14/1/2003 E 10.1

Exercises on space complexity

Exercise 1: Let A be an algorithm of space complexity $s(n)$.

Show that there is an algorithm A' such that

- $\mathcal{L}(A) = \mathcal{L}(A')$
- A' has space complexity $s'(n) = O(s(n) + \log n)$
- A' does not scan the input-tape beyond the boundaries of the input

Proof: we proceed in two steps

1) We prove that on input x, there is an algorithm A_1 such that
   - $\mathcal{L}(A_1) = \mathcal{L}(A)$
   - A_1 does not scan the input-tape beyond location $2^{O(s(n)+\log_2 n)}$ from the input

This proof is analogous to the one that we did in class to show that a poly-space bounded (N)TM is equivalent to one that has running time $t(n) \leq c^{q(n)}$ with $q(n) = O(s(n))$ (where $s(n)$ is a polynomial space bound)

We showed $q(n) = 2 \cdot s(n) + d$, where $c = |\Gamma| + |Q|$

In our case:

- $c^{q(n)} = 2^{\log_2 c \cdot q(n)} = 2^{O(s(n))}$
- we have also the position on the input tape that contributes to the configuration:

$$\Rightarrow n \cdot 2^{O(s(n))} = 2^{\log_2 n} \cdot 2^{O(s(n))} = 2^{O(s(n)+\log_2 n)}$$

different configurations at most

Note: a TM with running time $t(n) \leq 2^{O(s(n)+\log n)}$ can scan at most $2^{O(s(n)+\log n)}$ cells of the input tape

E10.2

2) We modify the algorithm A_1 in such a way that it does not move beyond the input.

The resulting algorithm A' works as follows:

- whenever A_1 would move right past the end of the input, A' instead:
  - does not move past the end of the input, but maintains a counter on the work tape
  - whenever A_1 moves right, the counter is incremented
    — " — left, — " — decremented

  In this way, A' can keep track of the position of the input head of A_1

  Whenever A_1 moves back again over the input symbol, A' does not update the counter (leaving it to 0)

- A' operates similarly whenever A_1 moves left past the beginning of the input

How much space does the counter use:

Since A_1 does not scan the input tape beyond $s_I(n) = 2^{O(s(n)+\log n)}$ the counter takes $\log_2 s_I(n) = O(s(n) + \log n)$

Hence, the total space used by A' is

$$s(n) + O(s(n) + \log 2) = O(s(n) + \log n)$$

Exercise 2: Let A be an algorithm of space complexity s.

Show that there is an algorithm A' such that

- A' computes the same function as A, i.e. $A'(w) = A(w)\ \forall w \in \{0,1\}^*$
- A' has space complexity $s'(n) = s(n) + O(\log \ell(n))$ where $\ell(n) = \max_{w \in \{0,1\}^n} |A(w)|$ is the size of the maximum output for input x of length n
- A' never rewrites on the same location of its output tape

Proof:

A' proceeds in successive iterations each time simulating the whole computation of A:

in the i-th iteration, A' outputs the i-th bit of $A(x)$

When emulating A, in its i-th iteration A' proceeds as follows:

- it does not directly (re)write on the output tape
- instead, it maintains on the work tape:
  - the counter i of the next output bit that will be written
  - a counter c of the bit that A is currently writing
  - the value of the bit written by A in position i
- when A would write an output bit, A' operates depending on the values of i and c:
  - if $i \neq c$, then A' does not output anything
  - if $i = c$, then A' stores the written bit on its worktape
- at the end of its simulation, A' outputs the stored bit to the i-th position of the output tape

E10.4

How much space $s'(n)$ does A' use on the working tape for inputs of length n?

- $s(n)$ cells, since it performs the computation of A
- the space for the counters i_e and c_o
  - i_e and c_o have to count positions on the output tape, and hence will use $\log_2 l(n)$ bits each.

We get that $s'(n) = s(n) + O(\log_2 l(n))$

Exercise 11.1.1 b)

E10.5

Consider the problem FALSE-SAT:

Given a boolean expression E that is false when all its variables are made false, is there some other truth assignment that makes E false, besides all-false?

Decide whether the problem is in NP or coNP.

Describe its complement.

If the problem or its complement is NP-complete, prove it.

Proof:

The problem is NP-complete.

- In NP: given a boolean expression E, we need to check:
  1) that E is false when all variables are assigned false
  2) that there is some other truth-assignment making E false

(1) can be done in poly-time by a DTM

(2) can be done in poly-time by a NTM

guess a truth-assignment T different from all false, and answer yes if under T, E evaluates to false

- NP-hard: by a reduction from SAT

Let E be a boolean expression with variables $x_1, \dots, x_n$,

We construct an expression E' s.t. $E \in SAT$ iff $E' \in$ FALSE-SAT

1) Test if E is true when all variables are false (polynomial)

If so, $E \in SAT$, and we convert it to a fixed expression that is in FALSE-SAT, e.g. $x \wedge y$.

2) Otherwise, let E' be $\neg E \wedge (x_1 \vee x_2 \vee \cdots \vee x_n)$. E10.6

Clearly, the reduction is poly-time.

We have that E' is false when all of $x_1, \ldots, x_n$ are false.

Notice that in case (2), E is false when all variables are false.

Hence, if $E \in SAT$, then it is satisfied by a truth assignment T different from all-false.

Thus, $\neg E$ is made false by T, and $E' \in$ FALSE-SAT.

Conversely, if $E' \in$ FALSE-SAT, then since $x_1 \vee \cdots \vee x_n$ is false only for the all-false truth assignment, there must be some other truth-assignment T that makes $\neg E$ false. Then T makes E true, and $E \in SAT$.