Today’s Lecture

- DAG structure
- Dynamic programming to find highest weight paths in WDAGs
Structure of DAGs

• Define the *depth* of a node $v$ in $V$ as:
  – the length of the longest path ending at $v$;
  by above, the depth is well-defined and $\leq |V| - 1$.

• *Every descendant $w$ of a node $v$ has higher depth than $v$:*
  If
  – $(u, \ldots, v)$ is path of length $n = \text{depth}(v)$ ending at $v$, and
  – $(v, \ldots, w)$ is path from $v$ to $w$,
  then $(u, \ldots, v, \ldots, w)$ is a path of length $> n$ ending at $w$, so $\text{depth}(w) > n$. 
Structure of DAGs (cont’d)

• Every node $v$ of positive depth has a parent of depth exactly one less:
  – Let $(u, ..., v', v)$ be path of length $n = \text{depth}(v)$ ending at $v$.
  – Then $v'$ is a parent of $v$.
  – Since $(u, ..., v')$ has length $n - 1$, $\text{depth}(v') \geq n - 1$.
  – Since also $\text{depth}(v') < n$ (because $v$ is a descendant of $v'$),
    $\text{depth}(v')$ is exactly $n - 1$.

• The nodes on any path are of increasing depth.
Structure of DAGs (cont’d)

Depth 0

Depth 1

Depth 2

Depth 3
Important special cases:

• A (rooted) tree is a DAG which
  – has unique depth 0 node (the root), and
  – every other node has in-degree 1
    • (i.e. has a unique parent, of depth one less than that of the node).

• A binary tree is a tree in which
  – every node has out-degree at most 2.

• A linked list is a tree in which
  – every node has out-degree at most 1
  – or equivalently, a DAG in which \( \exists \) at most one node of each depth
binary tree

linked list

\[
\begin{align*}
&v_0 \\
&\quad \quad v_1 \\
&\quad \quad \quad v_3 \\
&\quad \quad \quad \quad v_6 \\
&\quad \quad \quad \quad \quad v_7 \\
&\quad \quad v_4 \\
&\quad \quad \quad v_5 \\
&\quad \quad \quad \quad v_8 \\
&v_2 \\
&\quad \quad v_1 \\
&\quad \quad \quad v_2 \\
&\quad \quad \quad \quad v_3 \\
&\quad \quad \quad \quad \quad v_4 
\end{align*}
\]
Remarks on Depth Structure

• For *dynamic programming* algorithm
  – we need an order $v_1, v_2, ..., v_n$ for the vertices
    • (not a path!)
    in which parents appear before children.
  – From the above, *depth order*
    • (in which depth 0 nodes are listed first, then depth 1 nodes, etc.)
    is such an order.
  – In general there are many other such orders.

• We haven’t given constructive procedure for finding the depths of all vertices.
  – For an arbitrary DAG, can be done in $O(|V| + |E|)$ time;
  – we omit algorithm, since for DAGs related to sequence analysis, the depth structure is obvious.
Weighted Directed Graphs

• A *weighted directed graph* is
  – a directed graph \((V, E)\) together with
  – a function \(w\) from \(E\) to the real numbers,
    • i.e. with a numerical *weight* \(w(e)\) (which may be positive, negative, or 0) associated to each edge \(e\).

A weighted DAG is called a WDAG.

• The *(sum)* *weight of a path* is defined to be the sum of the weights on the edges of the path.

• Similarly, the *product weight of a path* is the product of the edge weights
  – usually only consider this when all weights are non-negative.

• weight of a path \(P\) is written \(w(P)\)

• For a path of length 0 (i.e. consisting of a single vertex):
  – the sum weight is 0
  – the product weight is 1
Highest Weight Paths on WDAGs

**Problem:** find a path with the highest possible weight.

**Solution:**
- “Brute force” approach
  - i.e. simply enumerating all possible paths and comparing their weights
    is usually impractical (too many paths!)
- Instead, use the method of *dynamic programming* (‘The Fundamental Algorithm of Computational Biology’).
Highest Weight Paths on WDAGs (cont’d)

• Let $P_n = (v_0, v_1, \ldots, v_n)$ be a path of highest weight.
• Then for each $k < n$, the sub-path $P_k = (v_0, v_1, \ldots, v_k)$ must have highest weight of all paths ending at $v_k$, because
  – if $Q = (u_0, u_1, \ldots, v_k)$ were another path ending at $v_k$ and having higher weight than $P_k$,
  – then the path $(Q, v_{k+1}, \ldots, v_n)$ would have weight
    $$w((Q, v_{k+1}, \ldots, v_n)) = w(Q) + w((v_k, \ldots, v_n))$$
    $$> w(P_k) + w((v_k, \ldots, v_n)) = w(P_n),$$
    contradicting assumption that $P_n$ has highest weight.
Subpaths of a highest-weight path can’t be improved:

If this has highest weight of all paths ending at $v_5$ then...

this must have highest weight of all paths ending at $v_4$
So generalize the problem as follows:

- find, for each vertex \( v \), the highest weight of all paths ending at \( v \) – call this \( w(v) \)

Can find \( w(v) \) in single pass through \( V \), as follows:

- process the \( v \) in depth order (or any order in which parents precede children)
- if \( v \) has no parents, \( w(v) = 0 \) (the only path ending at \( v \) is \((v))\).
- for any other \( v \), except for the path \((v)\) (which has weight 0), any path ending at \( v \) is of form \((v_0, v_1, \ldots, v_k, u, v)\). Then
  - \( u \) is a parent of \( v \), so \( w(u) \) has already been computed, and
    \[ w((v_0, v_1, \ldots, v_k, u, v)) \leq w(u) + w((u,v)) \]
    with equality for an appropriate choice of \( v_i \).
- Therefore we may compute \( w(v) \) as

\[
  w(v) = \max(0, \max_{u \in \text{parents}(v)} (w(u) + w((u,v))))
\]
Example
$w(v) – depth 0 nodes$

Depth 0

Depth 1

Depth 2

Depth 3

Depth 4
$w(v) – depth 1 nodes$
$w(v) – \text{depth 2 nodes}$
$w(v) –$ depth 3 nodes
$w(v) - depth 4$ nodes
Highest Weight Paths on WDAGs (cont’d)

• To reconstruct best path, need “traceback” pointer to immediate predecessor of \( v \) in best path:

\[
T(v) = \begin{cases} 
  v & \text{if } w(v) = 0 \\
  \arg \max_{u \in \text{parents}(v)} (w(u) + w((u,v))) & \text{if } w(v) \neq 0 
\end{cases}
\]

– in preceding graph, \( T(v) \) is the parent on red edge coming into \( v \)
  • if more than one such edge, pick one at random;
  • if no such edge, \( T(v) = v \)

• Sometimes useful to record \textit{beginning} of best path:

\[
B(v) = \begin{cases} 
  v & \text{if } w(v) = 0 \\
  B(T(v)) & \text{if } w(v) \neq 0 
\end{cases}
\]
Highest Weight Paths on WDAGs (cont’d)

- Then highest weight of any path in graph is
  \[ \max_{v \in V} (w(v)) \]
  – updated as each node is visited
  - indicated by [] in preceding graph –
  and so doesn’t require additional pass through vertices

- if \( u = \arg\max_{v \in V} (w(v)) \), can reconstruct highest weight path by tracing back from \( u \), using \( T \):
  – path ends at \( u \);
  – immediate predecessor of \( u \) is \( T(u) \);
  – predecessor of \( T(u) \) is \( T(T(u)) \); etc.
  – stop when \( T(v) = v \).

- In preceding example, highest weight is 6 and \( u = v_{11} \)
Dynamic programming on WDAGs

Depth 0

Depth 1

Depth 2

Depth 3

Depth 4
Complexity of Dynamic Programming

- Time to find a best path is $O(|E| + |V|)$:
  - in initial pass, visit each node, and each edge into that node: $O(|E| + |V|)$
  - in traceback, visit subset of nodes, and unique edge from each node: $O(|V|)$

(Complexity to find all highest weight paths can be higher)

For very large graphs, even $O(|E| + |V|)$ may be unacceptable!
• Space requirements:
  – If only want weight of best path, and beginning and end, then
    – don’t need $T(v)$, and
    – only need retain $w(v)$ and $B(v)$ until have processed all children of $v$ (or when best path found so far ends at $v$).

  Space depends on graph structure, but usually $<< O(|V|)$.

  – If want path itself, must store $T(v)$ $\forall v$
    – space $= O(|V|)$
    – $\exists$ algorithms (for some graphs) to reduce this, but may take more time.