15-455: UCT	**K. Sutner**
Assignment 7	Due: **April 1**, 2022.

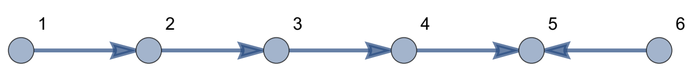
1. Immerman-Szelepsényi (30)

Background

The proof of the Immerman-Szelepsényi theorem depends critically on the inductive nondeterministic "algorithm" that determines the cardinality of the ℓth layer of reachable points $|R_\ell|$ from the previous one, $|R_{\ell-1}|$.

Task

A. Give a "correctness proof" for the Immerman-Szelepsényi "algorithm" by explaining how it handles non-reachability on the graph P_n, where the case $n = 6$ is shown below, $s = 1$ and $t = n$.



In particular explain what happens in the inductive computation of the cardinality of the reachable part.

B. Show that for any reasonable space complexity $s(n) \geq \log n$ we have $\text{NSPACE}(s(n)) = \text{co-NSPACE}$.

Comment For part (A) try to use the notation from slide 12.

2. Graphs and $\mathbb{NL}$ (20)

Background

Recall that a digraph is strongly connected (SC) if there is a path between any two vertices. There are well-known polynomial time algorithms to check whether a graph is strongly connected. Look up Tarjan's algorithm if you are not familiar with it; it's linear time and space, and absolutely amazing. Since connectivity has to do with path existence, one might wonder how SC relates to $\mathbb{NL}$.

For the last part, suppose H is an undirected graph. Recall that H is bipartite if we can partition the vertex set into $V = V_1 \cup V_2$ such that all edges connect V_1 with V_2 (so V_1 and V_2 are both independent sets).

Task

A. Show that SC is in $\mathbb{NL}$.

B. Show that SC is in $\mathbb{NL}$-hard (wrto log-space reductions).

C. Show that bipartiteness can be checked in $\mathbb{NL}$.

Comment For part (C) think about the negation first.

3. Computing Log-Space Functions (20)

Background

Given a transducer M with a separate output tape, it is reasonable to insist that there are no over-writes: every time the machine writes an output symbol, it moves the head one to the right (a write-once output tape). If M is also SPACE$(\log n)$ we call the machine and the corresponding function prompt.

A more tedious approach to computing a function is to associate it with decision problems describing its output, and then insist that the corresponding languages be "easy." More precisely, consider $\mathsf{bin}(z,i)$ to be ith digit in the binary expansion $\mathsf{bin}(z)$ of z, and write $|z|$ for the length of $\mathsf{bin}(z)$. For a function $f : \mathbf{2}^\star \to \mathbf{2}^\star$ define the two languages

$$\mathsf{BIN} = \{\, x\#i \mid \mathsf{bin}(f(x),i) = 1 \,\}$$
$$\mathsf{LEN} = \{\, x\#i \mid i < |f(x)| \,\}$$

If f is polynomially bounded in the sense that $|f(x)| \leq |x|^{O(1)}$, and both BIN and LEN are in $\mathbb{L}$, we call f logish.

Task

A. Show that every prompt function is logish.

B. Show that every logish function is prompt.

4. Overhead-Free LBA (30)

Background

In an LBA we constrain a Turing machine to use no more space than what is initially occupied by the input string $x \in \Sigma^\star$. Alas, we allow the Turing machine to use a tape alphabet Γ larger than Σ, which can be used to erase or mark symbols, compress the input, open a second track, and on so on.

This is slightly unrealistic, though: in a actual machine the alphabet would be fixed once and for all. To model this situation, suppose the input alphabet is Σ. The tape alphabet Γ contains an additional endmarker # but nothing else. The initial tape inscription has the form

$$\# x_1 x_2 \ldots x_{n-1} x_n \#$$

where $x_i \in \Sigma$ and the endmarkers cannot be overwritten or moved. The head is positioned at, say, x_1. So, only n tape cells are available for the computation and a configuration consists essentially of a word $w \in \Sigma^\star$, plus a state and a head position. This type of machine is called an overhead-free LBA.

It is known that overhead-free LBAs cannot accept all CSLs but do accept all context-free languages, and some non-context-free ones. A full proof is too hard, but here a two examples.

Task

For both parts, use a binary alphabet. Construct an overhead-free LBA that recognizes

A. palindromes,

B. all strings of the form $0^n 1^n 0^n$, $n \geq 1$.