Introduction to Programming

Bertrand Meyer

Last revised 26 January 2004
Lecture 25:
Topological Sort — 2: Algorithm
Back to software...
Overall structure (1)

Given:

- A type $G$
- A set of elements of type $G$

Required:

- An enumeration of the elements, in an order compatible with the constraints

```plaintext
class ORDERABLE [G] feature

  constraints: LIST [TUPLE [G, G]]

  topsort: LIST [G] is
    ...
  ensure
    compatible (Result, constraints)

end
```
Non-uniqueness

- In general there are several possible solutions

- In practice topological sort uses an optimization criterion to choose between possible solutions.
Overall structure (2)

topsort: LIST [G] is

require

no_cycle (constraints)

...

ensure

compatible (Result, constraints)
Cycles

- \( \subseteq \) must be a partial order: no cycle in the transitive closure of *constraints*
  - No circular chain of the form \( e_0 \rightarrow e_1, \ldots, e_n \rightarrow e_0 \)

- If there are cycles there exists no solution to the topological sort problem!
Overall structure (2)

topsort: LIST [G] is
  require
    no_cycle (constraints)
  ...
  ensure
    compatible (Result, constraints)

Assume there are no cycles in the constraints
  ▪ Not realistic since input may have errors
Overall structure (3)

Don’t assume anything; find cycles as byproduct of attempt to do topological sort

“Attempt to do the topological sort, accounting for possible cycles in the constraints”

if “Cycles found” then
    “Report presence of one or more cycles in constraints”
end
... 

**loop**

“Find a member $p$ of $elements$ for which there is no $q$ in $elements$ such that $p \leq q$”

“Append $p$ to the output list”

“Remove $p$ from $elements$”

**end**
Loop invariant

- Scheme 2: “The transitive closure of constraints has no cycles among the members of elements”

- Scheme 3: “The transitive closure of constraints has no cycles among the members of elements, except for any already implied by the input data”
Algorithm scheme

topsort: \textit{LIST} [T] \textbf{is}
\begin{itemize}
  \item \textbf{do}
  \item \textbf{from}
    \begin{itemize}
      \item read \textit{elements}; read \textit{constraints}
    \end{itemize}
  \item \textbf{variant}
    \begin{itemize}
      \item size of \textit{elements}
    \end{itemize}
  \item \textbf{invariant}
    \begin{itemize}
      \item "\textit{constraints} includes no cycles other than original ones" \textbf{and}
      \item "Order of output so far is compatible with \textit{constraints} \textbf{and}
      \item "All original elements are either output or still in \textit{elements}"
    \end{itemize}
  \item \textbf{until}
    \begin{itemize}
      \item "All members of \textit{elements} (if any) have at least one
        \textbf{predecessor according to \textit{constraints}}"
    \end{itemize}
  \item \textbf{loop}
    \begin{itemize}
      \item \textit{Let x be a task in T which has no predecessor according to C}
      \item \textbf{output} \textit{x}
      \item \textbf{remove} \textit{x} from \textit{T}
      \item \textbf{remove from C all constraints of the form \textit{[x, y]}}
    \end{itemize}
  \item \textbf{end}
\end{itemize}
...
... if $T$ is empty then
   print ("Topological sort complete")
else
   print ("[
     Topological sort incomplete, input not in strict
     partial order
   ]")
   print ("[
     There is at least one cycle relating the
     following elements:")
   print (T)
   print ("The constraints between these elements are:")
   print (C)
end
end
Data structures

- Elements: LIST [G]
- Constraints: LIST [TUPLE [G, G]]

(Number of elements: \( n \)
Number of constraints: \( m \))

- Best we can hope for: \( \mathcal{O}(m+n) \)

- Using elements and constraints as given wouldn’t allow reaching this.

- From these data structures we will produce another better suited to topological sort.

General form is then:

“Build internal data structure presenting constraints”
“Execute topological sort loop on internal data structure”
“Compiling” the original data

- Before processing the data, transform it into a representation that’s more efficient for this processing

- A frequently useful algorithmic scheme (“heuristics”)
Internal data structure for topological sort

- Give a number \( e.number \) to every element \( e \)

- \textit{successors}: ARRAY [LIST[ELEMENT]]
  - Has one entry for every element \( e \), holding list of \( f \) such that there's a constraint \( e \rightarrow f \)
  - List of successors of \( e \) is \( \textit{successors.item (e.number)} \)

- Rank...
- Minima...
End of lecture 24