Question 1 from the last problem set

Algorithm 1:

$Increment(a)$:

Input: a, a positive integer in SLM-representation.

Side effect: a is incremented by 1.

```
1.  j ← 0; c ← 0
2.  do {
3.      a_j ← a_j + c
4.      if (a_j ≥ β) then {
5.          c ← 1; a_j ← a_j − β }
6.      else
7.          c ← 0;
8.      j ← j + 1; }
9.  while (c = 1 and j < n);
10. if c = 1 then a_n ← 1;
```

SLM-representation:

The input is of the form: $\left(s, n, (a_0, \dots, a_{n-1})\right)$ where a_i are β-base digits.

We assume assignment and condition checks take time in $[\tau, \mathrm{T}]$. Let t be the computing time of algorithm 1 with k iterations, then:

$$2\tau + 4k\tau + \tau \le t \le 2\mathrm{T} + 6k\mathrm{T} + 2\mathrm{T}$$

This inequality shows that: $t \sim k$, i.e. the computing time is co-dominant with the number of times the loop is executed.

$\underline{t^-(n) \sim 1}$:

Clearly $1 \preccurlyeq t^-(n)$. In case $a = \beta^{n-1} = (1,0,0,\dots,0)_\beta$ we have one loop body execution and so $k = 1$, hence $t^-(n) \preccurlyeq 1$.

$\underline{t^+(n) \sim n}$:

Clearly $t^+(n) \preccurlyeq n$ since the loop is executed at most n times. For $a = \beta^n - 1 = (1,1,\dots,1)_\beta$ we have n loop executions, so $n \preccurlyeq t^+(n)$.

$\underline{t^*(n) \sim 1}$:

Clearly $1 \preccurlyeq t^*(n)$. The other direction:

For any input at least the loop body is executed at least once. For any $1 \le k \le n-1$ we have k additional loop executions, if $a_0 = \cdots = a_{k-1} = \beta - 1$, and this occurs with probability $\frac{1}{\beta} \cdot \dots \cdot \frac{1}{\beta} = \frac{1}{\beta^k}$. Therefore the expected number of executions is at most $1 + \frac{1}{\beta} + \frac{2}{\beta^2} + \frac{3}{\beta^3} + \cdots + \frac{n}{\beta^n} < 1 + S$ where $S = \sum_{i=1}^{\infty} \frac{i}{\beta^i}$. Observe that $\frac{S}{\beta} = \frac{1}{\beta^2} + \frac{2}{\beta^3} + \cdots \Rightarrow S - \frac{S}{\beta} = \frac{1}{\beta} + \frac{1}{\beta^2} + \cdots =$

$= \frac{1}{\beta} \cdot \frac{1}{1-\frac{1}{\beta}} = \frac{1}{\beta} \cdot \frac{\beta}{\beta-1} = \frac{1}{\beta-1} \Rightarrow S = \cdots = \frac{\beta}{(\beta-1)^2}$

If $\beta = 2$ then we have < 3 loop executions. If $\beta \ge 3$ we have < 2 loop executions, on average. Therefore the average computing time is bounded by a constant, hence $t^*(n) \preccurlyeq 1$.

The Class NP – Nondeterministic Polynomial Time

Hamiltonian path:

The Hamiltonian path problem: $HAMPATH = \{\langle G, s, t\rangle |\ G \text{ is a graph and contains a Hamiltonian path from } s \text{ to } t\}$ where a Hamiltonian path is a path that traverses each node of the graph exactly once.

The brute force method has to look at exponentially many paths. But, given a path that is a **certificate**, i.e. a Hamiltonian path, it can be verified as such in linear time.

Composites:

$COMPOSITES = \{n \in \mathbb{Z}^+ \mid n = p \cdot q, p, q \neq 1\}$. Since the inputs n are not given in unary representation, checking in brute force all possible divisors is exponential. However, it can be verified in polynomial time: given a divisor as a certificate.

Actually, for this problem there was discovered a poly-time decider.

Verifier:

A verifier V for language $A \subset \Sigma^*$ is an algorithm such that $A = \{w \mid V\ accepts\ \langle w, c\rangle\ for\ some\ string\ c\}$. Here c is the certificate.

Definition of NP: $NP = \{L \subset \Sigma^* \mid L\ has\ a\ polytime\ verifier\}$.

Theorem

NP is the class of all languages that can be decided by NTM in poly-time.

Another definition:

$NTIME\big(t(n)\big) = \{L \subset \Sigma^* \mid L\ is\ decided\ by\ NTM\ in\ time\ t(n)\}$

And then: $NP = \bigcup_{k=0}^{\infty} NTIME(n^k)$

Clique:

$Clique = \{\langle G, k\rangle \mid G\ is\ a\ graph\ that\ has\ a\ k-clique\}$.

It is clear that $P \subset NP$. The question is: $NP \subset P$? – Probably not.

Polynomial-time reduction

Let $f: \Sigma^* \to \Sigma^*$ such that $w \in A \Leftrightarrow f(w) \in B$ and f runs in polynomial time, then $A \leq_P B$.

- $B \in P \Rightarrow A \in P$
- $A \notin P \Rightarrow B \notin P$

Example: $3SAT \leq_P CLIQUE$

3SAT formulae are of the form $(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee x_1) \wedge \ldots$

The reduction: we construct a graph that has a node for each literal. We connect all pairs of nodes **except** nodes that appear in the same clause, or nodes that represent negating literals (e.g. x_1 and $\overline{x_1}$).

The resulting graph will have a k-clique $\Leftrightarrow$ the formula with k clauses is satisfyable.