DESIGNING AN ALGORITHM

**Algorithm:** Solution Method + DataStructure

- *Design:* choosing the solution method for the problem (typically, an optimization problem) and choosing the data-structure.


**Example.** Consider designing an algorithm to find the minimum of \( \text{nums}[0..n−1] \), where the \( A[i] \)’s change frequently.

- Method (divide and conquer):
  
  If we know both \( \min \text{nums}[0..(n−1)/2] \) and \( \min \text{A}[(n+1)/2..n−1] \), then we only need to recompute one of them when a change occurs.

- Tree dataStructure: \( O(\log n) \) comp. per change.

```
       min A[0..4]
        /     \
     /   \
      /    \
```

- Heap-dataStructure: a modified version of this, which uses roughly half as much memory.

**Approximation and Heuristic Algorithms:**

- If finding an optimal solution requires too much computation, one can use a solution method that produces non-optimal (lower quality) solution but uses a less amount of computation.

- Heuristic algorithms do not provide specific error bounds on approximate solution as done by approximation algorithms.
PIPELINE ARCHITECTURE: ABSTRACTION OF A SEQUENTIAL COMPUTATION

Restructuring/Simplifying a Seq. Computation:

- Assume function-calls have no side-effects. The sequential computations below are equivalent ($x$, $z$, $v$ have initial values).

<table>
<thead>
<tr>
<th>Original sequence</th>
<th>After reordering</th>
<th>After reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. $x = f(x)$;</td>
<td>$y = g(z)$;</td>
<td>$y = g(z)$;</td>
</tr>
<tr>
<td>2. $y = g(z)$;</td>
<td>$z = h(y, z)$;</td>
<td>$z = h(y, z)$;</td>
</tr>
<tr>
<td>3. $z = h(y, z)$;</td>
<td>$x = f(x)$</td>
<td></td>
</tr>
<tr>
<td>4. $x = k(x, y, z)$;</td>
<td>$x = k(x, y, z)$</td>
<td>$x = k(f(x), y, z)$</td>
</tr>
<tr>
<td>5. $u = m(x, z)$;</td>
<td>$u = m(x, z)$;</td>
<td></td>
</tr>
<tr>
<td>6. $u = p(x, v)$;</td>
<td>$u = p(x, v)$;</td>
<td>$u = p(x, v)$</td>
</tr>
<tr>
<td>7. $v = f(u)$;</td>
<td>$v = f(u)$;</td>
<td>$v = f(u)$</td>
</tr>
</tbody>
</table>

Use-Define Digraph:

- Nodes are the variables in the reduced computation.
- Links $\{(x, y): x$ is used in defining the value of $y\}$, in the reduced computation.

Pipeline Architecture from Use-Define Digraph:

- An acyclic digraph, with strong-components as "fat nodes".
- Computations proceed from one strong components to the next.

Question:

• If we replace $p(x, v)$ by $p(x, z, v)$, do we still get a pipeline architecture?
OPTIMAL ORDERING OF ITEMS IN A DISPLAY-PAGE: A DESIGN PROBLEM

Acyclic Dependency Relation Among Concepts:

- A concept $x_j$ may directly depend on a concept $x_i$ in the sense that one must understand/recall $x_i$ in order to understand $x_j$.
- Assume: the dependency relations $(x_i, x_j)$ form an acyclic digraph $\overrightarrow{G}$ on the nodes $x_i$.
  - For some $(x_i, x_j)$, $x_i$ may be a generalization or a specialization of $x_j$.
- Each $x_i$ has associated with it a weight $w(x_i) > 0$, which can be thought of as the semantic complexity of $x_i$; a higher $w(x_i)$ means a higher "memory load" to understand and recall $x_i$.
  - We take $w(x_i)$ to be a measure of the required space in a display to describe (or time required to discuss) $x_i$.

Example. Shown below are a digraph $\overrightarrow{G}$ with 5 concepts and 6 links, and one of 3 possible linear (topological) orderings to arrange the concepts on a display page.

![Diagram](image)

The presence of $B$ between $A$ and $D$ in the linear ordering adds to the difficulty of recalling $A$ at $D$; recalling $A$ at $C$ is even more difficult due to $D$.

Design Problem: Find a linear ordering that minimizes the total additional difficulties of concept recalls.
THE DESIGN PROBLEM

Topological Ordering:

- A linear ordering of the nodes \( x \) of an acyclic digraph \( \vec{G} \) such that each link \( (x, y) \) goes from left to right.

**Example.** An acyclic digraph with 3 topo-orderings.

![Graph](image)

**Added Recall Difficulty** \( d_L(x, y) \) for \( x \) at \( y \):

- Given a topo-ordering \( L \) of \( \vec{G} \), \( d_L(x, y) = \sum w(z) \), summed over all nodes \( z \) between \( x \) and \( y \) in \( L \).
- \( d_L(x, y) \) is not a "distance" because for links \( (x, y) \), \( (y, z) \), and \( (x, z) \) in \( \vec{G} \): \( d_L(x, z) - [d_L(x, y) + d_L(y, z)] = w(y) > 0 \).

**Total Added Recall Difficulty** \( R(L) = \sum_{(x_i, x_j) \in \vec{G}} d_L(x_i, x_j) \):

- Node \( x \) contributes \( C_L(x_j) = w(x_j) \times \# \) (links \( (x_i, x_k) \) such that \( x_i < x_j < x_k \) in \( L \)) to \( R(L) \) and \( R(L) = \text{sum of } C_L(x) \)'s.

**Example.** \( R(L_1) = R(L_3) = 20 > 17 = R(L_2) \). The non-zero terms in \( R(L_2) \) are: \( d(A, C) = 6+2 = 8 \), \( d(A, D) = 6 \), and \( d(D, E) = 3 \), and \( C(B) = 12 \), \( C(C) = 3 \), and \( C(D) = 2 \). The optimal ordering is the same here for all \( w(x_j) \)'s.

- The measure \( C_L(x) \) might be useful in choosing successive items in an opt. ordering \( L \).
EXERCISE

1. If \( w(x_i) = \) time needed to discuss the concept \( x_i \), then does the optimal topo-ordering of the digraph below fit your intuition about the order in which one should cover concepts \( \{ A, B, C \} \)?

2. Given an acyclic digraph \( \vec{G} \) and a link \( (x, y) \) in it, let \( d_{\min}(x, y) = \min\{d_L(x, y): L \text{ is a topo-ordering of } \vec{G}\} \). Give a formula to compute \( d_{\min}(x, y) \); verify it for \( \vec{G} \) in the previous page.

3. Show an acyclic digraph \( \vec{G} \) for which there is no topo-ordering \( L \) such that \( d_L(x, y) = d_{\min}(x, y) \) for all links \( (x, y) \).

4. Let \( C_{\min}(x) = \min\{C_L(x): L \text{ is a topo-ordering of } \vec{G}\} \). How can you compute \( C_{\min}(x) \)? Show an acyclic \( \vec{G} \) which has no topo-ordering \( L \) such that \( C_L(x) = c_{\min}(x) \) for all nodes \( x \).

5. Let \( \vec{G}' = \{(y, x): (x, y) \in \vec{G}\} \). Argue that each optimal topo-ordering \( L' \) of \( \vec{G}' \) corresponds to the reversal of an optimal topo-ordering \( L \) of \( \vec{G} \). Then, prove that the optimal topo-ordering of \( \vec{G}' \) is simply the reverse of that for \( \vec{G} \).

A matrix representation: For \( L_2 \), the rows for non-zero \( d_L(x, y) \) are shown below; the column sums give \( C_L(x) \)'s:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>( (A, C) )</td>
<td>6</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( (A, D) )</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( (D, E) )</td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
PATH-CLOSED SUBSET:  
A GENERALIZATION OF INTERVAL

Interval:
• Given a ordered list of items $L = \langle x_1, x_2, \ldots, x_n \rangle$, each subset of the form $L_{i,j} = \{x_i, x_{i+1}, \ldots, x_{j-1}, x_j \}$ is called an interval.

Two Special Kinds of Intervals:
• $Left(x_j) = \{x_i: i \leq j\}$ and $Right(x_j) = \{x_k: k \geq j\}$.

Question:
• How many subsets of $\{x_1, x_2, \ldots, x_n\}$ are intervals in $L$, how many are of the special type, and how many are non-intervals?

Theorem.  Each intervals $L_{i,j}$ is of the form $left(x_p) \cap Right(x_q)$ for some $x_p$ and $x_q$.

Path-Closed Subset of Nodes in a Digraph $\vec{G}$:
• A subset of nodes $S$ in $\vec{G}$ is called path-closed if all nodes on each path $\pi_{x,y}$ that connects two nodes $x, y \in S$ are in $S$.

![Diagram]

Path-closed: $\{A, B, D\}$  Not path-closed: $\{A, D\}$  $\{A, B, C, D\}$  $\{A, B, D, E\}$

Two Special Kinds of Path-Closed Subsets:
• $\text{pred}(x_j) = \{x_i: \text{there is a path from } x_i \text{ to } x_j \text{ in } \vec{G}\}$ and $\text{succ}(x_j) = \{x_k: \text{there is a path from } x_j \text{ to } x_k \text{ in } \vec{G}\}$.

Question:
• Which of $\text{pred}(x_j)$ and $\text{succ}(x_j)$ generalizes which of $Left(x_j)$ and $Right(x_j)$?
EXERCISE

1. How do the intersections of the form \( \text{pred}(x_p) \cap \text{succ}(x_q) \) relate to the path-closed subsets of \( \vec{G} \)?

2. The *transitive closure* of an digraph \( \vec{G} \) is the digraph \( \vec{G}^+ \), which has the same nodes as \( \vec{G} \) and the additional links \( \{(x, y): \text{there is a path } \pi_{x,y} \text{ from } x \text{ to } y \text{ in } \vec{G} \text{ and } (x, y) \not\in \vec{G}\} \). Show that:
   
   (i) If \( \vec{G} \) is acyclic, then \( \vec{G}^+ \) is also acyclic.

   (ii) \( \vec{G} \) and \( \vec{G}^+ \) have the same topo-orderings and hence the same path-closed sets based on results in the next page).

3. Suppose \( \vec{G} \) is a digraph (may not be acyclic) and the sets \( S_1 \) and \( S_2 \supset S_1 \) are path-closed. Now suppose that there is a node \( x_1 \in S_1 \) from which all nodes in \( \vec{G} \) are reachable. Prove that \( S_2 - S_1 \) is also path-closed. Give an example to show that \( S_2 - S_1 \) need not be path-closed without the assumption about \( x_1 \).

\[
\begin{array}{c}
\bullet x_1 \\
S_1 \\
S_2
\end{array}
\]
Each path-closed subset is an interval in some topo-ordering.
Each interval in each topo-ordering is a path-closed subset.

Effects of Merging Nodes in $S$ and Path-Closedness of $S$:
Merging the nodes $S$ in an acyclic digraph $\overrightarrow{G}$ into a single node gives an acyclic-digraph if and only if $S$ is path-closed.

A Topo-ordering of $\overrightarrow{G}$ with Path-Closed set $S$ as An Interval:
Create a topo-ordering $\tau'$ of the digraph $\overrightarrow{G}'$ obtained by merging the nodes $S$ into a single node $x_S$.
Find a topo-ordering $\tau_S$ of the acyclic subdigraph of $\overrightarrow{G}$ on $S$.
Now replace the node $x_S$ in $\tau'$ by $\tau_S$.

Example. Only $L_2$ is obtained in this way for the path-closed set $S = \{A, B, D\}$. There is no topo-ordering with $S' = \{A, D\}$ as an interval because $S'$ is not path-closed.
PROBLEM REDUCTION
BY REDUCTION OF THE DIGRAPH

- Consider the following \( \vec{G} \) and a topo-ordering \( L \) of it.

\[
\begin{align*}
\vec{G}: & \quad x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_4 \rightarrow x_5 \rightarrow x_6 \rightarrow x_7 \rightarrow x_8 \rightarrow x_9 \rightarrow x_{10} \\
L: & \quad x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_4 \rightarrow x_5 \rightarrow x_6 \rightarrow x_7 \rightarrow x_8 \rightarrow x_9 \rightarrow x_{10}
\end{align*}
\]

- If we merge the interval \( \{x_1, x_2, x_3\} \) in \( L \) into a single node \( x_{1:3} \), we get the topo-ordering \( L' \) of the new acyclic digraph \( \vec{G}' \).

\[
\begin{align*}
L': & \quad x_{1:3} \rightarrow x_4 \rightarrow x_5 \rightarrow x_6 \rightarrow x_7 \rightarrow x_8 \rightarrow x_9 \rightarrow x_{10} \\
\end{align*}
\]

The number 2 on link \((x_{1:3}, x_4)\) accounts for two original links \((x_1, x_4)\) and \((x_2, x_4)\), and likewise for \((x_{1:3}, x_7)\); \(C_L(x_4) = 4 = C_{L'}(x_4)\).

- This suggests a generalization of the original problem by introducing a strength \( s(x, y) > 0 \) for the dependence of \( y \) on \( x \).

- We can now assume that \( \vec{G} \) has a unique source node; otherwise, we add a new node \( x_0 \) with \( w(x_0) = 0 \), make \( x_0 \) adjacent to each original source-node \( x_i \), and let \( s(x_0, x_i) = 0 \). The node \( x_0 \) is the new unique source-node and the new optimal topo-orderings are of the form \( \langle x_0 \rangle . L \), where \( L \) is optimal for \( \vec{G} \).

Likewise, we may assume that \( \vec{G} \) has a unique sink-node.
GENERALIZATION OF $C_L(x)$ AND $R(L)$ FOR WEIGHTED LINKS

**Link Strength** $s(x, y)$:

- $s(x, y) \geq 0$ is the *strength* of the link from $x$ to $y$.

  If $s(x, y) = 0$, we cannot remove the link $(x, y)$ because that may affect the possible topo-orderings (except when the link $(x, y)$ is a transitive link) and hence the optimal ordering.

- While $w(x_j)$ indicates the amount of stretching of a link $(x_i, x_k)$ when $x_j$ is placed between $x_i$ and $x_k$ in a topo-ordering, $s(x_i, x_k)$ accounts for the impact on stretching of $(x_i, x_k)$, which we define to be $w(x_j) \times s(x_i, x_k)$.

**Generalization of $C_L(x_j)$ and $R(L)$:**

- $C_L(x_j) = w(x_j) \times \sum_{i<j<k, (x_i, x_k) \in \bar{G}} s(x_i, x_k)$

- $R(L) = \sum_{all \ x_j} C_L(x_j)$.

**Example.** Consider the link-weighted acyclic digraph $\bar{G}$ below with each $w(x) = 1$ and the topo-ordering $L = \langle A, B, C, D, E \rangle$. Then, $C_L(A) = 0$, $C_L(B) = 1 = C_L(D) = 0$, and $C_L(C) = 2$, giving $R(L) = 4$.

(i) Acyclic $\bar{G}$.

(ii) A topo-ordering $L$. 
EXERCISE

1. Give a generalization of $d_L(x, y)$ using the link-strengths so that we still have
   \[ \sum_{(x, y) \in \vec{G}} d_L(x, y) = \sum_{x \in \vec{G}} C_L(x) \]

2. Consider a topo-ordering $L = \langle x_1, x_2, \ldots, x_k, \ldots \rangle$ of an acyclic digraph $\vec{G}$, where the first $k$ nodes of $L$ are fixed. Let $S = \{x_1, x_2, \ldots, x_k\}$. Then, for $1 \leq j \leq k$, $C_L(x_j) = \sum_{1 \leq p < j < q \leq k} s(x_p, x_q) + \sum_{p < j, y \notin S} s(x_p, y)$. The second sum in $C_L(x_j)$ is independent of the ordering of the remaining nodes of $\vec{G}$ and it can be computed based on the initial part $\langle x_1, x_2, \ldots, x_k \rangle$ of $L$.
   - If the order of the remaining nodes in $L$ are $\langle x_{k+1}, x_{k+2}, \ldots, x_n \rangle$ and $L' = \langle x_1:k, x_{k+1}, x_{k+2}, \ldots, x_n \rangle$ is the corresponding topo-ordering of the digraph $\vec{G}'$ obtained by merging the nodes $S$ into a single node $x_{1:k}$, then prove that for $k + 1 \leq j \leq n$, $C_L(x_j) = C_{L'}(x_j)$.
   - Now, show that $L = \langle x_1, x_2, \ldots, x_k, x_{k+1}, x_{k+2}, \ldots, x_n \rangle$ is optimal for the given initial part $\langle x_1, x_2, \ldots, x_k \rangle$ if and only if $L' = \langle x_1:k, x_{k+1}, x_{k+2}, \ldots, x_n \rangle$ is optimal for $\vec{G}'$.

3. Show that $\langle x_1, x_2, \ldots, x_k, x_{(k+1):n} \rangle$ is an optimal topo-ordering of $\vec{G}''$ obtained by merging $\{x_{k+1}, x_{k+2}, \ldots, x_n\}$ into a single sink-node $x_{(k+1):n}$ if and only if $\langle x_1, x_2, \ldots, x_k, x_{k+1}, x_{k+2}, \ldots, x_n \rangle$ is an optimal topo-ordering of $\vec{G}$ for some choice of the ordering $\langle x_{k+1}, x_{k+2}, \ldots, x_n \rangle$ of the remaining nodes of $\vec{G}$. 
PROBLEM REDUCTION
BY TRANSITIVE LINK ELIMINATION

Transitive Link:
- The link \((x, y)\) is *transitive* if there is an alternate \(xy\)-path \(\langle x, x_1, x_2, \ldots, x_k, y \rangle\), \(k \geq 1\).

Removing \((x, y)\):
- Select an arbitrary alternate \(xy\)-path \(\langle x, x_1, x_2, \ldots, x_k, y \rangle\), \(k \geq 1\).
- Let \(x_0 = x\) and \(x_{k+1} = y\). Then, for each \(0 \leq i \leq k\), replace \(s(x_i, x_{i+1})\) by \(s(x_i, x_{i+1}) + s(x, y)\).

Impact of \(C_L(z)\):
- For \(z = x\) or \(z = y\) or for \(z\) not between \(x\) and \(y\) in \(L\), there is no change in \(C_L(z)\).
- For \(z = x_i\), \(1 \leq i \leq k\), we have \(x < z < y\) in \(L\) and \(C_L^{new}(z) = C_L^{old}(z) - s(x, y)\).
- For other \(z\) between \(x\) and \(y\) in \(L\), the contribution of \((x, y)\) to \(C_L(z)\) is taken care of by that of the unique link \((x_i, x_{i+1}), x_i < z < x_{i+1}\), and \(C_L^{new}(z) = C_L^{old}(z)\).

\[\text{Impact on } R(L) \text{ and Optimum Topo-Ordering:}\]
- \(R^{new}(L) = R^{old}(L) - k \cdot s(x, y)\).
- No change in optimum topo-orderings; \(R_{opt}^{new}(L) = R_{opt}^{old}(L) - k \cdot s(x, y)\).
A SIMPLE STRUCTURE THEOREM FOR OPTIMAL TOPO-ORDERING

Separating Node $x$:

- $x \neq$ the unique source-node, $x \neq$ the unique sink-node, and for each $y \neq x$, either $y \in \text{pred}(x)$ or $y \in \text{succ}(x)$.

\[\begin{array}{c}
\text{unique} \\
\text{source-node}
\end{array}
\begin{array}{ccc}
x_1 & \cdots & x \\
\vdots & \ddots & \vdots \\
x & \cdots & x_n
\end{array}
\begin{array}{c}
\text{unique} \\
\text{sink-node}
\end{array}\]

Theorem.

- If $x$ is a separating-node, then each topo-ordering is of the form $L = L_1.L_2$, where $L_1 = \langle x_1, x_2, \ldots, x_k, x=x_{k+1} \rangle$ is a topo-ordering of \{x\} $\cup$ pred($x$), $L_2 = \langle x=x_{k+1}, x_{k+2}, \ldots, x_n \rangle$ is a topo-ordering of \{x\} $\cup$ succ($x$), $|\text{pred}(x)| = k$, and $|\text{succ}(x)| = n - k - 1$.

Theorem.

- If $x$ is a separating node, then $L$ is optimal if and only $L_1$ is optimal when we consider the merged node $x_{(k+1):n}$ and $L_2$ is optimal when we consider the merged node $x_{1:(k+1)}$.
- If $\vec{G}$ has no transitive link, then $C_L(x) = 0$ and $R(L) = R(L_1) + R(L_2)$. 
A VALID BUT INEFFICIENT ALGORITHM FOR OPTIMAL TOPO-ORDERING

- Let $x_1 =$ unique source-node, $N^+(x_1) = \{x_2, x_3, \ldots, x_k\}$.
- For $2 \leq j \leq k$, let $\tilde{G}_j$ be the acyclic digraph obtained by merging $x_1$ into $x_j$ (which becomes the unique source-node, denoted by $x_{1,j}$ in $\tilde{G}_j$). $\tilde{G}_j$ may have transitive links.

**Theorem.**

- The topo-ordering $\langle x_1, x_j \rangle. T_{opt}(\tilde{G}_j)$ is optimal for $\tilde{G}$ if its value $w(x_j) \cdot [\text{outWt}(x_1) - s(x_1, x_j)] + R_{opt}(\tilde{G}_j)$ is the minimum among the cases $2 \leq j \leq k$.

**Example.** For $\tilde{G}$ below, we can merge $x_1$ into $x_2$ or $x_1$ into $x_3$. The optimal topo-ordering is given by the one corresponding to the minimum of $w(x_2)s(x_1, x_3) + w(x_3)s(x_2, x_4) + R_{opt}(\tilde{G}_{1,2})$ and $w(x_3)s(x_1, x_2) + w(x_2)s(x_3, x_4) + R_{opt}(\tilde{G}_{1,3})$.

**Question:**

- Show the results of merging $x_{1,2}$ into $x_3$ and $x_{1,3}$ into $x_2$. Then, explain the two key sources of inefficiency in this approach.
FINDING AN OPTIMAL TOPO-ORDERING

Indeg\( (x_i) \), Outdeg\( (x_i) \), in Wt\( (x_i) \), and outWt\( (x_i) \):

- \( N^-(x_j) = \{ x_i: (x_i, x_j) \text{ is a link} \} \), indeg\( (x_j) = |N^-(x_j)| \), and inWt\( (x_j) = \sum s(x_i, x_j) \) summed over \( x_i \in N^-(x_j) \); \( N^+(x_j) \), outdeg\( (x_j) \), and outWt\( (x_j) \) are defined similarly.
- inLessOut\( (x_i) = \) inWt\( (x_i) \) – outWt\( (x_i) \) and \( \delta_{ij} = w(x_i) - w(x_j) \).
- \( \sum \) inLessOut\( (x_i) = 0 \), summed for all \( x_i \).

A Key Observation:

- Consider the topo-orderings \( L \) and \( L' \) below, where in \( L' \) we moved \( x_j \) to the left over a block of nodes. This means none of the links \( (x_j, x_k) \) and \( (x_k, x_j) \), \( x_k \) in block-II, are present.

\[
\begin{array}{ccc}
\text{block I} & \text{block II} & \text{block III} \\
L: & \cdots & \cdots & x_k & \cdots & x_j & \cdots \\
\text{block I} & \text{block II} & \text{block III} \\
L': & \cdots & x_j & \cdots & x_k & \cdots & \cdots \\
\end{array}
\]

- \( R(L') - R(L) = w(x_j).\sum_{x_k} \text{inLessOut}(x_k) - \sum_{x_k} w(x_k).\text{inLessOut}(x_j) \)
  because \( \sum_{x_k} \text{inLessOut}(x_k) = \sum_{x_k} [\text{inWt}(x_k) - \text{outWt}(x_k)] = \sum_{x_k} [s(x_i, x_k) \text{ for } x_i \in \text{blockI} - s(x_k, x_j) \text{ for } x_j \in \text{blockIII}] \).

- Thus, \( R(L') < R(L) \) if and only if

\[
\frac{\text{inLessOut}(x_j)}{w(x_j)} > \frac{\sum \text{inLessOut}(x_k)}{\sum w(x_k)}.
\]

Question: How do you generalize the condition for \( R(L') < R(L) \) when \( x_j \) is replaced by a block of nodes?
A SIMPLE GREEDY ALGORITHM THAT DOES NOT WORK

A Greedy Algorithm: \( x_1 \) = unique source-node in acyclic \( \hat{G} \).

1. For each node, compute \( v(x) = \text{inLessOut}(x)/w(x) \).
2. Let \( S = \{ x_1 \} \), the set of current source-nodes.
3. While \( S \) not empty do the following:
   (i) Choose \( x \in S \) with the largest \( v(x) \), let \( S = S - \{ x \} \), and output \( x \).
   (ii) For each node \( y \in N^+(x) \), let \( \text{indeg}(y) = \text{indeg}(y) - 1 \) and if \( \text{indeg}(y) \) is 0 then add \( y \) to \( S \).

Note: If each \( w(x) = 1 \), then \( v(x) > v(y) \) means \( x \) has more left pull than right pull compared to \( y \) and \( x \) laid out before \( y \).

Example. Let each \( w(x) = 1 \) and each \( s(x, y) = 1 \). Replacing link \( (A_3, D) \) by \( (A_2, D) \) decreases \( v(A_2) \) by 1, increases \( v(A_3) \) by 1, but does not affect \( v(D) \) or the greedy output or the optimal topo-ordering.

Greedy Ordering: \( \langle A_1, A_2, A_3, A_4, D, B, C, E \rangle \), \( R = 16 \)
Optimal topo-ordering: \( \langle A_1, A_2, A_3, A_4, B, C, D, E \rangle \), \( R = 15 \)

Question: If we select a node \( x \in S \) that minimizes the estimate \( C_L(x) = w(x).|\{ s(x_i, y): x_i \text{ is already laid out and } y \neq x \}| = w(x).[\text{outLessIn}(\text{nodes already laid out}) - \text{inWt}(x)] \), will it work?
EXERCISE

1. Consider the acyclic digraph $\vec{G}$ below with each $w(x) = 1$.

```
\begin{center}
\begin{tikzpicture}
\node [draw] (A) at (0,0) {$A$};
\node [draw] (B) at (1,1) {$B$};
\node [draw] (C) at (1,0) {$C$};
\node [draw] (D) at (2,1) {$D$};
\node [draw] (E) at (3,0) {$E$};
\draw (A) -- (B) node [midway, above] {3};
\draw (B) -- (C) node [midway, above] {2};
\draw (B) -- (D) node [midway, above] {1};
\draw (C) -- (D) node [midway, above] {4};
\end{tikzpicture}
\end{center}
```

Show all possible topo-orderings $L$ and their $R(L)$. In particular, show that $\langle A, B, C, D, E \rangle$ is the only optimal topo-ordering and has $R = 4$. Which of the greedy algorithms (based on $v(x)$ vs. based on estimates $C(x)$) give this solution?

2. Assume all $w(x) = 1$ and all $s(x, y) = 1$. Verify that the following optimal topo-ordering is obtained by the greedy algorithm.

```
\begin{center}
\begin{tikzpicture}
\node [draw] (A) at (0,0) {$A$};
\node [draw] (B) at (1,1) {$B$};
\node [draw] (C) at (1,0) {$C$};
\node [draw] (D) at (2,1) {$D$};
\node [draw] (E) at (3,0) {$E$};
\draw (A) -- (B) node [midway, above] {1};
\draw (B) -- (C) node [midway, above] {2};
\draw (B) -- (D) node [midway, above] {3};
\draw (C) -- (D) node [midway, above] {4};
\end{tikzpicture}
\end{center}
```

Decreasing order of $\text{inLessOut}(x_i)$:
$E(3), D(1), C(0), B(-1), A(-3)$

Successive $S$ and output:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>${A}$</td>
<td>${B, C}$</td>
<td>${B}$</td>
<td>${D}$</td>
<td>${E}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A$</td>
<td>$C$</td>
<td>$B$</td>
<td>$D$</td>
<td>$E$</td>
</tr>
</tbody>
</table>

Verify that another optimal ordering is: $\langle A, B, D, C, E \rangle$.

3. Show all possible optimal ordering for the following items in a display. How do you explain the difference in the optimal orderings for $\vec{G}_1$, where we had $w(B) = 6$, $w(C) = 3$, and $w(D) = 2$?

```
\begin{center}
\begin{tikzpicture}
\node [draw] (A) at (0,0) {$A$};
\node [draw] (B) at (1,1) {$B$};
\node [draw] (C) at (1,0) {$C$};
\node [draw] (D) at (2,1) {$D$};
\node [draw] (E) at (3,0) {$E$};
\draw (A) -- (B) node [midway, above] {1};
\draw (B) -- (C) node [midway, above] {2};
\draw (B) -- (D) node [midway, above] {3};
\draw (C) -- (D) node [midway, above] {4};
\end{tikzpicture}
\end{center}
```

- each $w(x) = 1$
- each $s(x, y) = 1$
4. Assume $w(x) = 1$ and each $x(x, y) = 1$. Apply the greedy algorithm and show that it obtains an optimal topo-ordering (by showing each possible topo-ordering $L$ and the associated $R(L)$).
SOLUTION

Shown at each node is its contribution to \( R(L) \).
FINDING SYNTACTICALLY MOVABLE NODE-GROUPS

Movable Groups in $L = \langle x_1, x_2, \ldots, x_n \rangle$:

- Let $x_{j:j+p} = \{x_j, x_{j+1}, \ldots, x_{j+p}\}$; $x_{j:j} = \{x_j\}$.
- The node-group $x_{j:j+p}$ can be moved to the left immediately after $x_i$ if $\#\text{links}(x_{i+1:j-1}, x_{j:j+p}) = 0$.
- For $j > 1$, let $f(j) = \min\{i: i < j \text{ and } x_j \text{ can be moved after } x_i\}$.
  - $i = f(j)$ is the largest $i < j$ such that $(x_i, x_j) \in \overrightarrow{G}$.
  - If $(x_{j-1}, x_j) \in \overrightarrow{G}$, then $f(j) = j - 1$.
- For each $f(j) \leq k \leq j - 2$ let $p = g(k, j)$ be the largest value such that $x_{j:j+p}$ can be moved after $x_k$.
  - $p = \max\{p': \text{no link from group } x_{k+1:j-1} \text{ to group } x_{j:j+p}\}$.
  - For $k < k'$, $g(k, j) \leq g(k', j)$.
- The transitive links do not affect either of $f(j)$ and $g(k, j)$.

Example. For the $L$ and the digraph $\overrightarrow{G}$ below, we have

<table>
<thead>
<tr>
<th></th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$x_5$</th>
<th>$x_6$</th>
<th>$x_7$</th>
<th>$x_8$</th>
<th>$x_9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(j)$</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

The non-trivial earliest possible moves are: $\{x_4\}$ after $x_1$, $\{x_6, x_7\}$ after $x_4$, and $\{x_7\}$ after $x_3$. 
MOVING LARGE BLOCKS

$L$: …… ⋅⋅⋅ ⋅⋅⋅ Block $B_1$ Block $B_2$ Block $B_3$ ……

Notations:

- $a_i = \text{inLessOut}(B_i) = \sum \text{inLessOut}(x_j)$, summed over $x_j \in B_i$.
- $w_i = w(B_i) = \sum w(x_j)$, summed over $x_j \in B_i$.
- $B_{23} = B_2 \cup B_3$, with $a_{23} = a_2 + a_3$ and $w_{23} = w_2 + w_3$.

Block $B_{23}$ can be moved to the left of $B_1$ for gain in $R$:

- That is, $\frac{a_2 + a_3}{w_2 + w_3} \geq \frac{a_1}{w_1}$ and the gain in $R$ is $g_{23} = (a_2 + a_3)w_1 - a_1(w_2 + w_3) > 0$.

Two Possibilities:

1. Move $B_2$ first to the left of $B_1$ and then move $B_3$ the same way.
   - That is, $\frac{a_i}{w_i} \geq \frac{a_1}{w_1}$ for $i = 2, 3$ and strict ">" for at least one of them.
   - The successive gains in $R$ are $g_2 = (a_2w_1 - a_1w_2) \geq 0$ and $g_3 = (a_3w_1 - a_1w_3) \geq 0$, with the total gain $g_2 + g_3 = g_{23}$.

2. $B_2$ can be moved but then $B_3$ cannot be moved.
   - That is, $\frac{a_2}{w_2} > \frac{a_1}{w_1} > \frac{a_3}{w_3}$.
   - Since $g_3 < 0$, $g_{23} < g_2$ and thus it is actually better not to move $B_{23}$ as a whole.

Do not move a large block over another block unless you can move every tail of the former.
EXERCISE

1. Suppose that the blocks $B_1$, $B_2$, and $B_3$ are consecutive blocks of nodes in some topo-ordering $L$. Suppose that $B_3$ can be moved to the left $B_{12} = B_1 \cup B_2$ and also between $B_1$ and $B_2$, but $B_3$ cannot be moved to left of $B_1$ from its position between $B_1$ and $B_2$. Argue that we should only move $B_3$ between $B_1$ and $B_2$ to keep the gain in $R$ maximized. (This is the case when $\frac{a_1}{w_1} > \frac{a_3}{w_3} \geq \frac{a_1 + a_2}{w_1 + w_2} > \frac{a_2}{w_2}$.)

Do not move a block $B'$ over a large block $B$ unless you can move $B$ over every subblock of $B$.

2. Suppose that $B_3$ can be moved between $B_1$ and $B_2$ and both $B_2$ and $B_{23}$ can be moved to the left of $B_1$ to improve $R$. That is, $a_3/w_3 \geq a_2/w_2 \geq a_1/w_1$, which implies that $(a_3 + a_2)/(w_3 + w_2) \geq a_1/w_1$. Also, that there are no links between $B_3$ and any of $B_1$ and $B_2$, and there are no links between $B_1$ and $B_2$. Thus, it is legal to arrange these three blocks consecutively in the order: $B_3$, $B_2$, $B_1$ in a topo-ordering.) Then, show that after moving $B_2$, we can still move $B_3$ to the left of $B_{21}$ to improve $R$. 


A SEMI-GREEDY ALGORITHM THAT WORKS

A Semi-Greedy Algorithm With Intermittant Adjustments:
1. For each node, compute \( v(x) = \text{inLessOut}(x)/w(x) \).
2. If \( x_1 \) = unique source-node in acyclic \( \vec{G} \), then let \( S = \{x_1\} \), the set of current source-nodes.
3. While \( (S \) not empty) do the following:
   (i) Choose \( x \in S \) with the largest \( v(x) \), let \( S = S - \{x\} \), and output \( x \).
   (i+) If \( v(\text{prev}(x)) < v(x) \), where \( \text{prev}(x) \) = the node previous to \( x \) in the topo-ordering created so far, then collect as large a block of immediate previous nodes of \( x \) as possible forming a non-decreasing \( v(.) \) sequence and move it as far left as possible based on the criteria \( a_{\text{movingBlock}}/w_{\text{movingBlock}} > a_{\text{blockMovedOver}}/w_{\text{blockMovedOver}} \).
   (iii) For each node \( y \in N^+(x) \), let \( \text{indeg}(y) = \text{indeg}(y) - 1 \) and if \( \text{indeg}(y) \) is 0 then add \( y \) to \( S \).

Notes on Step (i+):
- A moving block of nodes \( \{x_i, x_{i+1}, \ldots, x_j\} \), where \( v(x_i) \leq v(x_{i+1}) \leq \cdots \leq v(x_j) \) and where one or more "\( \leq \)" are actually the strict inequality "\( < \)", must have the property that \( (x_k, x_{k+1}) \in \vec{G} \) for \( i \leq k < j \).
- There is no need to wait to see if the moving-block of nodes can be extended by adding another node \( x_{j+1} \), where \( v(x_j) \leq v(x_{j+1}) \).
FINDING THE NEXT NODE IN AN OPTIMAL TOPO-ORDERING

Notation:

- For a set of nodes $X$, $\text{inLessOut}(X) = \sum_{x \in X} \text{inLessOut}(x)$, $w(X)$
  $= \sum_{x \in X} w(x)$, and $v(X) = \text{inLessOut}(X)/w(X)$.

Necessary Condition for Next Node $x_{k+1}$:

- Let $L' = \langle x_1, x_2, \ldots, x_k \rangle$ be the initial part of an optimal topo-ordering and $X = \{x_1, x_2, \ldots, x_k\}$.
- If $Y = \{y_1, y_2, \ldots, y_m\}$ are the source-nodes when we delete $S$ from $\overrightarrow{G}$, then the optimal topo-ordering has the form

$$L'.\langle \ y_1, y_{11}, y_{12}, \ldots, y_{1n_1}, \ y_2, y_{21}, y_{22}, \ldots, y_{2n_2}, \ u_{21}, u_{22}, \ldots, u_{2q_2} \ \ldots, \ y_m, y_{m1}, y_{m2}, \ldots, y_{mn_m}, u_{m1}, u_{m2}, \ldots, u_{mq_m} \rangle$$

where the following conditions hold:

- $y_1 = x_{k+1}$;
- $Y_j = \{y_j, y_{j1}, y_{j2}, \ldots, y_{jn_j}\}$ is a path-closed subset of $E(y_j) = \{y_j, y_{j1}, y_{j2}, \ldots, y_{js_j}\}$ ($n_j \leq s_j$), the set of nodes reachable exclusively from $y_j$ and no other $y_p \in Y$; in particular, the sets $E(y_j)$ are mutually disjoint and $y_j \in E(y_j)$. (See the figure on next page.)
- The nodes $u_{ij}$ can be reached only from one or more $y_p \in Y$, $p \leq i$; $u_{ij}$ can be outside $E(y_1) \cup E(y_2) \cup \cdots E(y_{i-1})$
- For $2 \leq j \leq m$, $v(Y_1) \geq \max \{v(Y'_j): y_j \in Y'_j \subseteq Y_j\}$. 

STRUCTURE OF $\tilde{G}$ IN TERMS OF $E(y_j)$’S

- An $E(y_j)$ may not have a unique sink-node; $y_j$ its the unique source-node.
**SUFFICIENT CONDITION FOR NEXT NODE**

\[ x_{k+1} = y_1 \]

**Simplest case: \( m = 1 \).**
- Since \( Y = \{ y_1 \} \), the only choice for \( x_{k+1} = y_1 \).

**General case: \( m \geq 2 \).**
- Let \( Z \) be the nodes other than \( X \cup \bigcup_{1 \leq j \leq m} E(y_j) \).
- Let \( \tilde{G}_j \) be acyclic digraph after merging \( X \) into a single source-node \( X_0 \), merging \( Z \) into a single sink-node \( Z_0 \), and deleting the nodes in \( E(y_p) \) for \( p \neq j \).
- Since \( \tilde{G}_j \) is smaller in size than \( \tilde{G} \), we can find an optimal topordering \( L_j \) for it by induction (recursion).

\[
\begin{align*}
L_1 & : \langle X_0, y_1, y_{11}, y_{12}, \ldots, y_{1s_1}, Z_0 \rangle \\
L_2 & : \langle X_0, y_2, y_{21}, y_{22}, \ldots, y_{2s_2}, Z_0 \rangle \\
& \quad \vdots \\
L_m & : \langle X_0, y_m, y_{m1}, y_{m2}, \ldots, y_{ms_m}, Z_0 \rangle
\end{align*}
\]
- For each \( L_j \), let \( I_j : \langle y_j, y_{j1}, y_{j2}, \ldots, y_{jk_j} \rangle, 0 \leq k_j \leq s_j \), be the initial part excluding \( X_0 \) such that \( E'(y_j) = \{ y_j, y_{j1}, y_{j2}, \ldots, y_{jk_j} \} \) has the largest \( v(E'(y_j)) \) among all possible initial parts.
- Let \( v_j = v(E'(y_j)) \) and \( v_1 = \max \{ v_j : 1 \leq j \leq m \} \).
- Then, \( x_{k+1} = y_1 \).
DESIGN

Design vs. Implementation:

- Design consists of some high level decisions.
  - Each higher-level decision is "refined into", more precisely, is "accomplished by" certain lower-level decisions, as we move closer to an actual implementation into code.

Design Decision vs. Implementation Decision:

- Design decisions affect other entities and have a global impact.
  - A design decision can limit the possible implementations.
- Implementation decisions are localized; such a decision for one entity does not affect other entities.
  - An implementation may involve "lower-level" design decisions within it; we may call them implementation-designs.

Example.

- Arranging the items in an array in increasing order or as a 2-3 tree (for another function which finds, based on its input, sum, min, max, median, or kth item for any k) is a design decision.
- If finding sum, etc is not delegated to another function, then arranging the array-items in a particular way is an implementation design decision.

Example.

- If the drink-selection needs coordination with the main-item selection, then these selections are design decisions. Otherwise, they are implementation decisions.
- The decision to have "drink plus a main-item" is a design decision. (One may offer only coffee and/or candy in a gathering.)
TOP-DOWN BOTTOM-UP DESIGN

Top-Down:

- Design is basically *top-down* because one is trying to get a global structure of the software.

Bottom-Up: One can talk of bottom-up design only in the sense that one can consider a subset of the operations (a subsystem) and create an initial design.

- The initial design may have to be modified in order to extend it to a total system design.

Design for Security

Design for Efficiency

Design for Evolution

Design for User-friendliness

Design for Testability
DESIGN AS AN ABSTRACTION OF A PROGRAM

**Abstractions** (upper approximations):

- If we ignore the specific operation in an assignment-statement:

  \[ x[i] = 2 \times x[j] \times y[j] + z - 1 \]

  it becomes

  \[ x[i] = f(x[j], y[j], z) \]

  which becomes after we ignore specific components of \( x \) and \( y \)

  \[ x = f(x, y, z) \]

  That is, a \( \langle \text{def}, \text{use} \rangle \)-pair: \( \langle x, \{ x, y, z \} \rangle \).

- The abstraction of an if-then-else, with then-part and else-part already abstracted,

  \[
  \begin{align*}
  \text{if } (y > 0) & \quad \langle x, V_1 \rangle \\
  \text{else} & \quad \langle x, V_2 \rangle
  \end{align*}
  \]

  becomes \( \langle x, V_1 \cup V_2 \cup \{ y \} \rangle \).

  The abstraction of

  \[
  \begin{align*}
  \text{if } (y > 0) & \quad \langle x, V_1 \rangle \\
  \text{else} & \quad \langle y, V_2 \rangle
  \end{align*}
  \]

  becomes two use-def pairs \( \langle x, V_1 \cup \{ y \} \rangle \) and \( \langle y, V_2 \cup \{ y \} \rangle \).

Similar to the denotational-semantics of a program, may be we should have a denotational semantics of designs (expressed in some language). The denotational semantics of a design is a more abstract function than that of an implementation, the latter being a more concrete form of the former.
HOUSE FLOOR-PLAN DESIGN PROBLEM: A SERVICE-VIEW

- Each room in a house provides a service of some kind.
  - A bed-room for sleeping/resting.
  - A wash-room (toilet) provides a different kind of service than a kitchen, and thus its internal designs (to provide those services) are different.
  - For some rooms, there must be interfaces for activities outside of the house as well as the activities inside the house.

- One service of kitchen is preparation of food and storage of both cooked food and food-supplies.
  - This requires (justifies) different entry/exit points in a kitchen, i.e., interfaces of kitchen for coordination with other services:
    1. One to bring food into kitchen from the market (and taking trash out of kitchen).
    2. One to serve food in other rooms (dining room or to a bedroom when someone is sick).
    3. A direct passage between bedrooms and kitchen is not a high priority because going to a bedroom directly from the kitchen (after dinner) or coming to kitchen directly from a bedroom (in the morning) are not,

- If a room provides multiple services, then it may be possible to partition it into different areas for those services, with proper interfaces between the different areas.