME is the class of problems that can be decided in TIME(c^{n^k}) for some numbers c, k. It is possible to show that P $\neq$ EXPTIME. No other inequalities are known. This means that no inequalities like P $\neq$ NP eller NP $\neq$ PSPACE are known to be true.

PSPACE Complete Problems

A problem is PSPACE-Complete if

1. $A \in$ PSPACE
2. Every problem $B \in$ PSPACE can be reduced to A, i.e. $B \leq_P A$.

The problem QSAT

A QSAT-formula is of the form

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

where ϕ is in 3-SAT-form.

possible values for the variables are $\{0, 1\}$.

$\exists x_1 \forall x_2 \phi(x_1, x_2)$ means that there is a value for x_1 (0 or 1) such that $\phi(x_1, x_2)$ Is true for all values for x_2 (0 och 1).

We want to decide if a formula of this kind are *valid* or not.

QSAT:

Input: A QSAT-formula

Goal: Decide if the formula is valid or not.

Obs: SAT Is equivalent to the problem of deciding if a formula

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

is valid or not.

QSAT $\in$ PSPACE

Let the formulas we use be written $Q_i x_i Q_{i+1} x_{i+1} \dots Q_n x_n \phi_i(x_i, \dots, x_n)$.

QSAT(ϕ)=
if The first quantifier is $\exists x_i$
  if QSAT $(Q_{i+1} \dots \phi(0, x_{i+1}, \dots, x_n)) = 1$
    or
   QSAT $(Q_{i+1} \dots \phi(1, x_{i+1}, \dots, x_n)) = 1$
Erase all recursively active memory
Return 1
if The first quantifier is $\forall x_i$
   if QSAT $(Q_{i+1} \dots \phi(0, x_{i+1}, \dots x_n)) = 1$
     and
   QSAT $(Q_{i+1} \dots \phi(1, x_{i+1}, \dots x_n)) = 1$
Erase all recursively active memory
Return 1
if ϕ does not contain any quantifier Compute the value of ϕ and return it

When we have a formula with k variables we use $p(k)$ (polynomial) bits of memory for each variable. This shows that $p(n) + p(n-1) + \dots p(1) \leq np(n)$ bits of memory are used and this shows that QSAT $\in$ PSPACE.

The Planning Problem

We have a set of *state variables* $c_1, c_2, \ldots, c_n$ with values 0 or 1. The values of $c_1, c_2, \ldots, c_n$ tells us what *state* we are in. We have *operators* $O_1, O_2, \ldots O_k$ which changes the state variables. The problem is:

Input : Lists $c_1, c_2, \ldots, c_n$ and $O_1, O_2, \ldots O_k$. A start state C_0 and a goal state C^*.

Goal: Is there a sequence $O_{i_1}, O_{i2}, \ldots O_{i_j}$ that transforms C_0 to C^*?

Savitch' Theorem

Given a graph G with n vertices and two vertices a, b there is an algorithm with space complexity $O((\log n)^2)$ which decides if there is a path between a and b or not.

We define

Path(x, y, L)

(1) **if** $L = 1$ and $x = y$ or $(x, y) \in E(G)$

(2) **return** 1

(3) **if** $L > 1$

(4) Enumerate all vertices with a counter using $\log n$ bits of memory

(5) **foreach** $z \in V(G)$

(6) Compute $Path(x, z, \lceil \frac{L}{2} \rceil)$. Erase used memory and return value

(7) Compute $Path(z, y, \lceil \frac{L}{2} \rceil)$. Erase used memory and return value

(8) save all returned values

(9) **if** both computations returns 1

(10) **return** 1

(11) **return** 0

Compute $Path(a, b, n)$. If the answer is 1 we know that there is a path $a \to b$.

In each recursive step we store the information x, y, L. That takes $3 \log n$ bits of memory. The recursion depth is at most $\log n$. The space complexity is $O((\log n)^2)$.

Planning $\in$ PSPACE

We use Savitch's Theorem. There can be at most 2^n different states in Planning. We want to know if there is a path $C_0 \to C^*$. Such a path has length $\leq 2^n - 1$. Use the algorithm in Savitch's Theorem. It uses $O(n^2)$ bits of memory.

NSPACE

A non-deterministic algorithm decides a language L if

- $A(x) =$ Yes with probability $> 0 \Leftrightarrow x \in L$.
- $A(x) =$ No with probability $1 \Leftrightarrow x \notin L$.

TIME($f(n)$) is the class of problems which can be decided in time $O(f(n))$ by a deterministic algorithm.

NTIME($f(n)$) is the class of problems which can be decided in time $O(f(n))$ by a non-deterministic algorithm.

It is possible to show that $A \in$ NTIME($f(n)$) $\Rightarrow$ $A \in$ TIME($c^{f(n)}$)

$A \in \mathsf{P} \Leftrightarrow A \in \mathsf{TIME}(n^k)$ for some k.

$A \in \mathsf{NP} \Leftrightarrow A \in \mathsf{NTIME}(n^k)$ for some k

In the same way we can define NPSPACE by

$A \in \mathsf{NPSPACE} \Leftrightarrow A \in \mathsf{NSPACE}(n^k)$ for some k

PSPACE = NPSPACE

Sketch proof:

Let X be a problem in NPSPACE. Let M be a non-deterministic Turing Machine which decides X and uses $O(n^k)$ bits of memory. The computation graph contains at most $O(c^{n^k})$ vertices.

The algorithm in Savitch's Theorem finds an accepting computation in the computation graph (if there is one) and uses at most $O((\log c^{n^k})^2) = O(n^{2k})$.

So we get $X \in$ PSPACE.

The game (GENERALIZED) GEOGRAPHY

Let G be a directed graph with a start vertex v.

Let us assume that we have two players I and II.

I makes the first move. Then the players take turns and make moves.

The moves allowed are moves from a vertex x to an adjacent vertex y *which has not been visited before*.

The first player that cannot move loses the game.

Input: A graph G and a start vertex v.

Goal: Is there a winning strategy for player I?

GEOGRAFI $\in$ PSPACE

We will look at a sketch of an algorithm which decides if there is a winning strategy for the first player in GEOGRAPH.

Given the start configuration $< G, v >$ we let G_1 be G with v and all edges going from v removed.

Compute $Path(a, b, n)$. If the answer is 1 we know that there is a path $a \to b$.

In each recursive step we store the information x, y, L. That takes $3 \log n$ bits of memory. The recursion depth is at most $\log n$. The space complexity is $O((\log n)^2)$.

Planning $\in$ PSPACE

We use Savitch's Theorem. There can be at most 2^n different states in Planning. We want to know if there is a path $C_0 \to C^*$. Such a path has length $\leq 2^n - 1$. Use the algorithm in Savitch's Theorem. It uses $O(n^2)$ bits of memory.