Algorithms & Complexity

Space Complexity

Nicolas Stroppa
Patrik Lambert - plambert@computing.dcu.ie

CA313@Dublin City University. 2008-2009.

December 4, 2008
Hierarchy of problems

- NP
- NP-Comp
- P
- NP
- NP-Comp
- P
**NP-intermediate Languages**

If $P \neq NP$, then are there languages which neither in $P$ or $NP$ — complete?

The answer is yes. These languages are called $NP$-intermediate languages, and their class is called $NPI$. There are real problems in $NP$ for which neither polynomial-time algorithms nor $NP$-completeness proofs have been found. It makes sense to assume that these belong to $NPI$, until we know differently.

Examples of such problems are:

- Graph isomorphism
- Composite number (Given $N \in \mathbb{Z}+$, find $m, n \in \mathbb{Z}+$ such that $mn = N$)
- Linear programming was considered $NP$-intermediate but in 1979 it was proven to be in $P$
Co-NP Problems

Definition (Complement)
The \textit{complement} of a decision problem $X$, denoted $\overline{X}$, is the decision problem in which the “Yes” instances of $\overline{X}$ are the “No” instances of $X$ and vice versa.

Examples

- Consider the problem: “is a number a prime number?”. Its complement is to determine whether a number is a composite number (a number which is not prime).
- Consider the subset sum problem: Given a set of integers $S = \{i_1, i_2, \ldots, i_n\}$, does any (non-empty) subset $A \subseteq S$ sum to 0? Its complement asks: Given a finite set of integers does every non-empty subset have a nonzero sum?
**Co-NP Problems**

**Definition (Co-NP)**

Co-NP is the class of complements of NP problems.

**Example**

- The subset sum problem is in NP (actually it is NP-complete) thus its complement problem is in co-NP.
Hierarchy of problems

Assuming $P \neq NP$ and $NP \neq co-NP$: 

\[ \text{NP} \cap \text{co-NP} \]

\[ \text{NPC} \cap \text{P} \cap \text{co-NPC} \]
Space complexity

Two main characteristics for programs

▶ Time complexity: $\sim$ CPU usage
▶ Space complexity: $\sim$ RAM usage
Space complexity: an informal definition

**Definition (Space complexity)**

The *space complexity* of a program (for a given input) is the number of elementary objects that this program needs to store during its execution. This number is computed with respect to the size \( n \) of the input data.
Space complexity: a formal definition

Definition (Space complexity)
For an algorithm $T$ and an input $x$, $DSPACE(T, x)$ denotes the number of cells used during the (deterministic) computation $T(x)$. We will note $DSPACE(T) = O(f(n))$ if $DSPACE(T, x) = O(f(n))$ with $n = |x|$ (length of $x$).
Note: $DSPACE(T)$ is undefined whenever $T(x)$ does not halt.
Space complexity: example 1

// note: x is an unsorted array
int findMin(int[] x) {
    int k = 0; int n = x.length;
    for (int i = 1; i < n; i++) {
        if (x[i] < x[k]) {
            k = i;
        }
    }
    return k;
}
Space complexity: example 1

// note: x is an unsorted array
int findMin(int[] x) {
    int k = 0; int n = x.length;
    for (int i = 1; i < n; i++) {
        if (x[i] < x[k]) {
            k = i;
        }
    }
    return k;
}

\[
T(\text{findMin}, n) = n + 2
\]

\[
T(\text{findMin}, n) = O(n)
\]
Space complexity: example 2

// note: x is an unsorted array
void multVect(int[] x, int[][][] a) {
    int k = 0; int n = x.length;
    for (int i = 1; i < n; i++) {
        for (int j = 1; j < n; j++) {
            a[i][j] = x[i] * x[j]
        }
    }
}
Space complexity: example 2

// note: x is an unsorted array
void multVect(int[] x, int[][][] a) {
    int k = 0; int n = x.length;
    for (int i = 1; i < n; i++) {
        for (int j = 1; j < n; j++) {
            a[i][j] = x[i] * x[j]
        }
    }
}

\[ T(multVect, n) = n \times n + n + 2 \]
\[ T(multVect, n) = O(n^2) \]
Polynomial space complexity

The class $PSPACE$ is defined as:

$$PSPACE = \bigcup_{k \in \mathbb{N}} DSPACE(n^k).$$

$PSPACE$ is the (complexity) class of decision problems that can be solved using a deterministic Turing Machine and a polynomial amount of space (⇒ polynomial space complexity).
Non-Deterministic \textit{SPACE}

For nondeterministic Turing machine the space complexity is denoted \textit{NSPACE}. \textit{NSPACE}(T, x) denotes the number of cells used by the \textit{non-deterministic} computation $T(x)$. We will note $\textit{NSPACE}(T) = O(f(n))$ if $\textit{SPACE}(T, x) = O(f(n))$ with $n = \lvert x \rvert$ (length of $x$).
Space complexity: some results

Savitch’s theorem:
For a function $f(n) \geq \log(n)$:

$$NSPACE(f(n)) \subseteq DSPACE((f(n))^2).$$

Corollary:

$$PSPACE = NPSPACE$$

This follows directly from the fact that the square of a polynomial function is still a polynomial function.

The set of problems with polynomial space complexity is the same if we consider deterministic or non-deterministic Turing machines...
Relations between time and space

\[ P \subseteq NP \subseteq \text{PSPACE} \]
\[ P \subseteq \text{co-NP} \subseteq \text{PSPACE} \]

Note: \( P \) is clearly a subset of \( \text{PSPACE} \), because you cannot use more than polynomial space in only polynomial time.
Relations between time and space

\[ P \subseteq NP \subseteq \text{PSPACE} \]

\[ P \subseteq \text{co-NP} \subseteq \text{PSPACE} \]