

Summary: Lecture 4

Summary for the chapter 7.3 from page 150 on. [3]

Nondeterministic Turing Machine

A nondeterministic Turing machine (*NTM*) has states, which have more than one possible next state for an action. The states are not completely determined by its action and the current symbol it sees, (unlike a deterministic Turing Machine).

NTMs are for example used in thought experiments. One of the most important problems is the P versus NP problem: How difficult it is to simulate nondeterministic computation with a deterministic computer? [1, 3]

Asymmetry of non-determinism

Asymmetry of nondeterministic acceptance:

- Example: find out if a formula φ is satisfiable ($\varphi \in SAT$):
 - choose truth values for the variables nondeterministically
 - check if they make φ become true
- this approach seems to be unpractical so check whether φ is not satisfiable ($\varphi \in \overline{SAT}$)
- Question whether $NP = coNP$ is a statement about *all* options

TODO with book

Asymmetry of nondeterministic space:

- Example: REACHABILITY $\in NL$
 - starting at start node 1
 - algorithm walks through nondeterministically chosen edges $\leq n$ times
 - only current position is remembered ($\log n$ space)
 - accepts if current node is node n
- this approach seems to be unpractical so check if node n is *not* reachable from node 1

TODO with book

$\log n$ space

A graph algorithm using $O(\log n)$ space stores a fixed number of pointers, independent of n , and manipulates them in some way. [2]

Nondeterministically computing functions

- A nondeterministic Turing Machine M computes a function f if the following hold for every input x :
 - one of the computations of M stops in the halting state h with the correct result $f(x)$ on the output tape
 - all computations that do not correctly output $f(x)$ stop instead in a *no*-state (this path failed then)

TODO with book

Questions:

Does this lead to the Haltingproblem?

Problem: GRAPH REACHABILITY

Given a graph G and two nodes $n_1, n_2 \in V$, is there path from n_1 to n_2 ?
A graph $G = (V, E)$ is a finite set V of nodes and a set E of edges as node pairs.

REACHABILITY can be nondeterministically solved in space $\log n$.

Immerman-Szelepscènyi

Theorem (Counting problem):

Given a graph G and a start node x , the number of nodes that are reachable from x in G can nondeterministically be computed in space $\log n$ (where n is the number of nodes of G).

- can be non-deterministically solved
- solving as an extension to REACHABILITY
- counting of nodes that can *not* be reached similar: subtract result from n
→ counting problem and its complement are identical

Algorithm:

- nodes $1, \dots, n$ with start node 1
- $S(i)$ is the set of nodes which are reachable from the startnode with a pathlength of i
 - $S(0)$ will contain node 1
 - $s(1)$ will contain all neighbours of 1
- Algorithm consists out of 4 nested for-loops:
 - outer for loop:
 - * computes number of nodes reachable from initial node (for loop with k steps) as $|S(1)|, |S(2)|, \dots, |S(n-1)|$
 - * $|S(n-1)|$ is the desired answer (n is the number of nodes)
 - * $|S(0)| = 1$ (contains only start node)
 - * $|S(k)|$ is computed after producing $|S(k-1)|$
 - * in each step the previous set is overwritten with the next one because the space is limited
 - second for loop:
 - * it is computed how far the previous steps got and summed up how far it can get
 - * a counter l is initialized to 0
 - * l gets incremented for each node u which is in $S(k)$ (in the end: $l = |S(k)|$)
 - third loop:
 - * deciding if node u belongs to $S(k)$
 - * iterating over all nodes $v \in V$ one by one to reuse space
 - * if node v is in $S(k-1)$, a counter m is incremented
 m counts the members of $S(k-1)$ that were found so far
 - * if $u = v$ or there is an edge from u to v : $u \in S(k)$
→ variable *reply* gets set to true

- * if end is reached:
 - * $u \notin S(k)$ if end is reached and reply is false:
 - if $m < |S(k-1)|$ not all members of $S(k-1)$ have been encountered: return *no*
 - * else return *reply*
- fourth loop:
- * checking whether $v \in S(k-1)$ with non-determinism (similar to REACHABILITY)
- nodes can't be marked (this would use linear space)
 - runs in space $\log n$ with a Turing Machine M
 - M has separate strings holding each of the variables: $k, |S(k-1)|, l, u, m, v, p, w_p, w_{p-1}, input, output$
all of those need only to be compared to each other and incremented by 1
all bounded by n

Questions:

REACHABILITY \in NL

- NL = nondeterministic logarithmic space
- REACHABILITY \in NL

Modify the non-deterministic Turing Machine from above so that it returns *yes* if the innermost subroutine ever reaches the target node n , otherwise, return *no*.

- run previous algorithm
- if target node is found, yes is returned
- else algorithm continues

Questions:

Did I understand this right?

NSPACE is closed under complement

$$\text{NSPACE}(f(n)) = \text{coNSPACE}(f(n))$$

for all proper complexity functions $f(n) \geq \log n$.

TODO

Questions:

References

- [1] Jeff Erickson. *Nondeterministic Turing Machine*. <http://jeffe.cs.illinois.edu/teaching/algorithms/models/09-nondeterminism.pdf>. 2016.
- [2] *Logarithmic Space and NL-Completeness*. http://www.cs.toronto.edu/~ashe/logspace_handout.pdf. 2020.
- [3] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley Publishing Company, 1994.