CSE 421

Longest Path in DAG
Longest Increasing Subsequence

Shayan Oveis Gharan
Sequence Alignment
Given two strings $x_1, ..., x_m$ and $y_1, ..., y_n$ find an alignment with minimum number of mismatch and gaps.

An alignment is a set of ordered pairs $(x_{i_1}, y_{j_1}), (x_{i_2}, y_{j_2}), ...$ such that $i_1 < i_2 < ...$ and $j_1 < j_2 < ...$

**Example:** CTACCG vs. TACATG. 
Sol: We aligned $x_2$-$y_1$, $x_3$-$y_2$, $x_4$-$y_3$, $x_5$-$y_4$, $x_6$-$y_6$.

So, the cost is 3.
DP for Sequence Alignment

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

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

Sequence-Alignment(m, n, x_1 x_2 ... x_m, y_1 y_2 ... y_n) {
    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( (x_i = y_j ? 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.

English words or sentences: $m, n \leq 10,..,20.$

Computational biology: $m = n = 100,000.$ 10 billions ops OK, but 40GB array?
If we are not using strong induction in the DP, we just need to use the last (row) of computed values.

```plaintext
Sequence-Alignment(m, n, x_1x_2...x_m, y_1y_2...y_n) {
    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( (x_i=y_j ? 0:1) + M[i-1, j-1],
                           1 + M[i-1, j],
                           1 + M[i, j-1])

    return M[m, n]
}
```

Just need $i - 1, i$ rows to compute $M[i,j]$
DP with $O(m + n)$ memory

- Keep an Old array containing values of the last row
- Fill out the new values in a New array
- Copy new to old at the end of the loop

```java
Sequence-Alignment(m, n, x1x2...xm, y1y2...yn) {
    for i = 0 to m
        O[i] = i
    for i = 1 to m
        N[0] = i
    for j = 1 to n
        N[j] = min( (x_i=y_j ? 0:1) + O[j-1],
                    1 + O[j],
                    1 + N[j-1])
    for j = 1 to n
        O[j] = N[j]
    return N[n]
}
```
Lesson

Advantage of a bottom-up DP:

It is much easier to optimize the space.

History:

• [Backurs, Indyk’14]: If edit distance of two strings of length $n$ can be computed in $O(n^{1.99})$ then Satisfiability on $n$ variables problem can be solved in $2^{0.9999999n}$

• [Andoni-Nosatzki’21] There is a constant factor approximation algorithm for edit distance that runs in time $n^{1.01}$
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.
DP for Longest Path in a DAG

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 in the longest path ending at \(j\), last edge is \((i, j)\).

Then, none of the \(i + 1, \ldots, j - 1\) are in this path since topological ordering. Furthermore the path ending at \(i\) must be the longest path ending at \(i\),

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

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

\[
OPT(j) = \begin{cases} 
0 & \text{If } j \text{ is a source} \\
1 + \max_{i:(i,j) \text{ an edge}} OPT(i) & \text{o.w.}
\end{cases}
\]
Let G be a DAG given with a topological sorting: For all edges (i,j) we have i<j.

```cpp
Compute-OPT(j) {
    if (in-degree(j) == 0)
        return 0
    if (M[j] == empty)
        M[j] = 0;
    for all edges (i,j)
        M[j] = max(M[j], 1 + Compute-OPT(i))
    return M[j]
}
```

Output \(\max(M[1], \ldots, M[n])\)

**Running Time:** \(O(n + m)\)

**Memory:** \(O(n)\)

Can we output the longest path?
Let G be a DAG given with a topological sorting: For all edges (i,j) we have i<j.

Initialize Parent[j]=-1 for all j.

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
        record the entry that we used to compute OPT(j)
    return M[j]
}

Let M[k] be the maximum of M[1],...,M[n]

While (Parent[k]!=-1)
    print k
    k=Parent[k]
Longest Increasing Subsequence
Longest Increasing Subsequence

Given a sequence of numbers
Find the longest increasing subsequence

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
**DP for LIS**

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

**Observation**: Suppose the $OPT(j)$ is the sequence

$$x_{i_1}, x_{i_2}, ..., x_{i_k}, x_j$$

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

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

**Remark**: 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 $x_i < x_j$. 
DP Techniques Summary

Recipe:

• Follow the natural induction proof.
• Find out additional assumptions/variables/subproblems that you need to do the induction
• Strengthen the hypothesis and define w.r.t. new subproblems

Dynamic programming techniques.

• Whenever a problem is a special case of an NP-hard problem an ordering is important:
• Adding a new variable: knapsack.
• Dynamic programming over intervals: RNA secondary structure.

Top-down vs. bottom-up:

• Different people have different intuitions
• Bottom-up is useful to optimize the memory
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 \( c_{u,v} \)

Goal: Find the shortest path from \( s \) to all vertices of \( G \).

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

**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 $s, v_1, v_2, \ldots, v_{i-1}, v$ be the $OPT(v, i)$ path with $i$ edges.
- Then, $s, v_1, \ldots, v_{i-1}$ must be the shortest $s - v_{i-1}$ path with at most $i - 1$ edges. So,
  $$OPT(v, i) = OPT(v_{i-1}, i - 1) + c_{v_{i-1}, v}$$
DP for Shortest Path

**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{verbatim}
for v=1 to n
    if v \neq s then
        M[v,0]=\infty
    M[s,0]=0.

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

**Running Time:** $O(nm)$

Can we test if G has negative cycles?
Bellman Ford Algorithm

```python
for v=1 to n
    if v ≠ s then
        M[v,0]=∞
M[s,0]=0.

for i=1 to n-1
    for v=1 to n
        M[v,i]=M[v,i-1]
    for every edge (u,v)
        M[v,i]=min(M[v,i], M[u,i-1]+c_u,v)
```

**Running Time:** $O(nm)$

Can we test if $G$ has negative cycles?
Yes, run for $i=1...2n$ and see if the $M[v,n-1]$ is different from $M[v,2n]$