CS2010 – Data Structures and Algorithms II

Lecture 10 – Algorithms on DAG

stevenhalim@gmail.com
Outline

(Dynamic Programming) Algorithms on DAG

• SSSP on DAG Revisited
  – A gentle introduction to Dynamic Programming (DP) technique
  • Optimal sub structure
  • Overlapping sub problems

• SS Longest Paths (SSLP) on DAG
  – SSLP on DAG → Longest Increasing Subsequence (LIS)

• Counting Paths on DAG

Reference: CP3 Section 3.5 (only parts of this big section) & Section 4.7.1
SSSP in DAG

Still not yet implemented in http://visualgo.net/sssp.html :(

One Topological Sort of this DAG is \{0, 2, 1, 3, 4, 5\}

- Try relaxing the outgoing edges of the vertices listed in the toposort above
  - With just one pass, all vertex will have the correct D[v]
One Topological Sort of this DAG is {0, 2, 1, 3, 4, 5}
• Start from source (vertex 0)
One Topological Sort of this DAG is \{0, 2, 1, 3, 4, 5\}:

- Continue with vertex 2
One Topological Sort of this DAG is \{0, 2, 1, 3, 4, 5\}
• Then vertex 1
One Topological Sort of this DAG is \{0, 2, 1, 3, 4, 5\}

- Then vertex 3; Vertices that have been processed so far, i.e. \{0, 2, 1, 3\} already have the correct final shortest path values, we do not have to re-trace our steps \rightarrow good performance!
Review: SSSP in DAG (5)

One Topological Sort of this DAG is \( \{0, 2, 1, 3, 4, 5\} \)
- Q: Almost done (or is it “already done?”)
One Topological Sort of this DAG is \{0, 2, 1, 3, 4, 5\}

- Final state
- The thick red edges form the shortest paths spanning tree
Analysis of SSSP on DAG

1. Pre-processing step: Topological sort
   - This can be done with $O(V+E)$ modified DFS as in Lecture 06
     (try TopoSort animation @ http://visualgo.net/dfsbsfs.html)

2. Then, following this topological order ($V$ items),
   relax a total of $E$ edges
   - The total number of outgoing edges from all vertices is $E$
   - So again, it is $O(V+E)$

Thus, SSSP on DAG can be solved in **linear time**: $O(V+E)$
- Linear in terms of $V$ and $E \to 1$ pass of all $V$ vertices and $E$ edges
- Another name: “One-Pass Bellman Ford’s”
Why It Works? (1)

On general graph, Bellman Ford’s algorithm repeats this all-edges $O(E)$ relaxation $V-1$ times, thus $O(VE)$.

- After $\text{relax}(u, v, w_{u,v})$ is performed, there may be other better path \textit{in the future} that reaches vertex $u$ (the origin) so that this $\text{relax}(u, v, w_{u,v})$ has to be repeated...
  - We can only \textit{be sure} after we have done this all-edges relaxation $V-1$ times (recall the proof of correctness of Bellman Ford’s).

Let edge ordering be:

![Graph diagram]

You will repeat $\text{relax}(1,3,3)$ and $\text{relax}(3,4,6)$ twice; note that this graph is not a DAG as $1 \rightarrow 2 \rightarrow 1$ is a trivial cycle.
Why It Works? (2)

On DAG, there is **no cycle** $\rightarrow$ we have topological order

- Recall the meaning of topological order:
  - Linear ordering of vertices such that for every edge $(u, v)$ in DAG, vertex $u$ comes before $v$ in the ordering.

One Topological Sort of this DAG is $\{0, 2, 1, 3, 4, 5\}$.
Why It Works? (3)

• If the vertices are processed according to topological order and u is the next vertex, then after $\text{relax}(u, v, w_{u,v})$ is performed, there will never be any better other path in the future that reaches vertex u so that this $\text{relax}(u, v, w_{u,v})$ has to be repeated...
  – There is no way vertex v can reach back to vertex u because vertex v appears later in the topological ordering
    • There is no cycle that allows v $\rightarrow$ some other vertices (with –ve path weight) $\rightarrow$ u
    – That means $\text{dist}[u] = \delta(s, u)$ and we can safely propagate this final shortest path value to its neighbor v

• Thus SSSP on DAG can be solved in $O(V+E)$ time
  – We do not have to repeat this V-1 times 😊
Where is the Recursion/DP? (Part 1)

Observe, for example, shortest path $0 \rightarrow 2 \rightarrow 4 \rightarrow 5$

- Sub paths of this path (e.g. $0 \rightarrow 2 \rightarrow 4$) are shortest paths too!
- We do **not** re-compute these (clearly) overlapping sub paths
  - Topological order is the correct order to avoid re-computations
  - This is called **“bottom-up” DP**: From known base case (distance to source is 0, compute the distance to other vertices using the topological order of DAG)
SS LONGEST PATHS ON DAG

Sorry, still not yet implemented in http://visualgo.net/sssp.html :(
Program Evaluation and Review Technique (PERT)

• PERT is a project management technique
• It involves breaking a large project into several tasks, estimating the time required to perform each task, and determining which tasks can not be started until others have been completed
  – This is similar to module pre-requisites!
  – This is a DAG!
• The project is then summarized in chart form
• See the next few slides for an example
Longest Paths on DAG (2)

Problem source: **UVa 452 – Project Scheduling**

- Verify that this graph is a DAG!
  - The weight is **on vertices**, e.g. weight(0) = 5 (see 2 slides later)
- The shortest way to complete this project is the...
  - **longest path** found in the DAG... (a bit counter intuitive)

Notice the -∞
First, find one topological order: \{0, 3, 1, 2, 4, 5\} 
- Can be found with \(O(V+E)\) modified DFS as in Lecture 06

Then “stretch” (antonym of “relax”) the outgoing edges of the vertices listed in this topological order.
Review: “Relax” Operation

relax(u, v, w_{u,v})

if D[v] > D[u]+w_{u,v} // if SP can be shortened
   D[v] \leftarrow D[u]+w_{u,v} // relax this edge
   p[v] \leftarrow u // remember/update the predecessor
"Stretch" Operation

```
stretch(u, v, w_u_v)
if D[v] < D[u]+w_u_v  // if LP can be lengthened
    D[v] ← D[u]+w_u_v  // stretch this edge
    p[v] ← u  // remember/update the predecessor
```
Longest Paths on DAG (4)

Dealing with vertex weight issue

- Initially, we set \( \text{dist}[0] = \text{weight}(0) = 5 \)
- Then, we use the weight of destination vertex as the “edge weight” of that edge, i.e. we transform the graph like this

![Graph Diagram]

1. From vertex 0 to vertex 1, the weight is 3.
2. From vertex 1 to vertex 2, the weight is 2.
3. From vertex 2 to vertex 5, the weight is 2.
4. From vertex 3 to vertex 4, the weight is 4.
5. From vertex 4 to vertex 5, the weight is 2.
6. From vertex 0 to vertex 3, the weight is 2.
Longest Paths on DAG (5)

One topological order: \{0, 3, 1, 2, 4, 5\}

• But we can use \texttt{stretch(0, 1, weight(1))}, \texttt{stretch(0, 3, weight(3))}
  instead of physically altering the graph, like below when we stretch outgoing edges from vertex 0
Longest Paths on DAG (6)

One topological order: {0, 3, 1, 2, 4, 5}
• Continue with vertex 3
Longest Paths on DAG (7)

One topological order: \{0, 3, 1, 2, 4, 5\}
- Continue with vertex 1
Longest Paths on DAG (8)

One topological order: \{0, 3, 1, 2, 4, 5\}

- Continue with vertex 2
  - Notice (again) that the vertices that have been processed so far, i.e. \{0, 3, 1, 2\} already have the correct final longest path values 😊, we do not have to re-trace our steps
Longest Paths on DAG (9)

One topological order: \(0, 3, 1, 2, 4, 5\)

- Same Q: Can we stop here? i.e. the second last vertex?
Longest Paths on DAG (10)

One topological order: \{0, 3, 1, 2, 4, 5\}

• Final solution, again the thick red edges are the LP Sp Tree
  – Scan the whole \text{dist}[v]$, find the largest one
    • In this example \text{dist}[5] = 16 is the largest
    • Use predecessor information (the thick red edges) to reconstruct the longest path: 0 \rightarrow 1 \rightarrow 2 \rightarrow 4 \rightarrow 5
Where is the Recursion/DP? (Part 2)

These two major ingredients for DP technique are also present in the SSLP on DAG problem:

1. Optimal sub-structure
   - Sub paths of longest paths **on DAG** are longest paths too

2. Overlapping sub-problem
   - Longest path 0->1->2->4->5 contains longest path 0->1->2->4, etc

Again, topological order enable us to avoid re-computations: Bottom-up DP
Analysis of Longest Paths on DAG
(The same as SSSP on DAG)

1. Pre-processing step: topological sort
   – This can be done in $O(V+E)$ using modified DFS

2. Then, following this topological order ($V$ items),
   “stretch” a total of $E$ edges
   – Again, it is $O(V+E)$

In overall, longest paths on DAG can be solved
in linear time: $O(V+E)$

• Linear in terms of $V$ and $E$
• Later in CS3230, you will learn that the longest paths problem
  is NP-Complete (i.e. ‘very hard’)

Longest Paths $\leftrightarrow \rightarrow$ LIS :O

There is one more classical CS problem that can be modelled as longest paths in (implicit) DAG

• The Longest Increasing Subsequence (LIS)

While we are at this topic, let’s discuss it as well 😊

• In the next few slides, we will see LIS, the implicit DAG in LIS, and the solution
A sister problem that is very related to SSLP on DAG

(actually in http://visualgo.net/recursion.html, alternative view)

LONGEST INCREASING SUBSEQUENCE (LIS)
Longest Increasing Subsequence (1)

Problem Description (Abbreviated as LIS):

• As implied by its name....
  Given a sequence \( \{A[0], A[1], ..., A[N-1]\} \) of length \( N \),
determine the Longest Increasing Subsequence
  – Subsequence is not necessarily contiguously
• Example: \( N = 8 \), sequence \( A = \{-7, 10, 9, 2, 3, 8, 8, 1\} \)
  – LIS is \( \{-7, 2, 3, 8\} \) of length 4
• Variants:
  – Longest Decreasing Subsequence
  – Longest Non Decreasing\(^\wedge\) Subsequence
Longest Increasing Subsequence (2)

There is an **implicit DAG** in this sequence A

- See the implicit DAG of sequence A = \{-7, 10, 9, 2, 3, 8, 8, 1, \infty\}
  
  - We do not have to store implicit graph in a graph DS

Note: The edges from -7 to every other vertices except edge (-7, 2) and the edges all vertices to dummy target vertex t except edge (8, t) are not shown to avoid cluttering this picture.
Longest Increasing Subsequence (3)

Let $D[i] = \text{the best LIS ending at index (vertex) } i$

The topological order is obviously $\{0, 1, 2, \ldots, N\}$, \textbf{Q: Why?}

- “Stretch” all index (vertex) one by one using this order

  
  
  $D[0] = 0$ // base case
  
  for $i = 0$ to $N-1$ // this is $O(N^2)$ Bottom-Up DP
    
    for $j = i+1$ to $N$
      
        
        stretch($i, j, 1$) // edge weight is 1
      
      report $D[N]$ // the answer is here
Longest Increasing Subsequence (4)

<table>
<thead>
<tr>
<th>Index</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>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>-7</td>
<td>10</td>
<td>9</td>
<td>2</td>
<td>3</td>
<td>8</td>
<td>8</td>
<td>1</td>
<td>∞</td>
</tr>
<tr>
<td>D (initial)</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>D (i = 0)</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>D (i = 1-2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>no change during these two iterations</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D (i = 3)</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>D (i = 4)</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>D (i = 5)</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>D (i = 6-8)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>no more change</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This LIS problem can be solved in $O(n^2)$, analysis:

1. Use the fact that there are two nested loops of size $n$, or
2. Use the analysis of the longest paths on (implicit) DAG where there are $V = n$ vertices and $E = n^2$ edges. Q: why?
Where is the Recursion/DP? (Part 3)

This LIS problem is more naturally solved in “Top-Down Dynamic Programming (DP)” fashion

• Let \( \text{LIS}(i) \) be the value of the longest LIS starting from index \( i \) until \( N-1 \)
  – This can be written as a function with one parameter, index \( i \)

• We can write the solution using this recurrence relations:
  – \( \text{LIS}(N-1) = 1 \) // at last position, we cannot extend the LIS anymore
  – \( \text{LIS}(i) = \max(\text{LIS}(j)+1), \forall j \in [i+1 .. N-1] \) where \( A[i] < A[j] \)

• To avoid recomputations, memoize the LIS value of each index/vertex \( i \)
  – This term “memoize” (memo table) will be explained soon
Visually

LIS(i) is the **recursive case**, we want to find the largest LIS(j) for all \( j \in [i+1..N-1] \) plus one but only if \( A[i] < A[j] \)

LIS(\( j_1 \)) is the longest IS length from index \( j_1 \) to \( N-1 \)

LIS(\( j_2 \)) is the longest IS length from index \( j_2 \) to \( N-1 \)

LIS(N-1) = 1

Base case, at least \( A[N-1] \) itself

We pick the maximum among all values of LIS(j) where \( A[i] < A[j] \) and \( j \in [i+1..N-1] \)

This maximum plus one is the answer for LIS(i) and we do this recursively for the other \( i \in [0..N-2] \)
Where is the Recursion/DP? (Part 4)

This can be written using (Java) recursive function

• Notice that this version is very slow due to recomputations

```java
private static int LIS(int i) {
    if (i == N-1) return 1;

    int ans = 1; // at least A[i] itself
    for (int j = i+1; j < N; j++)
        if (A[i] < A[j])
            ans = Math.max(ans, LIS(j)+1);
    return ans;
}
```
Turn Recursion into Memoization

*Key Point: Space-Time Tradeoff*

initialize memo table in the main method

```java
return_value recursive_function(state) {
    if state already calculated, simply return its value
    calculate the value of this state using recursion
    save the value of this state in the memo table
    return the value
}
```
A much better version (see LISDPDemo.java):

```java
private static int LIS(int i) {
    if (i == N-1) return 1;
    if (memo.get(i) != -1) return memo.get(i);

    int ans = 1; // at least A[i] itself
    for (int j = i+1; j < N; j++)
        if (A[i] < A[j])
            ans = Math.max(ans, LIS(j)+1);
    memo.set(i, ans);
    return ans;
}
// values in memo are set to -1 in main method
```
Final discussion for today, again about DAG 😊
(also not in VisuAlgo yet, maybe later, together with SSSP on DAG)

COUNTING PATHS ON DAG
Counting Paths on DAG

Given some real-life time line (obviously a DAG, Q: Why?)

• How many different possible lives that you can live (from birth/vertex 0 to death/vertex 8)?

Answer = 6
Find the toposort first: \{0, 1, 2, 3, 4, 6, 5, 7, 8\}

- \textit{numPaths}[0] = 1, propagate to vertex 1, and then 2, 4
Toposort/Graph/Bottom-Up Way (2)

Find the toposort first: \{0, 1, 2, 3, 4, 6, 5, 7, 8\}
• numPaths[2] = 1, propagate to vertex 3 and 4
Find the toposort first: \{0, 1, 2, 3, 4, 6, 5, 7, 8\}
- \text{numPaths}[3] = 1, propagate to vertex 4
Find the toposort first: \{0, 1, 2, 3, 4, 6, 5, 7, 8\}
- numPaths[4] = 3, propagate to vertex 5 and 6
Toposort/Graph/Bottom-Up Way (5)

Find the toposort first: \{0, 1, 2, 3, 4, 6, 5, 7, 8\}

• >> >> Fast forward..., this is the final state

The toposort:
\{0, 1, 2, 3, 4, 6, 5, 7, 8\}
Where is the Recursion/DP? (Part 6)

We can solve “counting paths in DAG” with Top-Down DP

• That is: Using functions, parameters, and “memo table”
• Let $\text{numPaths}(i)$ be the number of paths starting from vertex $i$ to destination $t$
• We can write the solution using this recurrence relations:
  – $\text{numPaths}(t) = 1$ // at destination $t$, obviously only one path
  – $\text{numPaths}(i) = \sum \text{numPaths}(j)$, for all $j$ adjacent to $i$
• To avoid recomputations, memoize the number of paths for each vertex $i$
• Only brief code is shown in the next slide
  – The overall code is similar to the LISDPDemo.java shown earlier
private static int numPaths(int i) {
    if (i == V-1) return 1;
    if (memo.get(i) != -1) return memo.get(i);
    int ans = 0;
    for (int j = 0; j < AdjList.get(i).size(); j++)
        ans += numPaths(AdjList.get(i).get(j).first());
    memo.set(i, ans);
    return ans;
}
The way the answer is computed is now from destination to source.

Q: What happen if you do not use memoization?
Transition to DP Topics

In this lecture, we link graph topic (DAG) to DP

- SSSP on DAG (revisited)
- SSLP on DAG $\leftarrow \rightarrow$ LIS
- Counting Paths on DAG
- We show both “graph way” (algorithms on DAG/also known as Bottom-Up DP) and recursive way (also known as Top-Down DP)
- DP Ingredients:
  - Optimal sub-structure and Overlapping sub-problem
- Mapping the Terminologies:
  - Vertices $\rightarrow$ States; Edges $\rightarrow$ Transitions
  - $|V| \rightarrow$ Space Complexity; $|E| \rightarrow$ Time Complexity
Plan

In the next 2 lectures (11-12), 3 tutorials (09-10-optional 11), and 1 PS (6), we will use more implicit DAGs (on more structured problems) and use more DP terminologies rather than graph terminologies 😊