What, if anything, can be done in linear time?

Yuri Gurevich

Tallinn, April 29, 2014

Agenda

1. What linear time? Why linear time?
2. Propositional primal infon logic
3. A linear time decision algorithm
4. Extensions with
    1. Disjunction
    2. Conjunctions as sets
    3. Transitivity

WHAT LINEAR TIME?
WHY LINEAR TIME?

Why

- Big data.

- Remark. In many cases, big-data algorithms are approximate and randomized, necessarily so.

What linear time?

- A short answer:
  We use the standard computation model of the analysis of algorithms.
- A longer answer, with examples and all, follows.

Example 1: Sorting.

- A well-known lower bond is this:
  Sorting n items requires $\Omega(n \cdot \log(n))$ comparisons and thus $\Omega(n \cdot \log(n))$ time.
- There is no way around the lower bound.
  Or maybe there is?

An array A if length n

- Indices: 0, 1, …, n-1
- Values A[0], A[1], …, A[n-1]

Distinct natural numbers $< n$ can be sorted in time $O(n)$.

We illustrate this with
$n = 7$ and $A = \langle A[0], A[1], A[2] \rangle = \langle 3,6,0 \rangle$.

1. Create and auxiliary array B and zero it: $B = \langle 0,0,0,0,0,0,0 \rangle$.
2. Traverse A; for each value k, set $B[k] = 1$. B becomes $\langle 1,0,0,1,0,0,1 \rangle$.
3. Traverse B outputing indices with positive values: $\langle 0,3,6 \rangle$.

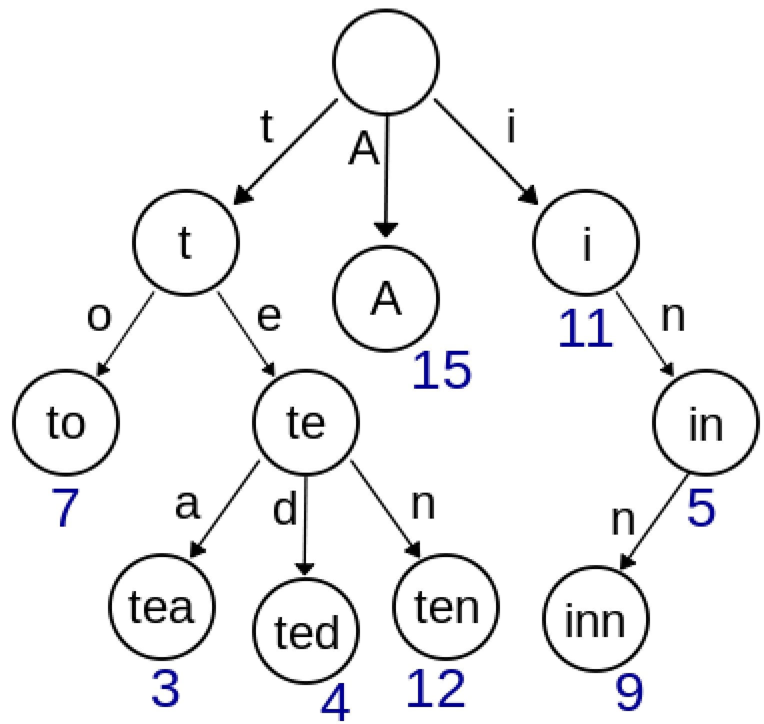
We forgo interesting generalizations.

The computation model

- Random Access Machine
  with registers of length $O(\log n)$.
  - Only the initial polynomial many registers are used, with address of length $O(\log n)$.
  - Relations $=, \geq, \leq,$ and operations $+, -$ are constant time.
- The model reflects the standard computer architecture and the regular intuition of programmers.

Example 2: Tries

One application:
lexical analyzers



to, tea, ted, ten, A, inn

Example 3: Suffix arrays.

- Let $s = c_0 \ldots c_{n-1}$. Each $i < n$ is the key for the suffix $c_i \ldots c_{n-1}$.
- The *suffix array* for s is an array A of length n of s where each $A[j]$ is (the key of) the j-th suffix in the lexicographical order.
- An amazing algorithm constructs the suffix array in linear time.

Parsing logic formulas

- Using the tools above + a deterministic pushdown automaton, produce – in linear time – the parse tree of a given logic formula.
- The nodes and edges are decorated with useful labels and pointers.
- Two nodes may represent different occurrences of the same subformula; call them *homonyms*. All pointers $H(u)$ from any node u to its *homonymy original* can be constructed in $O(n)$.

PROPOSITIONAL PRIMAL INFON LOGIC

Motivation for primal logic

- Access control. DKAL

Why propositional?

- DKAL rules have the form

$$v_1 \colon T_1, v_2 \colon T_2, \ldots$$
$$\text{upon } \pi(w_1, \ldots)$$
$$\text{if } \alpha(\ldots)$$
$$\text{actions}$$

Meaning: If an arriving message fits the pattern π and if the condition α follows from your knowledge assertions, perform the actions.

- Often, by the time you arrive to check α, it is ground. The assertion are typically not ground but only few particular ground instances are relevant.

Expository simplifications

- For expository reasons, we restrict attention to the “topless” (without $\top$) fragment that is quote-free.

The derivation rules

$$\frac{x \wedge y}{x} \qquad \frac{x \wedge y}{y} \qquad \frac{x, y}{x \wedge y}$$

$$\frac{x, x \rightarrow y}{y} \qquad \frac{y}{x \rightarrow y}$$

The subformula property

- Theorem. If

$$\alpha_1, \dots, \alpha_\ell$$

  is a shortest derivation of φ from H
  then every α_i is a subformula of H, φ.
- In the "quoteful" case, instead of subformulas of a formula α, we have formulas *local to* α. There are $< |\alpha|$ such local formulas.

An interpolation lemma of sorts

- Lemma. If $H \vdash \varphi$ then there is a set I of subformulas of H that are also subformulas of φ, such that

1. Formulas I are derivable from H, and
2. φ is derivable from I using only introduction rules.

- We will not use the interpolation lemma but it gives a useful optimization in the case where the hypotheses change rarely.

The multi-derivation problem

- Definition. Given sets H (hypotheses) and Q (queries) of formulas, decide which queries follow from the hypotheses.
- Theorem. The multi-derivation problem for propositional infon logic is solvable in linear time.
- We explain the main ideas.
- n is always the input size, essentially $|H| + |Q|$.

A LINEAR TIME DECISION ALGORITHM FOR THE MULTI-DERIVATION PROBLEM

Approach: derive them all

Compute all subformulas of H, Q derivable from the hypotheses H.

High-level algorithm

- Initially all subformulas of H, Q are raw,
  only hypotheses are pending and
  there are no processed formulas.
- Pick the first pending formula α,
  apply all possible inference rules to α,
  then mark α processed.
  - In the process some raw formulas may become pending.
- Repeat until no formula is pending.

One easy case

- Apply the ∧-elimination rule $\frac{x \land y}{x}$.
- In this case, α is a conjunction. If the first conjunct of α is raw, mark it pending.

One harder case

- Apply the ∧-introduction rule $\frac{x,y}{x \wedge y}$ with α playing the role of x.
- All raw formulas of the form $\alpha \wedge y$ where y is pending or processed, should be marked pending.
- How do we find them? We don't have the time to walk through the raw formulas.

Local search

- Every homonymy original node u is endowed with four so-called *use sets* denoted
$$(\wedge, l), (\wedge, r), (\rightarrow, l), (\rightarrow, r)$$
computed as follows.
- Traverse the parse tree, in the depth-first way.
- If a homonymy original u is the left child of a conjunction node w, put $H(w)$ into the use set $(\wedge, l)$ of u. If u is the right child of w, put $H(w)$ use $(\wedge, r)$ instead.
- Similarly for $\rightarrow$.

Back to applying $\frac{x \wedge y}{x}$

- Recall: we are looking for raw formulas of the form $\alpha \wedge y$ where α is the first pending formula.
- Just walk through the use set $(\wedge, l)$ of α.

EXTENTION 1: DISJUNCTIONS

Motivations

Recall the DKAL rule

$$v_1 : T_1, \ldots, v_j : T_j$$

upon $\pi(w_1, \ldots)$

if $\alpha(\ldots)$

actions

and suppose that $\alpha = \beta \vee \gamma$, e.g.

passport(traveller,UK) ∨ passport(traveller,EU).

There may be many such disjunctions. They may be eliminated but they make rule much more succinct.

Add only introduction rules

$$\frac{x}{x \lor y} \qquad \frac{y}{x \lor y}$$

The linear decision algorithm generalizes in a rather obvious way.

EXTENSION 2: CONJUNCTIONS (AND DISJUNCTIONS) AS SETS

Motivation

While $x \wedge y$ entails $y \wedge x$,

- $(x \wedge y) \to z$ doesn't entail $(y \wedge x) \to z$,
- $z \to (x \wedge y)$ doesn't entail $z \to (y \wedge x)$,
- $(x \wedge y) \wedge z \to w$ doesn't entail $x \wedge (y \wedge z) \to w$, etc.

The idea, a problem, and a solution

- View conjunctions as sets of conjuncts.
  This repairs the missing entailments.
- But sets are not constructive objects.
- Represent sets as sequences by ordering the conjuncts lexicographically.

The decision algorithm

- The resulting multi-derivability problem is solvable in expected linear time.
- It is the algorithm that introduces randomization. No probability distribution on inputs is assumed.

EXTENSION 3: TRANSITIVE PRIMAL INFON LOGIC

Motivation

- In primal infon logic,
  $(x \to y), (y \to z)$ don’t entail $(x \to z)$.

New axiom and rule

- In the quoteless case, transitive primal infon logic is the extension of primal infon logic with an axiom $x \to x$ and the rule

$$\frac{x \to y, \;\; y \to z}{x \to z}$$

An alternative presentation of transitivity

$$\frac{x_1 \to x_2, x_2 \to x_3, \dots, x_{k-1} \to x_k}{x_1 \to x_k}$$

Logically the alternative presentation is equivalent to the original one but algorithmically it makes a lot of difference.

Multi-derivability

- Multi-derivability problem for the transitive primal infon logic is solvable in quadratic time.

THANK YOU

VAULT

High-level algorithm

Initially all local formulas are *raw*,
except that hypotheses are *pending*.
No formulas are *processed*.

1. Pick the first pending formula α,
2. apply all (applicable) inference rules R to α;
   if any of the conclusions are raw, make them pending.
3. mark α processed.
4. Repeat until no formula is pending.

- Pending and processed formulas have been derived.
- Formulas move only from raw to pending to processed.

One easy case

- $\alpha = \beta \wedge \gamma$, R is $\frac{x \wedge y}{x}$.
- If β is raw, mark it pending.

One harder case

- Apply $R = \frac{x, y}{x \wedge y}$ to α, with α being the left premise.
  - It will be convenient to abbreviate this sentence thus: apply R_l to α.
- All raw formulas $\alpha \wedge y$, with y pending or processed, should be marked pending. But how do we find them?

Succinct representation, 1

- Local formulas are too big objects to manipulate in linear time. So we work with the parse tree of H, Q. The subtree rooted at a node u of ParseTree(H, Q) is the parse tree of some formula φ, the *formula* of u.
- Draft definition. If φ = Formula(u) then u represents φ.
- But then φ may have many representations.
- Call nodes u, v *homonyms* if their formulas are isomorphic.

Succinct representation, 2

- *Lemma*. There is a linear-time algorithm that
  - chooses a *homonymy leader* in every homonymy class, and
  - sets pointers Hu from any node u to its homonymy leader.
- The algorithm uses suffix arrays.
- *Def*. If φ = Formula(u) then Hu represents φ. Further, $Hu = \mathrm{N}ode(\varphi)$.

The use sets $\mathrm{US}(R_l, u)$

- Traverse the parse tree in the depth-first manner. For every homonymy leader w with Formula(w) $= x \wedge y$,

  put w into the use set $\mathrm{US}(R_l, Hw_l)$.

  – Here w_l is the left child of w.

  – Notice that Hw_l =Node(x).

  – Notice that every Node($\alpha \wedge y$) occurs in US(R_l,Node(α)).

Applying R_l to α

- Walk through $\mathrm{US}(R_l, \mathrm{Node}(\alpha))$ and mark every raw w there pending.
- How do you find Node(α)?
  That is how α is given in the first place.