Copyright ©2005 Timothy Howard Merrett
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and full citation in a prominent place. Copyright for components of this work owned by others than T. H. Merrett must be honoured. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or fee. Request permission to republish from: T. H. Merrett, School of Computer Science, McGill University, fax 514 398 3883.

The author gratefully acknowledges support from the taxpayers of Québec and of Canada who have paid his salary and research grants while this work was developed at McGill University, and from his students (who built the implementations and investigated the data structures and algorithms) and their funding agencies.

T. H. Merrett ©05/1
Secondary storage must transfer *large blocks* of data to and from RAM..

.. because of *latency*, the *relative* cost, of finding it.

T. H. Merrett

©05/1

2
CS++: Reinventing Computer Science
(for Secondary Storage)

1. Algorithms & Data Structures  cs420
   • variable multidimensional arrays
   • finding all substrings
   • variable-resolution maps
   • data compression

2. Programming Language  cs612
   • software engineering
   • parallel algorithms
   • expert systems
   • object-orientation
   • data mining
   • semistructured data
   • Internet distributed db

T. H. Merrett
1. Algorithms & Data Structures
Variable-sized arrays

E.g., a Leontieff matrix for the economy:

<table>
<thead>
<tr>
<th></th>
<th>out\in</th>
<th>C</th>
<th>F</th>
<th>H</th>
<th>earned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Charles makes Clothes</td>
<td>C</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>Fred makes Food</td>
<td>F</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>Harry makes Houses</td>
<td>H</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>

spent | 6 6 10

Woops, we forgot Pete, who supplies Power:

<table>
<thead>
<tr>
<th></th>
<th>out\in</th>
<th>C</th>
<th>F</th>
<th>H</th>
<th>P</th>
<th>earned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Charles makes Clothes</td>
<td>C</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>Fred makes Food</td>
<td>F</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>Harry makes Houses</td>
<td>H</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>Pete makes Power</td>
<td>P</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>10</td>
</tr>
</tbody>
</table>

spent | 7 9 15 9

Represent these in memory:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>2?</td>
<td>CC</td>
<td>CF</td>
<td>CH</td>
<td>CP</td>
<td>FC</td>
<td>FF</td>
<td>FH</td>
<td>FP</td>
<td>HC</td>
<td>HF</td>
<td>HH</td>
<td>HP</td>
<td>PC</td>
<td>PF</td>
<td>PH</td>
</tr>
<tr>
<td>1.</td>
<td>CC</td>
<td>CF</td>
<td>CH</td>
<td>FC</td>
<td>FF</td>
<td>FH</td>
<td>HC</td>
<td>HF</td>
<td>HH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2!</td>
<td>CC</td>
<td>CF</td>
<td>CH</td>
<td>FC</td>
<td>FF</td>
<td>FH</td>
<td>HC</td>
<td>HF</td>
<td>HH</td>
<td>CF</td>
<td>FP</td>
<td>HP</td>
<td>PC</td>
<td>PF</td>
<td>PH</td>
</tr>
</tbody>
</table>

\[
j \quad 0 \quad 1 \quad 2 \quad 3 \\
i \quad 0 \quad 1 \quad 2 \quad 9 \\
0 \quad 0 \quad 0 \quad 1 \quad 2 \quad 9 \\
1 \quad 3 \quad 3 \quad 4 \quad 5 \quad 10 \\
2 \quad 6 \quad 6 \quad 7 \quad 8 \quad 11 \\
3 \quad 12 \quad 12 \quad 13 \quad 14 \quad 15 \\
\]

1. \( a = j + 3i \)
2. \( a = j + 4i \)
2! \( a = \max(\text{rowbase}(i), \text{colbase}(j)) \) + the other one, \( i \) or \( j \)

Refs: E. J. Otoo ’83; D. E. Knuth ’97
1. Algorithms & Data Structures
Finding all substrings

E.g., Mycobacterium tuberculosis from codon 729

16
letters:
32 bits
16×17/2
letters: 272 bits

Trie: 174 bits
Trie [ref De la Briandais '59]

0123456789012345
atgtcatatgtgatcg

Sequential?

Ref.: J. A. Orenstein '83

T. H. Merrett

©05/1
1. Algorithms & Data Structures
Variable-resolution maps

Map zooming (Ref.: H. Shang ’94)

T. H. Merrett
1. Algorithms & Data Structures

Compression by tries

\[ h \times 2^h \text{ vs } (2^h - 1) \times 2 : \frac{2}{h} = \frac{2}{\lg n} \]

<table>
<thead>
<tr>
<th>( n )</th>
<th>10^3</th>
<th>10^6</th>
<th>10^9</th>
<th>10^{12}</th>
</tr>
</thead>
<tbody>
<tr>
<td>2/\lg n</td>
<td>1/5</td>
<td>1/10</td>
<td>1/15</td>
<td>1/20</td>
</tr>
</tbody>
</table>

| Lossless compression | 80% | 90% | 93% | 95% |

Experimental (Ref.: X. Y. Zhao, ’00)

![Graph showing compression ratio vs. #records](image-url)
2. Programming Language
Software engineering

Ref.: Merrett '84

T. H. Merrett
2. Programming Language
Parallel programming

Matrix multiplication

\[
\text{let } ab \text{ be equiv } + \text{ of } a \times b \text{ by } i, k; \\
AB \leftarrow [i,k,ab] \text{ in } (A \text{ natjoin } B); \\
\text{Leave ordering to implementation:}
\text{already “parallelized”}
\text{NB Domain algebra } \perp \text{ Relational algebra}
\]
Gaussian elimination

let $a'$ be $a$; let $a''$ be $a$;

for row $\leftarrow 1$ to $\text{red max of } i$ in $A$

\{ $A' \leftarrow [j, a']$ where $i = \text{row}$ in $A$;

$A'' \leftarrow [i, a'']$ where $j = \text{row}$ in $A$;

let $aa$ be $(a' \times a'')/A[\text{row}, \text{row}]$;

update $A$ change $a \leftarrow$

if $i \leq \text{row}$ then $a$ else $a - aa$

using $[i, j, aa]$ in

$(A'' \text{ natjoin } A')$

\}

let $ax$ be equiv + of $a \times x$ by $j$;

for row $\leftarrow [\text{red max of } i]$ in $A$ to 1 by $-1$

\{ $AX \leftarrow [ax]$ in $(A \text{ natjoin } X)$;

let $x$ be $(X[\text{row}, \text{red max of } i + 1] - ax)/A[\text{row}, \text{row}]$;

let $j$ be row;

update $X$ add $[j, x]$ in $AX$

relation $X(j, x)$;

T. H. Merrett

CS++

©05/1
2. Programming Language
Expert systems

Horn Clauses  An Inference Engine

<table>
<thead>
<tr>
<th>[New]Facts (Concl)</th>
<th>Horn (Rule#)</th>
<th>Ante</th>
<th>Concl</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Concl] lays eggs</td>
<td>1</td>
<td>lays eggs</td>
<td>is bird</td>
</tr>
<tr>
<td>has feathers</td>
<td>1</td>
<td>has feathers</td>
<td>is bird</td>
</tr>
<tr>
<td>swims</td>
<td>2</td>
<td>flies</td>
<td>is bird</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>is not mammal</td>
<td>is bird</td>
</tr>
<tr>
<td>is bird</td>
<td>3</td>
<td>is bird</td>
<td>is duck</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>swims</td>
<td>is duck</td>
</tr>
<tr>
<td>is duck</td>
<td>3</td>
<td>is brown</td>
<td>is duck</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>is bird</td>
<td>is duck</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>swims</td>
<td>is duck</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>is green</td>
<td>is duck</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>is red</td>
<td>is duck</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>is duck</td>
<td>migrates</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>is not tame</td>
<td>migrates</td>
</tr>
</tbody>
</table>

NewFacts is Facts union

[Concl] in (NewFacts[Concl ⊇ Ante]Horn)

Relixpert expands this 1-line inference engine to 50, in a 200-line expert system shell: TDKE 6 (1991) 151

T. H. Merrett

©05/1
2. Programming Language

Bill of materials

E.g., a wallplug

let $A'$ be $A$; let $S'$ be $S$; let $Q'$ be $Q$;

let $Q''$ be equiv + of $Q \times Q'$ by $A, S'$;

let $Q'''$ be $Q + Q''$; let $Q$ be $Q'''$;

Explo is $[A, S, Q]$ in $[A, S, Q''']$ in (PartOf $[A, S \ union A, S']$

$[A, S', Q'']$ in (Explo $[S \ natjoin A']$ $[A', S', Q']$ in PartOf));

E.g., wallplug has 4 connectors.

T. H. Merrett
2. Programming Language
Object orientation

**proc** bankAccount (Balance, Deposit) **is**

**state** BAL intg

{ **proc** Deposit(dep) **is**
  { BAL <- BAL + dep};
  **proc** Balance(bal) **is**
  { bal <- BAL;
    BAL <- 0
  }
}

Instantiation is join.

**relation** accts(acctno, client) <-

{ (1729, "Pat"),(4104, "Jan")};

Accounts <- accts **natjoin** bankAccount;

( acctno  client  Balance  Deposit  [BAL] )

1729     Pat          [0]
4104     Jan          [0]
Object orientation

update Accounts change Deposit(100)
    using where acctno=4104;

Ref.: Zheng '02

Inheritance

proc interest(Interest) is
state BAL intg;
{   proc Interest(int) is
    {   BAL <- BAL \times (1 + int/100.0)};
}

relation intaccts (acctno, intrate) <- \{(4104, 3)\};
InterestAccounts <- intaccts natjoin interest;
InterestAccounts isa Accounts;
update InterestAccounts change Interest(intrate)
    using intaccts;

( accno client intrate Balance Deposit Interest [BAL] )
1729    Pat    −       −       −       [0]
4104    Jan    3       −       −       [103]

T. H. Merrett
2. Programming Language
Data mining

E.g., Classification by Decision tree
using Datacube

<table>
<thead>
<tr>
<th>Training</th>
<th>(Outlook)</th>
<th>Humidity</th>
<th>Windy</th>
<th>N</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>sunny</td>
<td>high</td>
<td>f</td>
<td>2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>sunny</td>
<td>high</td>
<td>t</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>sunny</td>
<td>normal</td>
<td>f</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>sunny</td>
<td>normal</td>
<td>t</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>overcast</td>
<td>high</td>
<td>f</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>overcast</td>
<td>high</td>
<td>t</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>overcast</td>
<td>normal</td>
<td>f</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>overcast</td>
<td>normal</td>
<td>t</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>rain</td>
<td>high</td>
<td>f</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>rain</td>
<td>high</td>
<td>t</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>rain</td>
<td>normal</td>
<td>f</td>
<td>0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>rain</td>
<td>normal</td>
<td>t</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Datacube

let \( N \) be \( \text{totN} \);
let \( P \) be \( \text{totP} \);
domain \( \text{attr attribute} \);
relation \( \text{AllAttribs}(\text{attr}) \leftarrow \text{AttribsOf Training} \);
// Outlook, Humidity, Windy, \( N \), \( P \)
relation \( \text{ClassAttribs}(\text{attr}) \leftarrow \{(N),(P)\} \);
relation \( \text{TotAttribs}(\text{attr}) \leftarrow \{(\text{totN}),(\text{totP})\} \);
PropAttribs \leftarrow \text{AllAttribs} \text{ diff ClassAttribs} ;
LoopAttribs \leftarrow \text{PropAttribs} ;
while [] in LoopAttribs
{ \text{Attrib} \leftarrow \text{pick LoopAttribs} ;
  \text{update LoopAttribs delete Attrib} ;
  \text{let eval Attrib be "ANY" ;}
  \text{let totN be equiv + of } N \text{ by } (\text{PropAttribs} \text{ diff Attrib}) ;
  \text{let totP be equiv + of } P \text{ by } (\text{PropAttribs} \text{ diff Attrib}) ;
  \text{update Training add [AllAttribs] in}
  \quad [\text{PropAttribs} \text{ diff Attrib union TotAttribs} \text{ in Training} ;
}

The decision tree analysis follows directly; “one-rule” and Bayesian classification methods are special cases.

T. H. Merrett
2. Programming Language
Semistructured data

Text:
Ted married Alice in 1932. Their children, Mary (1934) married Alex in 1954 (Joe was born to Mary and Alex in 1956) and James (1935) married Jane in 1960 (James and Jane had Tom in 1961 and Sue in 1962).

Marked up text (xml):
<Person>
  <Name>Ted</Name> married
  <Family><Conj>Alice</Conj> in <Wed>1932</Wed>. Their children,
  <Children><Name>Mary</Name> (<DoB>1934</DoB>) married
    <Family><Conj>Alex</Conj> in <Wed>1954</Wed>
      :
    </Family>
  :<Children>
</Family>
</Person>

Convert to (recursively nested) relation:
let FAMILY be [Conj, Wed, CHILDREN] mu2nest Family;
let CHILDREN be [DoB, Name, FAMILY] mu2nest Children;
PERSON ← [Name, FAMILY] mu2nest Person;
Semistructured data

Here’s the relation

<table>
<thead>
<tr>
<th>PERSON (Name)</th>
<th>FAMILY (Conj Wed)</th>
<th>CHILDREN (DoB Name)</th>
<th>FAMILY (Conj Wed)</th>
<th>CHILDREN (DoB Name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ted</td>
<td>Alice 1932 1934</td>
<td>Mary Alex 1954</td>
<td></td>
<td>1956 Joe</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1935 James Jane</td>
<td></td>
<td>1961 Tom</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1962 Sue</td>
</tr>
</tbody>
</table>

Queries:

**PERSON/Name**

Ted

**PERSON/FAMILY/CHILDREN/Name**

Mary, James

**PERSON/FAMILY/CHILDREN/FAMILY/CHILDREN/Name**

Joe, Tom, Sue

**PERSON/(.\/)\*Name**

Ted, Mary, James, Joe, Tom, Sue

*Name where FAMILY/Conj="Alice" in PERSON*

Ted
2. Programming Language

Internet

E.g., aldat protocol

Extended names may be used anywhere permissions allow:

\[ F4 \leftarrow \text{aldatp://mimi/~jan/pubA/E3}; \]
\[ \text{aldatp://mimi/~jan/pubA/F3} \leftarrow E2; \]
\[ \text{aldatp://willy/~pat/pubC/}\{F7 \leftarrow E7\}; \]

Joining \( E3(A, B) \) with \( E7(B, C) \) by semijoin:

\( \text{(aldatp://mimi/~jan/pubA/(E3 natjoin} \]
\[ \text{aldatp://willy/~pat/pubC/([B] in E7))) natjoin E7} \]

Ref.: Wang ’02

T. H. Merrett
Conclusions

SS different from RAM => new thinking about:

Computer Science:
- object orientation
- parallel programming
- artificial intelligence
- networking

Applications:
- numerical analysis
- bioinformatics
- G.I.S.
- semistructure

Current work: visualization

Future work:
- constraint databases
- peer-to-peer cooperative work
- agent programming