Space Complexity

So far most of our theoretical investigation on the performances of the various algorithms considered has focused on "time".

Another important dynamic complexity measure that can be associated with the execution of a program is "space".

This week we will look at a number of complexity theoretic results concerning this other measure.

Our treatment will be fairly general and rather abstract so we will abandon the world of "real life" algorithms and we will go back to counting steps and cell usage in Turing machines (think of it as looking at algorithms written in machine code).

Back to basics

What is the minimum hardware needed to solve any computational problem?

1. In COMP101, they taught us that we need a PC, with JAVA on it, and we need to write a program, debug it and run it!
2. In more theoretical courses (COMP108, COMP202) they told us about pseudo-code: all one really need is plenty of paper, and some well-specified set of instructions in a semi-formal pseudo-code.
3. Pseudo-code algorithms worked on integers, arrays, strings, perhaps graphs.
4. If we really want to measure precisely the resource requirements of such algorithms in a way that is independent from the particular PC we may want to use to run the eventual program we need to simplify our model even further.

Claim. All we need is:

1. the ability to use our senses and move our hands
2. an ordered sequence of boxes, each of which (at a given time) contains either nothing or some object;
3. the ability to see, remove, or replace the object contained in any given box at a given time;
4. one sheet of paper containing a set of rules that always tells us what to do with the content of the particular box we are looking at at a given time, and which box to look at next.

This is a very informal definition of what a Turing machine is. Now we can be more formal.

Turing machines

A (deterministic) *Turing machine* (TM) is defined by

$$T = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

where

Q is a finite set of *states*

Γ is a finite set of allowed *tape symbols*.

B is a symbol of Γ, the *blank* symbol.

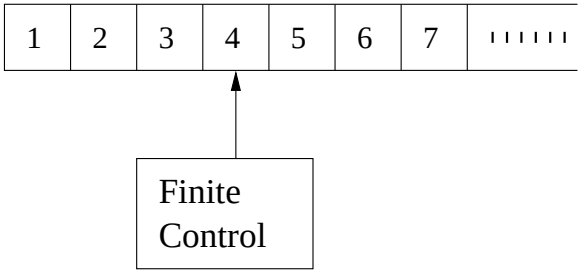
Σ is a subset of Γ not including B of *input symbols*.

δ is the *transition function*, a (partial) mapping from $Q \times \Gamma$ to $Q \times \Gamma \times \{L, R\}$.

q_0 is the *initial* (or *start*) state.

F is a set of states, the *final states*.

A TM can be pictured as a little box moving along a tape (or a string, or a sequence of boxes to go back to the introductory example) changing symbols on the tape according to the values of its transition function



Turing machine computations

A *configuration* of the TM T is a string $\alpha_1 q \alpha_2$ where $\alpha_1\alpha_2 \in \Gamma^*$ and $q \in Q$. It is meant to represent the full state of the machine at a given time instant.

A *move* of T rearranges the symbols of a configuration according to the function δ. So, for instance, if the configuration is

$$X_1 \dots X_{i-1} q X_i \dots X_h$$

and $\delta(q, X_i) = (p, Y, R)$ then the new configuration which describes the global state of T after the move has occurred is (assuming $h \geq i+1$)

$$X_1 \dots X_{i-1} Y p X_{i+1} \dots X_h$$

A *computation* of T is an ordered sequence of configurations such that each of them can be obtained from its predecessor by way of a single move.

The *language* $L(T)$ *accepted by the TM* T is the set of all strings $x \in \Sigma^*$ for which there is a computation of T starting with the configuration $q_0 x$ and ending with a configuration containing a final state.

For any $x \in \mathbb{N}$, let $\langle x \rangle \in \{0,1\}^*$ be the binary representation of x.

A numerical function $f : \mathbb{N} \to \mathbb{N}$ is *computed by* a TM T if for any $x \in \mathbb{N}$ there exists a computation of T which starts with the configuration

$$q_0 \langle x \rangle B \dots$$

and ends with the configuration

$$\langle x \rangle B \langle f(x) \rangle B q_f \dots$$

(where q_f is a final state of T).

Warming-up example

Define a TM accepting the language $L = \{0^n 1^n : n \geq 1\}$.

Let $Q = \{q_0, q_1, q_2, q_3, q_4\}$, $\Sigma = \{0,1\}$, $\Gamma = \Sigma \cup \{X, Y, B\}$ and $F = \{q_4\}$. The transition function is defined as follows:

State	0	1	X	Y	B
q_0	(q_1, X, R)			(q_3, Y, R)	
q_1	$(q_1, 0, R)$	(q_2, Y, L)		(q_1, Y, R)	
q_2	$(q_2, 0, L)$		(q_0, X, R)	(q_2, Y, L)	
q_3				(q_3, Y, R)	(q_4, B, R)
q_4					

Explanation

Often such definitions are very dry. Use them always having in mind some informal description of the meaning of each state. In the particular example:

q_0 Start of a session state: mark with an X the cell you are at if it contains a "0". The behaviour of the machine when in this state is NOT specified in any other case, except when it finds a Y (we will go back to this in the example)

q_1 Look for the first "1", skipping any "0" or Y. Replace "1" by Y and move to state q_2.

q_2 Move back looking for the rightmost X, skipping Y's and "0". As soon as an X is found the machine moves right and goes back in state q_0.

q_3 This state is entered if the machine has no more "0" left to match. In such state the machine will skip any Y and look for the end of the input string (marked by the leftmost "B").

q_4 This is the final state so nothing happens in this state.

q_0	0	0	0	1	1	1	B
X	q_1	0	0	1	1	1	B
X	0	q_1	0	1	1	1	B
X	0	0	q_1	1	1	1	B
X	0	q_2	0	Y	1	1	B
X	q_2	0	0	Y	1	1	B
q_2	X	0	0	Y	1	1	B
X	q_0	0	0	Y	1	1	B
X	X	q_1	0	Y	1	1	B
X	X	0	q_1	Y	1	1	B
X	X	0	Y	q_1	1	1	B
X	X	0	q_2	Y	Y	1	B
X	X	q_2	0	Y	Y	1	B
X	q_2	X	0	Y	Y	1	B
X	X	q_0	0	Y	Y	1	B
X	X	X	q_1	Y	Y	1	B
X	X	X	Y	q_1	Y	1	B
X	X	X	Y	Y	q_1	1	B
X	X	X	Y	q_2	Y	Y	B
X	X	X	q_2	Y	Y	Y	B
X	X	q_2	X	Y	Y	Y	B
X	X	X	q_0	Y	Y	Y	B
X	X	X	Y	q_3	Y	Y	B
X	X	X	Y	Y	q_3	Y	B
X	X	X	Y	Y	Y	q_3	B
X	X	X	Y	Y	Y	B	q_4

Example

Assume the input is the string 000111, the resulting computation is listed on the next page.

Exercises

1. Modify the TM above so that any accepting computation ends with the configuration $0^n1^nBq_4$.
2. Simulate the same machine on the input 011.
3. Define a TM that computes the identity function (i.e. the function that satisfies $f(x) = x$ for any $x \in \mathbb{N}$).
4. Define a TM that computes the successor of a binary number (e.g. 100111 becomes 101000).

Solution to Exercise 3

The set Σ clearly contains only "0" and "1". Let q_0 be the initial state of T. We will define Q and Γ as we go along. Now we define δ.

What do we want the machine to do? Following the definition given few pages back, the machine should start with the configuration $q_0\langle x\rangle B\ldots$ and end with the configuration $\langle x\rangle B\langle f(x)\rangle Bq_f\ldots$ (where $\langle x\rangle$ is the binary for the decimal number x, and q_f is a final state of T).

The transition function can then be defined drafted as follows:

> The machine will move in "sessions" again. During each session it will mark the next available bit, skip everything on the tape until the first unused tape cell, copy the marked bit there and go back looking for a new bit to mark.

A more precise definition is as follows:

State	0	1	Z	U	B
q_0	(q_{10}, Z, R)	(q_{11}, U, R)			(q_3, B, R)
q_{10}	$(q_{10}, 0, R)$	$(q_{10}, 1, R)$			(q_{10s}, B, R)
q_{10s}	$(q_{10s}, 0, R)$	$(q_{10s}, 1, R)$			$(q_2, 0, L)$
q_{11}	$(q_{11}, 0, R)$	$(q_{11}, 1, R)$			(q_{11s}, B, R)
q_{11s}	$(q_{11s}, 0, R)$	$(q_{11s}, 1, R)$			$(q_2, 1, L)$
q_2	$(q_2, 0, L)$	$(q_2, 1, L)$			(q_{2s}, B, L)
q_{2s}	$(q_{2s}, 0, L)$	$(q_{2s}, 1, L)$	$(q_0, 0, R)$	$(q_0, 1, R)$	
q_3	$(q_3, 0, R)$	$(q_3, 1, R)$			(q_4, B, R)
q_4					

Explanation

After reading the next "0"/"1" symbol, the machine "memorises" it in the sense that it moves to a different state (q_{10} or q_{11}) depending on the digit that it just read.

In each of the states of the form q_{1*} the machine is just moving towards the right hand side, skipping anything on its way.

Once the first unused cell has been reached it is changed to the digit that is stored in machine state and the machine moves to state q_2.

In q_2 (and q_{2s}) the machine is going back to the symbol that was last marked. It replaces it with the original digits and starts everything all over again.

More Exercises

1. Simulate the TM just defined on input 0, 10, and 1101.
2. Complete the definition of the TM defined above (what is Q? what is Γ? ...)
3. Define a TM that computes the constant function $f(x) = 5$.
4. (Worth 25% of the CA mark) A TM may compute functions with two or more variables. For instance $f(x, y) = x + y$ can be computed by a TM that starts in configuration $q_0\langle x\rangle\#\langle y\rangle B\ldots$ and ends with the configuration $\langle x\rangle\#\langle y\rangle B\langle x + y\rangle Bq_f\ldots$.

   Define the TMs computing the functions $f(x, y) = x$ and $g(x, y) = y$.
5. (Worth 25% of the CA mark) Define a TM that computes $f(x, y) = x + y$.
6. (Worth 25% of the CA mark) Define a TM that computes $f(x, y) = x \cdot y$ (Hint: it is easier to solve this problem if you look at the multiplication as a repeated sum, $3 \times 4 = 3 + 3 + 3 + 3$).

Generalisations

Two-way infinite tape

Multidimensional TM's The tape is a k-dimensional grid.

Multiple heads The machine has k heads (but a single tape).

Multiple tapes The machine has k tapes and k heads (one for each tape).

Offline TM's The machine has k *working tapes* plus a read-only input tape.

Nondeterministic TM's (NTM) The machine's transitions can be non-deterministic. If the machine is in some state q and sees a symbol S it may (completely arbitrarily) make one move out of a range of possibilities (not just one).

Hence a computation may not be a linear sequence of configurations. At different time points the computation may branch off into different paths. The TM accepts its input if it ends up in a final state in at least one of its computation paths.

Deterministic vs. Non-deterministic TMs

Let's go back to a common subsequence problem. Suppose we wanted to find a common subsequence of length k of two given strings X and Y, defined over the alphabet $\{0, 1\}$.

A deterministic machine for this problem would start with a configuration of the form

$$q_0 X \# Y \# k$$

The TM could then exhaustively (or through dynamic programming) look for all possible ways in which a subsequence of the appropriate length can be shared by X and Y and eventually leave the tape in the configuration

$$X \# Y \# k B Z B q_f$$

if Z is a common subsequence of length k of X and Y.

Non-determinism ... it's just magic!

The non-deterministic TM would guess a string of length k and then simply check that it is a sub-string of both X and Y. Additional complications are just a consequence of the very simple computational model.

State description

With slightly more details:

1. In the initial state q_0 the machine will just replace the leftmost character (either 0 or 1) with Z or U respectively. Move right and into state "start1".
2. In state "skp1" the machine would move right to the first '#' leaving everything unchanged (it is jumping over the first sequence X). On reaching '#' the machine would move to state "skp2".
3. In state "skp2" the machine would move right to the first '#' leaving everything unchanged (it is jumping over the sequence Y). On reaching '#' the machine would move to state "skp3".
4. In state "skp3" the machine would move right to the first B leaving everything unchanged (it is jumping over k). On reaching B the machine would move right and into state "guess".
5. In state "guess" the machine guess a digit (either 0 or 1), move left and into state "bck".

6. In state "bck" the machine moves back to the right end of k and eventually in state "dec".
7. State "dec" is the initial state of part of the Turing machine that decreases the current value of k by one unit, if $k > 0$. The machine then goes back to guessing more symbols if the new value of k is positive. Otherwise in the final part of the computation the guessed string Z is checked against X and Y (details omitted).

Such description can now help building the transition function δ. All we need to do is to make sure that, on each possible input, the various states behave correctly and on undesired inputs the machine either does nothing or enters some failure state. The detailed description is hinted on the following slide

Exercises

(each worth the usual 25% of the module CA component)

- Complete the definition of the non-deterministic LCS Turing machine.
- A list of natural numbers $n_1, n_2, \ldots, n_h$ is an arithmetic sequence if there exists a natural number d such that $n_i - n_{i-1} = d$ for each $i \in \{2, \ldots, h\}$ (in other words $n_i = n_1 + (i-1)d$).

  Design a non-deterministic TM that given three numbers n_1, n_2 and n_3, accepts its input if they form an arithmetic sequence.

  (Hint) design the algorithm using pseudo-code first, then tackle the problem of building the TM transition function. To solve the problem you will need to know how to add two numbers with a Turing machine (non-trivial) and how to delete portions of the tape (simpler).

State	0	1	Z	U	#	B
q_0	(skp1,Z, R)	(skp1,U, R)				
skp1	(skp1,0,R)	(skp1,1,R)			(skp2,#,R)	
skp2	(skp2,0,R)	(skp2,1,R)			(skp3,#,R)	
skp3	(skp3,0,R)	(skp3,1,R)				(guess,B, R)
guess	(guess,0,R)	(guess,1,R)				(bck,0,L) | (bck,1,L)
bck	(bck,0,L)	(bck,1,L)				(dec,B, L)
dec	. . .					

The initial segment of computation on $X = 1101$ and $Y = 10$ is sketched below ('?' stands for a guessed bit).

q_0	1	1	0	1	#	1	0	#	1	0	B	
U	skp1	1	0	1	#	1	0	#	1	0	B	
U	1	skp1	0	1	#	1	0	#	1	0	B	
U	1	0	skp1	1	#	1	0	#	1	0	B	
U	1	0	1	skp1	#	1	0	#	1	0	B	
U	1	0	1	#	skp2	1	0	#	1	0	B	
U	1	0	1	#	1	skp2	0	#	1	0	B	
U	1	0	1	#	1	0	skp2	#	1	0	B	
U	1	0	1	#	1	0	#	skp3	1	0	B	
U	1	0	1	#	1	0	#	1	skp3	0	B	
U	1	0	1	#	1	0	#	1	0	skp3	B	
U	1	0	1	#	1	0	#	1	0	B	guess	B
U	1	0	1	#	1	0	#	1	0	bck	B	?
U	1	0	1	#	1	0	#	1	dec	0	B	?
. . .	. . .											
U	1	0	1	#	1	0	#	0	1	B	guess	?
U	1	0	1	#	1	0	#	0	1	B	?	guess	B

Remarks

The multitape, offline and non-deterministic TM's will be important when we come to talk about computational complexity.

All these models are equivalent to the initial one in the sense that if a language L is accepted (resp. a function f is computed) by a TM T of one of these types then there exists a TM T' of another given type that accepts L (computes the function f).

Definitions of Resource bounded TM's

If for every input of length n the given TM T runs for at most $t(n)$ steps then T is said to be an $t(n)$ *time-bounded TM* or to have *time complexity* $t(n)$.

If for every input of length n the given TM T scans at most $s(n)$ cells on any of its working tapes then T is said to be an $s(n)$ *space-bounded TM* or to have *space complexity* $s(n)$.

We can then define classes of languages recognised by TM's with varying complexity parameters.

DTIME($t(n)$) is the class of all languages which are accepted by a deterministic TM that is $t(n)$ time-bounded.

DSPACE($s(n)$) is the class of all languages which are accepted by a deterministic TM that is $s(n)$ space-bounded.

Similar definitions can be given of non-deterministic complexity classes.

Of particular importance are DTIME(n^k) and DSPACE(n^k), and the "union" classes P = $\bigcup_k$ DTIME(n^k), NP = $\bigcup_k$ NTIME(n^k), and PSPACE = $\bigcup_k$ DSPACE(n^k).

Plan

In the next few lectures we will prove a number of results on the space complexity of particular families of TMs. We will address two main questions:

1. Do relations between time-complexity and space-complexity classes exist?

   Can we place bounds on the running time of an $s(n)$ space bounded TM?

   It is obvious that a TM T running on x for $t(|x|)$ steps cannot write more than this many cells, hence its space complexity is at most $t(|x|)$ (but obviously it may be better in particular cases!). We will see that we can do better in general.

2. What is the role of non-determinism in space-complexity classes?

   We have heard of the classes P and NP, and of the major open problem in Complexity Theory[a] of determining whether P is actually equal to NP. What can be said about PSPACE and NPSPACE?

[a] Solving this problem WOULD make you rich, even if the money is probably not worth the effort.

Proof systems

Finally we will introduce a different model of computation that can be used to give an alternative characterisation of polynomially space bounded computations and has interesting applications in the area of approximation algorithms.

Space bounds imply time bounds

Can we place a bound on the running time of an $s(n)$ space bounded TM? Well, the machine may loop forever, but if we have a bound on the number of tape cells that the machine uses then we can detect the loop.

> Let T be a k-tape TM and let x be an input such that the computation of T on x uses, at most, s cells on any of the k tapes. Then either T halts on x in at most $|\Gamma|^{sk} s^k |Q|$ steps, where Q and Γ denote the set of the states and the tape alphabet of T, respectively, or it does not halt at all.

Logical Argument

The number $|\Gamma|^{sk} s^k |Q|$ is an upper bound for the number of distinct configurations of T. Hence after these many steps T must "repeat itself"! (After these many steps the computation of T must contain a same configuration twice). But if this is the case, since the computation of T only depends on the sequence of scanned symbols and states the machine goes through, then T will go through the same configuration over an over again.

The result holds for non-deterministic machines as well, and nothing better than this is known.