CSE 421

Dynamic Programming
RNA, Sequence Alignment

Yin Tat Lee
Edit Distance

Cost = # of gaps + #mismatches.

Applications.
- Basis for Unix diff and Word correct in editors.
- Speech recognition.
- Computational biology.

Cost: 5

Cost: 3
DP for Sequence Alignment

Let $OPT(i, j)$ be min cost of aligning $x_1, ..., x_i$ and $y_1, ..., y_j$

**Case 1**: OPT matches $x_i, y_j$
- Then, pay mis-match cost if $x_i \neq y_j$ + min cost of aligning $x_1, ..., x_{i-1}$ and $y_1, ..., y_{j-1}$ i.e., $OPT(i - 1, j - 1)$

**Case 2**: OPT leaves $x_i$ unmatched
- Then, pay gap cost for $x_i + OPT(i - 1, j)$

**Case 3**: OPT leaves $y_j$ unmatched
- Then, pay gap cost for $y_j + OPT(i, j - 1)$
Sequence-Alignment(m, n, x₁x₂...xₘ, y₁y₂...yₙ) {
  for i = 0 to m
      M[0, i] = i
  for j = 0 to n
      M[j, 0] = j

  for i = 1 to m
      for j = 1 to n
          M[i, j] = min( (xᵢ=yⱼ ? 0:1) + M[i-1, j-1],
                          1 + M[i-1, j],
                          1 + M[i, j-1])
  return M[m, n]
}

Analysis: \( \Theta(mn) \) time and space.
Computational biology: m = n = 1,000,000. 1000 billions ops OK, but 1TB array?
Shortest Path

$M[i, j] = \min( (x_i=y_j ? 0:1) + M[i-1, j-1], 1 + M[i-1, j], 1 + M[i, j-1])$

Edit distance is the distance between $(0,0)$ and $(m, n)$ of the following graph.

- All horizontal edges has cost 1.
- All vertical edges has cost 1.
- The cost of edges from $(i - 1, j - 1)$ to $(i, j)$ is $1_{x_i \neq y_j}$

The graph is a DAG.

Question:
How to recover the alignment (or how to find the shortest path) without using $mn$ space?

Figure 6.17 A graph-based picture of sequence alignment.
How to recover the alignment?

Idea 1: Suffices to find the point a shortest path pass on the middle row.

Why?
Divide and Conquer!

Idea 2: $d_{(0,0)\rightarrow(m,n)} = \min_j d_{(0,0)\rightarrow(m/2,j)} + d_{(m/2,j)\rightarrow(m,n)}$

```plaintext
Find(i_1, j_1, i_2, j_2) { // Due to spacing, ignored boundary cases
    Let k = [(i_1 + i_2)/2]
    Compute $d_{(i_1,j_1)\rightarrow(k,j_2)}$ via Dijkstra at $(i_1,j_1)$.
    Compute $d_{(k,j)\rightarrow(i_2,j_2)}$ via Dijkstra at $(i_2,j_2)$ on reversed graph.
    Let k = argmin_k $d_{(i_1,j_1)\rightarrow(k,j_2)} + d_{(k,j_2)\rightarrow(i_2,j_2)}$
    p_1 = Find(i_1, j_1, k, j)
    p_2 = Find(k, j, i_2, j_2)

    return p_1, p_2
}
```
Lesson

Advantage of a bottom-up DP:
It is much easier to optimize the space.

By the way, edit distance
• can be computed in $O(s \times \min(m, n))$ if edit distance $\leq s$
• can be computed in $O\left(\frac{n^2}{\log^2 n}\right)$ (1980).
• can be approximated by log factor in $O(n^{1+\varepsilon})$ (~2010).
• cannot be solved in $O(n^{2-\delta})$ exactly (2015).
• can be approximated by O(1) factor in $O(n^{2-2/7})$ (~2018).
• can be approximated by O(1) factor in $O(n^{1+\varepsilon})$ (~2020).
Longest Path in a DAG
Longest Path in a DAG

**Goal:** Given a DAG $G$, find the longest path.

**Recall:** A directed graph $G$ is a DAG if it has no cycle.

This problem is NP-hard for general directed graphs:
- It has the Hamiltonian Path as a special case
Q: What is the right ordering?
Remember, we have to use that G is a DAG, ideally in defining the ordering

We saw that every DAG has a topological sorting
So, let’s use that as an ordering.
Suppose we have labelled the vertices such that \((i, j)\) is a directed edge only if \(i < j\).

Let \(OPT(j)\) = length of the longest path ending at \(j\) Suppose \(OPT(j)\) is \((i_1, i_2), (i_2, i_3), \ldots, (i_{k-1}, i_k), (i_k, j)\), then

\textbf{Obs 1:} \(i_1 \leq i_2 \leq \ldots \leq i_k \leq j\).

\textbf{Obs 2:} \((i_1, i_2), (i_2, i_3), \ldots, (i_{k-1}, i_k)\) is the longest path ending at \(i_k\).

\[ OPT(j) = 1 + OPT(i_k). \]
Suppose we have labelled the vertices such that \((i, j)\) is a directed edge only if \(i < j\).

Let \(OPT(j) = \text{length of the longest path ending at } j\)

\[
OPT(j) = \begin{cases} 
0 & \text{If } j \text{ is a source} \\
1 + \max_{i: (i, j) \text{ an edge}} OPT(i) & \text{o.w.}
\end{cases}
\]
Outputting the Longest Path

Let $G$ be a DAG given with a topological sorting:
For all edges $(i, j)$ we have $i < j$.

Initialize $\text{Parent}[j] = -1$ for all $j$.

\begin{verbatim}
Compute-OPT(j) {
    if (in-degree(j) == 0)
        return 0
    if (M[j] == empty)
        M[j] = 0;
    for all edges (i, j)
        if (M[j] < 1 + Compute-OPT(i))
            M[j] = 1 + Compute-OPT(i)
            Parent[j] = i
    return M[j]
}
\end{verbatim}

Let $k$ be the maximizer of $\text{Compute-OPT}(1), ..., \text{Compute-OPT}(n)$

While ($\text{Parent}[k] != -1$)
    Print $k$
    $k = \text{Parent}[k]$

Record the entry that we used to compute $\text{OPT}(j)$
Exercise:
Longest Increasing Subsequence
Longest Increasing Subsequence

Given a sequence of numbers
Find the longest increasing subsequence in $O(n^2)$ time

$41, 22, 9, 15, 23, 39, 21, 56, 24, 34, 59, 23, 60, 39, 87, 23, 90$

$41, 22, 9, 15, 23, 39, 21, 56, 24, 34, 59, 23, 60, 39, 87, 23, 90$
Find the longest increasing subsequence in $O(n^2)$ time.

I can do it in $O(n \log n)$
DP for LIS

Let $OPT(j)$ be the longest increasing subsequence ending at $j$.

**Observation**: Suppose the $OPT(j)$ is the sequence $x_{i_1}, x_{i_2}, \ldots, x_{i_k}, x_j$

Then, $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$ is the longest increasing subsequence ending at $x_{i_k}$, i.e., $OPT(j) = 1 + OPT(i_k)$

$$OPT(j) = \begin{cases} 
1 & \text{If } x_j < x_i \text{ for all } i < j \\
1 + \max_{i : x_i < x_j} OPT(i) & \text{o.w.} 
\end{cases}$$

**Alternative Soln**: This is a special case of Longest path in a DAG: Construct a graph 1,...,n where $(i, j)$ is an edge if $i < j$ and $x_i < x_j$. How to make it faster?
**Data structure for LIS**

We need a data structure with following operations:

- **Initialize()**: Set \(x_1, x_2, \ldots, x_n\) to 0 in \(O(n)\) time.
- **Set\((j, v)\)**: Set \(x_j\) to \(v\) in \(O(\log n)\) time.
- **Max\((a, b)\)**: Output \(\max_{a \leq j \leq b} x_j\) in \(O(\log n)\) time.

\[
OPT(j) = \begin{cases} 
1 \\
1 + \max_{i : x_i < x_j} OPT(i)
\end{cases}
\]

If \(x_j < x_i\) for all \(i < j\) o.w.
Shortest Paths with Negative Edge Weights
Shortest Paths with Neg Edge Weights

Given a weighted directed graph $G = (V, E)$ and a source vertex $s$, where the weight of edge $(u,v)$ is $c_{u,v}$ (that can be negative).

**Goal:** Find the shortest path from $s$ to all vertices of $G$.

Recall that Dijkstra’s Algorithm fails when weights are negative.

Why distance can be negative? Think distance as cost instead.
Impossibility on Graphs with Neg Cycles

**Condition:** No solution exists if G has a negative cycle.

This is because we can minimize the length by going over the cycle again and again.

So, suppose G does not have a negative cycle.
**DP for Shortest Path (First Attempt)**

**Def:** Let $OPT(v)$ be the length of the shortest $s - v$ path

$$OPT(v) = \begin{cases} 
0 & \text{if } v = s \\
\min_{u: (u,v) \text{ an edge}} OPT(u) + c_{u,v} & \text{otherwise}
\end{cases}$$

The formula is correct. But it is not clear how to compute it.
DP for Shortest Path

**Def:** Let $OPT(v, i)$ be the length of the shortest $s - v$ path with at most $i$ edges.

Let us characterize $OPT(v, i)$.

**Case 1:** $OPT(v, i)$ path has less than $i$ edges.
- Then, $OPT(v, i) = OPT(v, i - 1)$.

**Case 2:** $OPT(v, i)$ path has exactly $i$ edges.
- Let $s, v_1, v_2, ..., v_{i-1}, v$ be the $OPT(v, i)$ path with $i$ edges.
- Then, $s, v_1, ..., v_{i-1}$ must be the shortest $s - v_{i-1}$ path with at most $i - 1$ edges. So,

  $$OPT(v, i) = OPT(v_{i-1}, i - 1) + c_{v_{i-1}, v}$$
Def: Let $OPT(v, i)$ be the length of the shortest $s - v$ path with at most $i$ edges.

$$OPT(v, i) = \begin{cases} 
0 & \text{if } v = s \\
\infty & \text{if } v \neq s, i = 0 \\
\min(OPT(v, i - 1), \min_{u:(u,v) \text{ an edge}} OPT(u, i - 1) + c_{u,v}) & \text{otherwise}
\end{cases}$$

So, for every $v$, $OPT(v, ?)$ is the shortest path from $s$ to $v$.

But how long do we have to run?

Since $G$ has no negative cycle, it has at most $n - 1$ edges. So, $OPT(v, n - 1)$ is the answer.
Bellman Ford Algorithm

\[\begin{align*}
\text{for } v = 1 \text{ to } n \\
\quad \text{if } v \neq s \text{ then} \\
\quad \quad M[v,0] = \infty \\
\quad M[s,0] = 0.
\end{align*}\]

\[\begin{align*}
\text{for } i = 1 \text{ to } n-1 \\
\quad \text{for } v = 1 \text{ to } n \\
\quad \quad M[v,i] = M[v,i-1] \\
\quad \quad \text{for every edge } (u,v) \\
\quad \quad \quad M[v,i] = \min(M[v,i], M[u,i-1] + c_{u,v})
\end{align*}\]

Running Time: \(O(nm)\)

Can we test if G has negative cycles?
Yes, run for \(i = 1 \ldots 3n\) and see if the \(M[v,n-1]\) is different from \(M[v,3n]\)
Exercise:
Minimum Vertex Cover for Tree
Minimum Vertex Cover for Tree

Given an undirected tree \( T = (V,E) \).

We call \( S \subset V \) is a vertex cover if every edge touches some vertex in \( S \).

Give a linear time algorithm to find the minimum vertex cover of tree.

Answer:
Let \( F(v) \) be the size of minimum vertex cover of the subtree at \( v \).
Then
\[
F(v) = \min(\#children(v) + \sum_{g: \text{grandchild of } v} F(g), 1 + \sum_{c: \text{child of } v} F(c))
\]