ALG 1.1
Models of Computation:
(a) Random Access Machines (RAMs)
(b) Straight Line Programs and Circuits
(c) Decision Trees
(d) Machines That Make Random Choices

Main Reading Selection:
CLR, Chapter 1

Auxiliary Reading Selections:
AHU-Design, Chapter 1
BB, Sections 1.1-1.5, 1.8
AHU-Data, Chapter 1

RANDOM ACCESS MACHINE
**RAM assumptions**

(1) each register holds an integer
(2) program can't modify itself
(3) memory instructions involve simple arithmetic
  a) addition, subtraction
  b) multiplication, division

and control statements (goto, if-then, etc.)

Written in "Pidgin Algol"

\[
\begin{align*}
\text{r} & \leftarrow \text{constant} \\
\text{r}_3 & \leftarrow \text{r}_1 \quad \text{op} \quad \text{r}_2 \\
\text{op} & \in \{+, -, \times, \div\} \\
\text{goto} & \text{ label} \\
\text{if} \quad \text{r} = 0 \quad & \text{then} \quad \text{goto} \quad \text{L} \\
\text{read} & \quad (\text{r}) \\
\text{write} & \quad (\text{r})
\end{align*}
\]

**Complexity Measures of Algorithms**

\[
\begin{align*}
\text{Time}_A(X) & = \text{time cost of Algorithm A, input X} \\
\text{Space}_A(X) & = \text{space complexity}
\end{align*}
\]

- **worst case**
  - time complexity
  \[
  T_A(n) = \max_{X:|X|=n} (\text{Time}_A(X))
  \]

- **average case**
  - for random inputs
  \[
  E(T_A(n)) = \sum_{X:|X|=n} \text{Time}_A(X) \cdot \text{Prob}(X)
  \]

- **worst case**
  - space complexity
  \[
  S_A(n) = \max_{X:|X|=n} (\text{Space}_A(X))
  \]

- **average case**
  - for random inputs
  \[
  E(S_A(n)) = \sum_{X:|X|=n} \text{Space}_A(X) \cdot \text{Prob}(X)
  \]

*Note: "time" and "space" depend on machine*
UNIFORM COST CRITERIA

\[
\begin{align*}
\text{time} &= \#\text{RAM instructions} \\
\text{space} &= \#\text{RAM memory registers}
\end{align*}
\]

LOGORITHMIC COST CRITERIA

\[
\begin{align*}
\text{time} &= L(i) \text{ units per RAM instruction on integer size } i \\
\text{space} &= L(i) \text{ units per RAM register size } i
\end{align*}
\]

where \( L(i) = \begin{cases} 
\log |i| & i \neq 0 \\ 
1 & i = 0
\end{cases} \)

example

\[
\begin{align*}
Z & \leftarrow 2 \\
\text{for } & \ k = 1 \ \text{ to } \ n \ \text{ do } \ Z \leftarrow Z \cdot Z \\
\text{output } & \ Z = 2^n
\end{align*}
\]

uniform \quad \text{time cost} = n

logarithmic \quad \text{time cost} > 2^n

Varieties of Computing Machine Models

RAMs,
straight line programs,
circuits,
bit vectors,
lisp machines
:
Turing Machines
**Straight Line Programs**

**idea**

- fix \( n = \) input size
- unroll each iteration loop
- until result is
- loop-free program \( \Pi_n \)

**note:** this is only possible if we can eliminate all branching and all indirect addressing

⇒ for each \( n > 0 \), get a distinct program \( \Pi_n \)

**Example**

Given polynomial

\[ p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0 \]

with constant coefficients \( a_0, a_1, \ldots, a_n \)

**Horner's Rule for Polynomial Evaluation**

**RAM program in \( 2n \) steps**

- input \( X \)
- \( Y \leftarrow a_n \)
- \( Y \leftarrow (Y \cdot X) + a_i \)
- output \( Y \)

<table>
<thead>
<tr>
<th>( n )</th>
<th>( \Pi_1 )</th>
<th>( \Pi_2 )</th>
<th>( \Pi_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( Y \leftarrow a_1 \cdot X )</td>
<td>( Y \leftarrow a_2 \cdot X )</td>
<td>( Y \leftarrow a_3 \cdot X )</td>
</tr>
<tr>
<td></td>
<td>( Y \leftarrow Y + a_0 )</td>
<td>( Y \leftarrow Y + a_1 )</td>
<td>( Y \leftarrow Y + a_2 )</td>
</tr>
<tr>
<td>2</td>
<td>( Y \leftarrow Y \cdot X )</td>
<td>( Y \leftarrow Y \cdot X )</td>
<td>( Y \leftarrow Y \cdot X )</td>
</tr>
<tr>
<td></td>
<td>( Y \leftarrow Y + a_0 )</td>
<td>( Y \leftarrow Y + a_1 )</td>
<td>( Y \leftarrow Y + a_0 )</td>
</tr>
<tr>
<td>3</td>
<td>( Y \leftarrow Y \cdot X )</td>
<td>( Y \leftarrow Y \cdot X )</td>
<td>( Y \leftarrow Y \cdot X )</td>
</tr>
</tbody>
</table>
Straight Line Programs $\leftrightarrow$ Circuits

1-1 correspondence

(DAG) graph model for straight line programs

Input: $x_1, x_2$
Output: $Y$

$
\begin{align*}
\Pi_3 \\
Y &\leftarrow a_3 \cdot X \\
Y &\leftarrow Y + a_2 \\
Y &\leftarrow Y \cdot X \\
Y &\leftarrow Y + a_1 \\
Y &\leftarrow Y \cdot X \\
Y &\leftarrow Y + a_0 \\
\end{align*}
$

Restrictions:

1. All memory registers have value 0 or 1.
2. Use only logical operations: $\land, \lor, \oplus, \neg$.

Boolean Circuits (for VLSI design)
Vector Machines

(math modeling CONVEX computer)
logical operations \( \lor, \land, \oplus, \neg \)

applied to vector elements

memory locations hold boolean vectors

\[
\begin{array}{cccccc}
0 & 1 & 0 & 1 & \cdots & 0 \\
\end{array}
\]

may also allow vector shift operations

Example

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
1 & 0 & 1 & 0 & 0 \\
2 & 0 & 0 & 1 & 1 \\
3 & 0 & 0 & 0 & 1 \\
4 & 1 & 0 & 0 & 0 \\
\end{array}
\]

decision tree

Decision Trees

Example

Sorting

\[
\begin{array}{c}
\text{input } a, b, c \\
\end{array}
\]

To sort \( n \) keys

any decision tree must have \( n! \) output leaves

\( (n! = \# \text{ permutations of } n \text{ keys}) \)

hence height of tree is

\[
\geq \log_2 (n!) \geq c n \log n
\]
Lisp Machines

Complexity theory for AI

input tape

Heap

registers

program

output tape

operations

\[
\begin{align*}
\text{CDR (} r_i \text{)} & \leftarrow \text{CAR (} r_i \text{)} \\
\text{CDR (} r_j \text{)} & \leftarrow \text{CAR (} r_j \text{)} \\
\text{CAR (} r_i \text{)} & \leftarrow j \\
\text{CDR (} r_j \text{)} & \leftarrow j
\end{align*}
\]

Some problems are faster on LISP machine

Carl Fredrik Yngve Strömgren

"The VW of Machines"

"The Ultimate Program Language of Theory"

Turing Machine (TM)

read only

Finite State Control

TM


read/write memory tapes

write only output tape

Invented by Turing (a Cambridge logician)

Built by British for WWII cryptology!

T(n) = time cost = max steps of TM

S(n) = space cost = max cells written by TM on memory tapes
Reductions between TM and RAM models

(1) Given TM time cost $T(n)$ then 
$$
\exists \text{ equivalent Ram (obvious)}
$$

Time cost
- $c \cdot T(n)$ if uniform cost
- $c \cdot T(n) \cdot (\log n)$ if logarithmic cost

(2) Given RAM time cost $T(n)$ with logarithmic cost then 
$$
\exists \text{ equivalent TM with time cost } c' \cdot T(n)^2
$$

Proof idea:

- Registers: $r_0, r_1, r_2, \ldots, r_k$
- Memory: 
  - # | | | | | | | | | | | | | | |
  - # | | | | | | | | | | | | | | |
  - # | | | | | | | | | | | | | | |
  - # | | | | | | | | | | | | | | |
  - # | | | | | | | | | | | | | | |

Do arithmetic by Grammar School Method

Extensions of RAMS:

Reasonable
- (0) Modifiable Program
- (1) Random Choices
- (2) Non-uniformity

Not Reasonable
- (3) Non-deterministic Choices
**RASP Machine**

same as RAM but allow

program to change itself

---

**Randomized Machines**

- **Proof idea**
  - Let RAM use memory registers to store modifiable program of RASP (due to Von Neumann)

- **Extend RAM instructions to include**

  \[ r \leftarrow \text{RANDOM}(k) \]
  gives a random k bit number

- Let \( A_R(x) \) denote randomized algorithm with input \( x \), random choices \( R \)

- **Expected Time**

  \[
  \text{Expected Time Complexity } \quad T(n) = \max_{|x| \neq |x|} \text{Time (x)}
  \]

- **Expected Time**

  \[
  \frac{\text{input X}}{\text{Time (X)}} = \sum_{\forall R} \text{Time (X) Prob (R)}
  \]
A Randomized Computation
(1) If machine outputs value $v$ with prob $\frac{1}{2}$, then $v$ is considered its output.

(2) Machine accepts input $X$ if outputs 1 with prob $\frac{1}{2}$.

(3) Has 1-sided error if when not accepting 1, outputs only 0.

**Non-uniformity**

for each input size $n$, allow the program a distinct, finite, "advice tape" to read

1 0 1 1 \ldots 0

**Note**

if advice tape length $2^n$

can solve any boolean output problem with $n$ boolean inputs (obvious).
**Surprising Result**

[Adelman]

Given any polynomial time randomized algorithm with 1 side error,

\exists a non-uniform deterministic algorithm with polynomial time and polynomial advice!

( tricky proof )

Gives way of derandomizing a randomized algorithm.

---

**Nondeterministic Machines**

allow "nondeterministic choice" branches

if any sequence of choices succeed to accept x, then computation accepts.
**NP** = languages accepted by polynomial time nondeterministic TM machines.

- includes many hard problems:
  1. Traveling Salesman Problem
  2. Propositional Satisfiability
  3. Integer Programming

**P** = languages accepted by polynomial time deterministic TM machines

not known probably no

P = NP ?

---

Another Surprising Result
Levin

If P=NP but we don't know the proof (i.e., the polynomial time algorithm for NP)
find an optimal algorithm to find the solution of any solvable NP search problem, in polynomial time!

proof depends on assumption that there is a finite length program for NP search problems, running in poly time)
Conclusion

(1) There are many possible machine models.

(2) Most (but not Nondeterministic) are "Constructable" - so might be used if we have efficient algorithms to execute on machines.

(3) New machine models can help us invent new algorithms, and vice versa!