CS760 Topics in Computational Complexity **2024-25 Sem I**

Lecture 8: Space Complexity (27-08-2024)

Scribe: Sai Bhanu Teja, Anumalasetty Varshith *Lecturer: Rohit Gurjar*

Just like for time where we had **DTIME**$(T(n))$, for space we can define a complexity class **SPACE**$(S(n))$, as the set of problems for which there is a TM with working space $O(S(n))$ at any instant of execution, where $S : \mathbb{N} \to \mathbb{N}$ is a function. Below here we define working space of a TM

1 Space requirements in TMs

The Turing Machines is considered to have 3 Tapes, namely **Input tape** (Read only), **Working Tape** (Read and Write) and **Output Tape** (Write only).

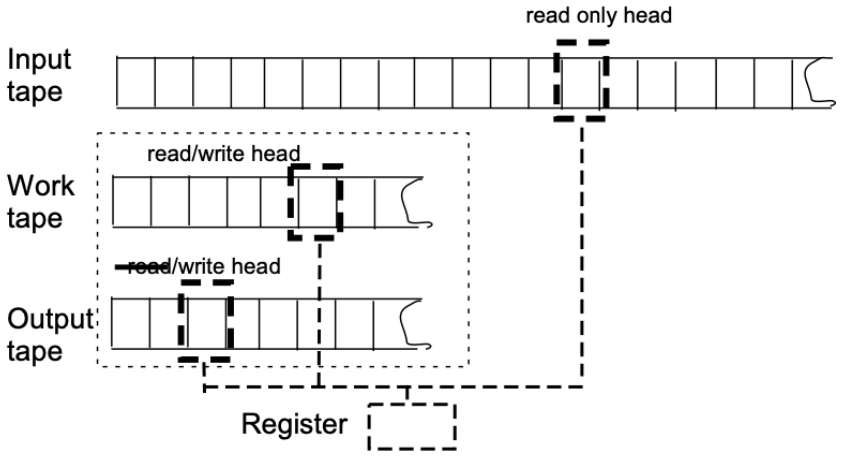


Figure 1: Space bounded computation
(Image Credits: *Computational Complexity: A Modern Approach* by Sanjeev Arora and Boaz Barak)

- Note that in some programs, Working tape takes significantly less space compared to input tapes and also output tape, some others might have working tape in polynomial of input tape length and output is arbitrary.
- Only the length (i.e., no of cells) of working tape are taken into consideration for calculating Space Complexity $S(n)$ of a problem.
- For decision problems, the output is only 0/1, so we don't need a separate output tape.

2 SPACE($S(n)$) and NSPACE($S(n)$) complexity classes

Theorem 8.1.

$$\textbf{DTIME}(S(n)) \subseteq \textbf{SPACE}(S(n))$$

Proof: In the context of Deterministic Turing Machines (DTMs), if the Turing Machines operate within a Time complexity of $T(n)$ then the space complexity $S(n)$ is at most $T(n)$ i.e., A problem solved in $S(n)$ time must be solved in $S(n)$ space. This is because, a machine running in time $T(n)$ can only access that many cells of the tape.

In particular, $\mathbf{P} \subseteq \mathbf{PSPACE}$ (i.e., if you run only polytime, you can take only poly space)

Definition 8.2. **NSPACE**$(S(n))$: *Similar to* **SPACE**$(S(n))$*, but Non-Deterministic TM instead.*

Theorem 8.3.

$$\mathbf{NSPACE}(S(n)) \subseteq \mathbf{DTIME}(2^{O(S(n))})$$

Proof: To prove this, we use the notion of a *configuration graph* of a TM. A configuration of a TM consists of 1) Content on tape 2) State 3) Head position. For a TM with space-bound $S(n)$, the length of tape in TM is $S(n)$, so the number of possibilities for the contents of the tape is $2^{S(n)}$, number of possible head positions on the work tape is $S(n)$, and number of head positions on the input tape is n. So, the total number of possible Turing Machine configurations is $n \times S(n) \times 2^{S(n)}$. Note that we will consider $n \times S(n) \times 2^{S(n)}$ as $2^{O(S(n))}$ because, in general, we assume $S(n) = \Omega(\log n)$.

The configuration graph of M on input x, denoted $G_{M,x}$ is a directed graph with various configurations as nodes and there is an edge from C$'$ and C$''$ iff C$''$ is among the next possible configurations after C$'$. After constructing $G_{M,x}$ in $2^{O(S(n))}$-time, check whether C_{start} is connected to C_{accept} in $G_{M,x}$ using the standard (linear in the number of edges of the graph) BFS algorithm.

So combining both the above results

$$\mathbf{DTIME}(S(n)) \subseteq \mathbf{SPACE}(S(n)) \subseteq \mathbf{NSPACE}(S(n)) \subseteq \mathbf{DTIME}(2^{O(S(n))})$$

3 Some space complexity classes

Definition 8.4.

$$\mathbf{PSPACE} = \cup_{c>0}\mathbf{SPACE}(n^c)$$

$$\mathbf{NPSPACE} = \cup_{c>0}\mathbf{NSPACE}(n^c)$$

$$\mathbf{L} = \mathbf{SPACE}(\log n)$$

$$\mathbf{NL} = \mathbf{NSPACE}(\log n)$$

Interestingly, many problems can be solved with space $S(n) < n$, but this is not the case with time. $T(n) < n$, doesn't make sense as we need at least n time to read input.

Can we prove **NP** $\subseteq$ **PSPACE**?

Yes, We know that any problem in **NP** reduces to 3SAT. If we prove 3SAT $\in$ **PSPACE**, then we are done. We can solve 3SAT by iterating over all possible assignments which takes $O(|\text{input}|)$ space (like all 0's to all 1's i.e., 0 to $2^n - 1$). Hence **NP** $\subseteq$ **PSPACE**.

Theorem 8.5.

$$\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{NPSPACE} \subseteq \mathbf{EXP}$$

Analogous to time hierarchy theorem, there is a space hierarchy theorem. By these hierarchy theorems, we definitely know that **L** $\neq$ **PSPACE** and **P** $\neq$ **EXP**. We don't know about equality or strict subsets for others (for now).

3.1 Some examples of problems in L

Some straightforward examples are

- Finding the index of minimum value in an array
- Verifying if $(a + b)$ is equal to c

Now let us see some Non-obvious examples. Both the problems are Homework. A suggestion is to first try to prove composition and later try multiplication.

- Verifying if $(a \times b)$ is equal to c

- Let $f, g : \mathbb{N} \to \mathbb{N}$ be two functions such that (computing f) $\in \mathbf{L}$ and (computing g) $\in \mathbf{L}$, then computing the composition $f(g(x)) \in \mathbf{L}$

It seems difficult to conceive of any complicated computations use only $O(\log n)$ space. Nevertheless, we cannot currently even rule out that 3SAT $\in \mathbf{L}$. For example, the path problems below.

1. In 2005, Reingold showed that the undirected graph st-connectivity problem described below is in $\mathbf{L}$ without using random walks

$$\mathbf{USTCON} = \{\langle G, s, t\rangle : G \text{ is an undirected graph in which there is a path from } s \text{ to } t\}$$

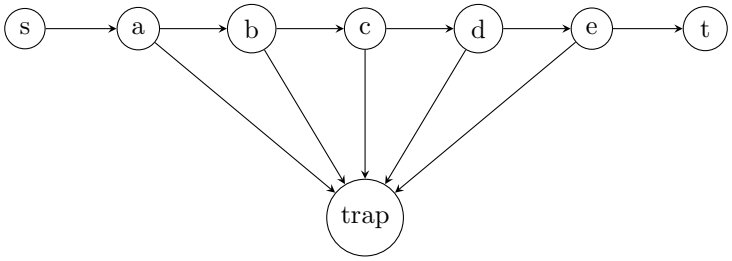
2. Similar problem in case of directed graph is showed to be in **NL**

$$\mathbf{DSTCON} = \{\langle G, s, t\rangle : G \text{ is an a directed graph in which there is a path from } s \text{ to } t\}$$

The reason is that a non-deterministic machine can take a "non-deterministic walk" (Not random walk). With starting at s, non-deterministically selecting a neighbour to go to the next node. The machine accepts iff the walk ends at t in at most n steps, where n is the number of nodes. If the non-deterministic walk has run for n steps already and t has not been encountered, the machine rejects. The space complexity is $O(\log n)$ because we only need to remember the index of the current node.

Is **DSTCON** $\in$ L? Open question.

Why don't random walks work in case of directed graph? Because they do not ensure high probability. For example,



At any position a or b etc, we have half the probability of taking right direction. But once we take wrong step, we can't go back again (not the case with undirected). So the probability of taking required path in random walk is very low.

Also similar to $\mathbf{P} = \mathbf{NP}$?, we have $\mathbf{L} = \mathbf{NL}$? as an open question. If someone wishes to solve former, it is suggested to start with latter one.

3.2 Some examples of problems in PSPACE

What are some interesting problems in **PSPACE**? "Interesting" means problems with polynomial time algorithm is not known but can be solved in polynomial space. Some obvious examples are SAT $\in$ **PSPACE** and UNSAT $\in$ **PSPACE**. Some other examples are

- Quantified Boolean Formula (QBF)

  Is there a satisfying assignment for this QBF $\psi = \exists x_1\ \forall x_2\ \exists x_3\ \forall x_4\ \ldots\ Qx_n\ \phi(x_1, x_2, \ldots, x_n)$

To show that this problem in **PSPACE**, we can write a recursive solution $x_1 = 0$ and check for subproblem recursively. If done, then good. If not assign $x_1 = 1$ and continue. This way it takes $O(|\text{input}|)$ space

- Combinatorial games (chess)

We have 8×8 standard chess board. But theoretical people work on a generalised chess with $n \times n$ size. If we observe, Asking if there is a winning strategy for player 1 is like QBF i.e., Is there a move for player 1 such that for all moves of player 2 (and so on) player 1 can win. Similarly, it was shown that all Two-player games with perfect information (no randomness) and polynomial stopping time were shown to be in **PSPACE**. Next ones are other examples of this.

- Chocolate poison game (Chomp)

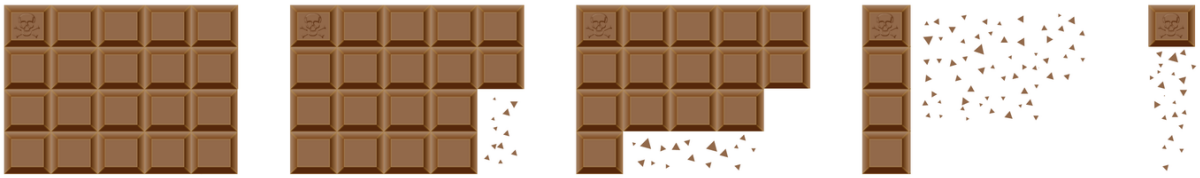


Figure 2: Chocolate poison game
(Image Credits: Chomp - Wikipedia)

The players take it in turns to choose one block and "eat it" (remove from the board), together with those that are below it and to its right. The top left block is "poisoned" and the player who eats this loses

- POSET games

In such games, two players start with a poset (a partially ordered set) and take turns choosing one point in the poset, removing it and all points that are greater. The player who is left with no point to choose, loses. (This is infact generalization of above Chomp game).

3.3 Towards PSPACE-completeness

We don't know if **P** and **PSPACE** are equal or not. We even don't know is POSET game is in **P** or not. (If we would like to prove **P** = **PSPACE**, it might be a good start to prove POSET game is in **P**).

To compare **NP** and **PSPACE**, we don't have any proof for not being equal. But we can see puzzles are like **NP** and games are like **PSPACE**. Proving **NP** and **PSPACE** are equal is like saying winning games is like solving puzzles. It is also stated in the book *Games, Puzzles, and Computation* by Robert A. Hearn and Erik D. Demaine as

"The problem of solving a general Sudoku puzzle is **NP-complete** and Determining the outcome of a generalized Chess game played on an $n \times n$ board is **PSPACE-complete**".

Since complete problems can help capture the essence of a complexity class, we'll next define a class of problems called **PSPACE-complete**. In fact POSET game is a **PSPACE-complete** problem.