Design and Analysis of Algorithms V

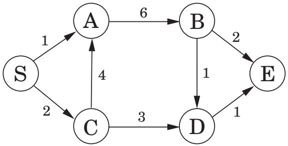
Sequence Alignment

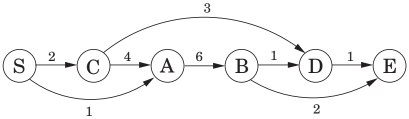
Guoqiang Li
School of Software

Shanghai Jiao Tong University

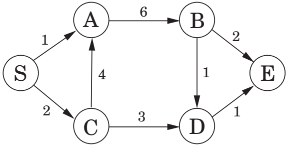
Shortest Paths in DAGs, Revisited

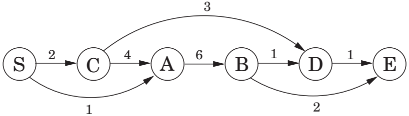
Shortest Paths in DAGs, Revisited





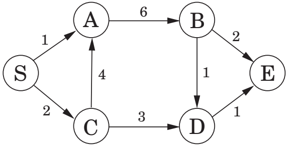
Shortest Paths in DAGs, Revisited

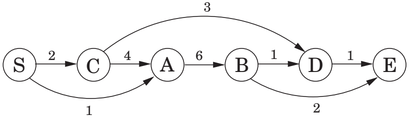




The special distinguishing feature of a DAG is that its nodes can be linearized, arranged on a line so that all edges go from left to right.

Shortest Paths in DAGs, Revisited





The special distinguishing feature of a DAG is that its nodes can be linearized, arranged on a line so that all edges go from left to right.

If compute `dist` values in the left-to-right order, we can always be sure that by the time we get to a node v, all the information we need is prepared to compute $\texttt{dist}(v)$.

Shortest Paths in DAGs, Revisited

Initialize all `dist`(.) value to ∞;
`dist`(s)=0;
for *each* $v \in V \backslash \{s\}$*, in linearized order* **do**
    `dist`$(v) = min_{(u,v) \in E}\{$`dist`$(u) + l(u,v)\}$;
end

Shortest Paths in DAGs, Revisited

Initialize all $\texttt{dist}(.)$ value to ∞;
$\texttt{dist}(s)$=0;
for *each* $v \in V \backslash \{s\}$, *in linearized order* **do**
    $\texttt{dist}(v) = min_{(u,v)\in E}\{\texttt{dist}(u) + l(u,v)\}$;
end

This algorithm is solving a collection of subproblems, $\{\texttt{dist}(u) \mid u \in V\}$

Dynamic Programming

Shanghai Jiao Tong University

The algorithm is solving a collection of subproblems, $\{\texttt{dist}(u) : u \in V\}$. Start with the smallest of them, $\texttt{dist}(s)$.

Dynamic Programming

The algorithm is solving a collection of subproblems, $\{\texttt{dist}(u) : u \in V\}$. Start with the smallest of them, $\texttt{dist}(s)$.

Proceed with progressively "larger" subproblems, where a larger subproblem is get to if a lot of other subproblems is solved.

Dynamic Programming

The algorithm is solving a collection of subproblems, $\{\texttt{dist}(u) : u \in V\}$. Start with the smallest of them, $\texttt{dist}(s)$.

Proceed with progressively "larger" subproblems, where a larger subproblem is get to if a lot of other subproblems is solved.

Dynamic programming is a powerful algorithmic paradigm where a problem is solved by identifying a collection of subproblems and tackling them one by one, until the whole lot of them is solved.

Dynamic Programming

SHANGHAI JIAO TONG UNIVERSITY

The algorithm is solving a collection of subproblems, $\{\texttt{dist}(u) : u \in V\}$. Start with the smallest of them, $\texttt{dist}(s)$.

Proceed with progressively "larger" subproblems, where a larger subproblem is get to if a lot of other subproblems is solved.

Dynamic programming is a powerful algorithmic paradigm where a problem is solved by identifying a collection of subproblems and tackling them one by one, until the whole lot of them is solved.

In dynamic programming we are given a DAG implicitly.

Longest Increasing Subsequences

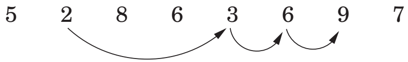
The input of longest increasing subsequence problem, is a sequence of numbers $a_1, \ldots, a_n$.

A subsequence is any subset of these numbers taken in order, of the form

$$a_{i_1}, a_{i_2}, \ldots, a_{i_k}$$

where $1 \leq i_1 < i_2 < \ldots < i_k \leq n$, and an increasing subsequence is one in which the numbers are getting strictly larger.

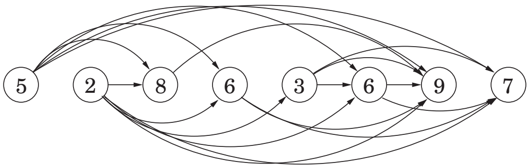
The task is to find the increasing subsequence of greatest length.



Graph Reformulation

A graph of all permissible transitions:

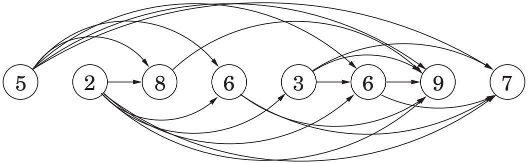
- A node i for each element a_i,
- Directed edges (i, j) whenever it is possible for a_i and a_j to be consecutive elements: $i < j$ and $a_i < a_j$



Graph Reformulation

A graph of all permissible transitions:

- A node i for each element a_i,
- Directed edges (i, j) whenever it is possible for a_i and a_j to be consecutive elements: $i < j$ and $a_i < a_j$



- This graph $G = (V, E)$ is a DAG, since all edges (i, j) have $i < j$
- There is a one-to-one correspondence between increasing subsequences and paths in this DAG.

The Algorithm

for $j = 1$ *to* n **do**

$\quad L(j) = 1 + \max\{L(i) \mid (i, j) \in E\}$;

end

`return(`$\max_j L(j)$`);`

Sequence Alignment

String Similarity

Q. How similar are two strings?

String Similarity

SHANGHAI JIAO TONG UNIVERSITY

Q. How similar are two strings?

Example. **ocurrance** and **occurrence**.

String Similarity

Q. How similar are two strings?

Example. **ocurrance** and **occurrence**.

o	c	u	r	r	a	n	c	e	–
o	c	c	u	r	r	e	n	c	e

6 mismatches, 1 gap

o	c	–	u	r	r	a	n	c	e
o	c	c	u	r	r	e	n	c	e

1 mismatch, 1 gap

o	c	–	u	r	r	–	a	n	c	e
o	c	c	u	r	r	e	–	n	c	e

0 mismatches, 3 gaps

Edit Distance

Edit distance. [Levenshtein 1966, Needleman-Wunsch 1970]

- Gap penalty δ; mismatch penalty α_{pq}.
- Cost = sum of gap and mismatch penalties.

Edit Distance

SHANGHAI JIAO TONG UNIVERSITY

Edit distance. [Levenshtein 1966, Needleman-Wunsch 1970]

- Gap penalty δ; mismatch penalty α_{pq}.
- Cost $=$ sum of gap and mismatch penalties.

C	T	-	G	A	C	C	T	A	C	G
C	T	G	G	A	C	G	A	A	C	G

cost=$\delta + \alpha_{CG} + \alpha_{TA}$

Edit Distance

SHANGHAI JIAO TONG UNIVERSITY

Edit distance. [Levenshtein 1966, Needleman-Wunsch 1970]

- Gap penalty δ; mismatch penalty α_{pq}.
- Cost $=$ sum of gap and mismatch penalties.

C	T	-	G	A	C	C	T	A	C	G		C	T	G	G	A	C	G	A	A	C	G

$$\textbf{cost}=\delta + \alpha_{CG} + \alpha_{TA}$$

Applications. Bioinformatics, spell correction, machine translation, speech recognition, information extraction, . . .

Edit Distance

SHANGHAI JIAO TONG UNIVERSITY

Edit distance. [Levenshtein 1966, Needleman-Wunsch 1970]

- Gap penalty δ; mismatch penalty α_{pq}.
- Cost $=$ sum of gap and mismatch penalties.

C	T	-	G	A	C	C	T	A	C	G		C	T	G	G	A	C	G	A	A	C	G

cost=$\delta + \alpha_{CG} + \alpha_{TA}$

Applications. Bioinformatics, spell correction, machine translation, speech recognition, information extraction, . . .

Example.

Spokesperson	*confirms*		*senior*	*government*	*adviser*	*was*	*found*
Spokesperson	*said*	*the*	*senior*		*adviser*	*was*	*found*

BLOSUM Matrix for Proteins

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	7	-3	-3	-3	-1	-2	-2	0	-3	-3	-3	-1	-2	-4	-1	2	0	-5	-4	-1
R	-3	9	-1	-3	-6	1	-1	-4	0	-5	-4	3	-3	-5	-3	-2	-2	-5	-4	-4
N	-3	-1	9	2	-5	0	-1	-1	1	-6	-6	0	-4	-6	-4	1	0	-7	-4	-5
D	-3	-3	2	10	-7	-1	2	-3	-2	-7	-7	-2	-6	-6	-3	-1	-2	-8	-6	-6
C	-1	-6	-5	-7	13	-5	-7	-6	-7	-2	-3	-6	-3	-4	-6	-2	-2	-5	-5	-2
Q	-2	1	0	-1	-5	9	3	-4	1	-5	-4	2	-1	-5	-3	-1	-1	-4	-3	-4
E	-2	-1	-1	2	-7	3	8	-4	0	-6	-6	1	-4	-6	-2	-1	-2	-6	-5	-4
G	0	-4	-1	-3	-6	-4	-4	9	-4	-7	-7	-3	-5	-6	-5	-1	-3	-6	-6	-6
H	-3	0	1	-2	-7	1	0	-4	12	-6	-5	-1	-4	-2	-4	-2	-3	-4	3	-5
I	-3	-5	-6	-7	-2	-5	-6	-7	-6	7	2	-5	2	-1	-5	-4	-2	-5	-3	4
L	-3	-4	-6	-7	-3	-4	-6	-7	-5	2	6	-4	3	0	-5	-4	-3	-4	-2	1
K	-1	3	0	-2	-6	2	1	-3	-1	-5	-4	8	-3	-5	-2	-1	-1	-6	-4	-4
M	-2	-3	-4	-6	-3	-1	-4	-5	-4	2	3	-3	9	0	-4	-3	-1	-3	-3	1
F	-4	-5	-6	-6	-4	-5	-6	-6	-2	-1	0	-5	0	10	-6	-4	-4	0	4	-2
P	-1	-3	-4	-3	-6	-3	-2	-5	-4	-5	-5	-2	-4	-6	12	-2	-3	-7	-6	-4
S	2	-2	1	-1	-2	-1	-1	-1	-2	-4	-4	-1	-3	-4	-2	7	2	-6	-3	-3
T	0	-2	0	-2	-2	-1	-2	-3	-3	-2	-3	-1	-1	-4	-3	2	8	-5	-3	0
W	-5	-5	-7	-8	-5	-4	-6	-6	-4	-5	-4	-6	-3	0	-7	-6	-5	16	3	-5
Y	-4	-4	-4	-6	-5	-3	-5	-6	3	-3	-2	-4	-3	4	-6	-3	-3	3	11	-3
V	-1	-4	-5	-6	-2	-4	-4	-6	-5	4	1	-4	1	-2	-4	-3	0	-5	-3	7

Exercise

What is edit distance between these two strings?

P A L E T T E   P A L A T E

Assume gap penalty $= 2$ and mismatch penalty $= 1$.

A. 1
B. 2
C. 3
D. 4
E. 5

Merging

Goal. Given two strings $x_1x_2\ldots x_m$ and $y_1y_2\ldots y_n$, find a min-cost alignment.

Merging

Goal. Given two strings $x_1 x_2 \ldots x_m$ and $y_1 y_2 \ldots y_n$, find a min-cost alignment.

Definition. An alignment M is a set of ordered pairs $x_i - y_j$ such that each character appears in at most one pair and no crossings.

Merging

Goal. Given two strings $x_1 x_2 \ldots x_m$ and $y_1 y_2 \ldots y_n$, find a min-cost alignment.

Definition. An alignment M is a set of ordered pairs $x_i - y_j$ such that each character appears in at most one pair and no crossings.

Definition The cost of an alignment M is:

$$\text{cost}(M) = \underbrace{\sum_{(x_i, y_j) \in M} \alpha_{x_i y_j}}_{\text{mismatch}} + \underbrace{\sum_{i : x_i \text{ unmached}} \delta + \sum_{j : y_j \text{ unmatched}} \delta}_{\text{gap}}$$

Merging

Goal. Given two strings $x_1 x_2 \ldots x_m$ and $y_1 y_2 \ldots y_n$, find a min-cost alignment.

Definition. An alignment M is a set of ordered pairs $x_i - y_j$ such that each character appears in at most one pair and no crossings.

Definition The cost of an alignment M is:

$$\text{cost}(M) = \underbrace{\sum_{(x_i, y_j) \in M} \alpha_{x_i y_j}}_{\text{mismatch}} + \underbrace{\sum_{i:x_i \text{ unmached}} \delta + \sum_{j:y_j \text{ unmatched}} \delta}_{\text{gap}}$$

x_1	x_2	x_3	x_4	x_5		x_6
C	T	A	C	C	-	G
-	T	A	C	A	T	G
	y_1	y_2	y_3	y_4	y_5	y_6

an alignment of CTACCG and TACATG

$$M = \{x_2 - y_1, x_3 - y_2, x_4 - y_3, x_5 - y_4, x_6 - y_6\}$$

Sequence Alignment: Problem Structure

Definition $OPT(i,j)$: min cost of aligning prefix strings $x_1 x_2 \ldots x_i$ and $y_1 y_2 \ldots y_j$.

Goal. $OPT(m,n)$.

Sequence Alignment: Problem Structure

SHANGHAI JIAO TONG UNIVERSITY

Definition $OPT(i, j)$: min cost of aligning prefix strings $x_1x_2 \dots x_i$ and $y_1y_2 \dots y_j$.

Goal. $OPT(m, n)$.

Case 1. $OPT(i, j)$ matches $x_i - y_j$.
Pay mismatch for $x_i - y_j$ + min cost of aligning $x_1x_2 \dots x_{i-1}$ and $y_1y_2...y_{j-1}$.

Sequence Alignment: Problem Structure

Shanghai Jiao Tong University

Definition $OPT(i,j)$: min cost of aligning prefix strings $x_1 x_2 \ldots x_i$ and $y_1 y_2 \ldots y_j$.

Goal. $OPT(m,n)$.

Case 1. $OPT(i,j)$ matches $x_i - y_j$.
Pay mismatch for $x_i - y_j$ + min cost of aligning $x_1 x_2 \ldots x_{i-1}$ and $y_1 y_2 ... y_{j-1}$.

Case 2a. $OPT(i,j)$ leaves x_i unmatched.
Pay gap for x_i + min cost of aligning $x_1 x_2 ... x_{i-1}$ and $y_1 y_2 \ldots y_j$.

Sequence Alignment: Problem Structure

Definition $OPT(i, j)$: min cost of aligning prefix strings $x_1 x_2 \dots x_i$ and $y_1 y_2 \dots y_j$.

Goal. $OPT(m, n)$.

Case 1. $OPT(i, j)$ matches $x_i - y_j$.
Pay mismatch for $x_i - y_j$ + min cost of aligning $x_1 x_2 \dots x_{i-1}$ and $y_1 y_2 ... y_{j-1}$.

Case 2a. $OPT(i, j)$ leaves x_i unmatched.
Pay gap for x_i + min cost of aligning $x_1 x_2 ... x_{i-1}$ and $y_1 y_2 \dots y_j$.

Case 2b. $OPT(i, j)$ leaves y_j unmatched.
Pay gap for y_j+ min cost of aligning $x_1 x_2 ... x_i$ and $y_1 y_2 \dots y_{j-1}$.

Sequence Alignment: Problem Structure

Bellman equation.

$$OPT(i,j) = \begin{cases} j\delta & \text{if i} = 0 \\ i\delta & \text{if j} = 0 \\ \min \begin{cases} \alpha_{x_i y_j} + OPT(i-1, j-1) \\ \delta + OPT(i-1, j) \\ \delta + OPT(i, j-1) \end{cases} & \text{otherwise} \end{cases}$$

Sequence Alignment: Algorithm

SHANGHAI JIAO TONG UNIVERSITY

```
SEQUENCEALIGNMENT(m, n, x_1, ..., x_m, y_1, ..., y_n, δ, α)
for i = 0 to m do
    M[i, 0] ← iδ;
end
for j = 0 to n do
    M[0, j] ← jδ;
end
for i = 1 to m do
    for j = 1 to n do
        M[i, j] ← min{α_{x_i y_j} + M[i − 1, j − 1], δ + M[i − 1, j], δ + M[i, j − 1]};
    end
end
RETURN M[m, n];
```

Sequence Alignment: Traceback

SHANGHAI JIAO TONG UNIVERSITY

		S	I	M	I	L	A	R	I	T	Y
	0	2	4	6	8	10	12	14	16	18	20
I	2	4	1	3	2	4	6	8	7	9	11
D	4	6	3	3	4	4	6	8	9	9	11
E	6	8	5	5	6	6	6	8	10	11	11
N	8	10	7	7	8	8	8	8	10	12	13
T	10	12	9	9	9	10	10	10	10	9	11
I	12	14	8	10	8	10	12	12	9	11	11
T	14	16	10	10	10	10	12	14	11	8	11
Y	16	18	12	12	12	12	12	14	13	10	7

Sequence Alignment: Analysis

Theorem

The DP algorithm computes the edit distance (and an optimal alignment) of two strings of lengths m *and* n *in* $\Theta(mn)$ *time and space.*

Sequence Alignment: Analysis

Theorem

The DP algorithm computes the edit distance (and an optimal alignment) of two strings of lengths m and n in $\Theta(mn)$ time and space.

Proof.

Sequence Alignment: Analysis

Theorem

The DP algorithm computes the edit distance (and an optimal alignment) of two strings of lengths m and n in $\Theta(mn)$ time and space.

Proof.

- Algorithm computes edit distance.
- Can trace back to extract optimal alignment itself.

Hirschberg's Algorithm

Sequence Alignment in Linear Space

[Hirschberg] There exists an algorithm to find an optimal alignment in $O(mn)$ time and $O(m+n)$ space.

Sequence Alignment in Linear Space

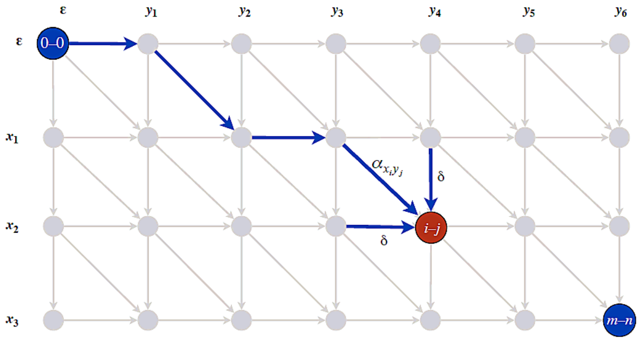
[Hirschberg] There exists an algorithm to find an optimal alignment in $O(mn)$ time and $O(m+n)$ space.

- Clever combination of divide-and-conquer and dynamic programming.
- Inspired by idea of Savitch from complexity theory.

Hirschberg's Algorithm

Edit distance graph.

- Let $f(i,j)$ denote length of shortest path from $(0,0)$ to (i,j).
- Lemma: $f(i,j) = OPT(i,j)$ for all i and j.



Hirschberg's Algorithm

Shanghai Jiao Tong University

Edit distance graph.

- Let $f(i,j)$ denote length of shortest path from $(0,0)$ to (i,j).
- Lemma: $f(i,j) = OPT(i,j)$ for all i and j.

Proof.

Hirschberg's Algorithm

SHANGHAI JIAO TONG UNIVERSITY

Edit distance graph.

- Let $f(i,j)$ denote length of shortest path from $(0,0)$ to (i,j).
- Lemma: $f(i,j) = OPT(i,j)$ for all i and j.

Proof. [by strong induction on $i+j$]

Edit distance graph.

- Let $f(i,j)$ denote length of shortest path from $(0,0)$ to (i,j).
- Lemma: $f(i,j) = OPT(i,j)$ for all i and j.

Proof. [by strong induction on $i+j$]

- Base case: $f(0,0) = OPT(0,0) = 0$.

Hirschberg's Algorithm

Edit distance graph.

- Let $f(i, j)$ denote length of shortest path from $(0, 0)$ to (i, j).
- Lemma: $f(i, j) = OPT(i, j)$ for all i and j.

Proof. [by strong induction on $i + j$]

- Base case: $f(0, 0) = OPT(0, 0) = 0$.
- Inductive hypothesis: assume true for all (i', j') with $i' + j' < i + j$.

Hirschberg's Algorithm

Shanghai Jiao Tong University

Edit distance graph.

- Let $f(i,j)$ denote length of shortest path from $(0,0)$ to (i,j).
- Lemma: $f(i,j) = OPT(i,j)$ for all i and j.

Proof. [by strong induction on $i+j$]

- Base case: $f(0,0) = OPT(0,0) = 0$.
- Inductive hypothesis: assume true for all (i',j') with $i'+j' < i+j$.
- Last edge on shortest path to (i,j) is from $(i-1,j-1)$, $(i-1,j)$, or $(i,j-1)$.

SHANGHAI JIAO TONG UNIVERSITY

Edit distance graph.

- Let $f(i,j)$ denote length of shortest path from $(0,0)$ to (i,j).
- Lemma: $f(i,j) = OPT(i,j)$ for all i and j.

Proof. [by strong induction on $i+j$]

- Base case: $f(0,0) = OPT(0,0) = 0$.
- Inductive hypothesis: assume true for all (i',j') with $i'+j' < i+j$.
- Last edge on shortest path to (i,j) is from $(i-1,j-1)$, $(i-1,j)$, or $(i,j-1)$.
- Thus,

$$
\begin{aligned}
f(i,j) &= \min\left\{\alpha_{x_i y_j} + f(i-1,j-1), \delta + f(i-1,j), \delta + f(i,j-1)\right\} \\
&= \min\left\{\alpha_{x_i y_j} + OPT(i-1,j-1), \delta + OPT(i-1,j), \delta + OPT(i,j-1)\right\} \\
&= OPT(i,j)
\end{aligned}
$$

Hirschberg's Algorithm

Edit distance graph.

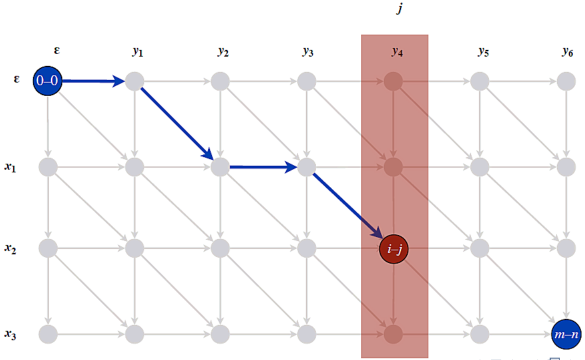
- Let $f(i, j)$ denote length of shortest path from $(0, 0)$ to (i, j).
- Lemma: $f(i, j) = OPT(i, j)$ for all i and j.

Hirschberg's Algorithm

SHANGHAI JIAO TONG UNIVERSITY

Edit distance graph.

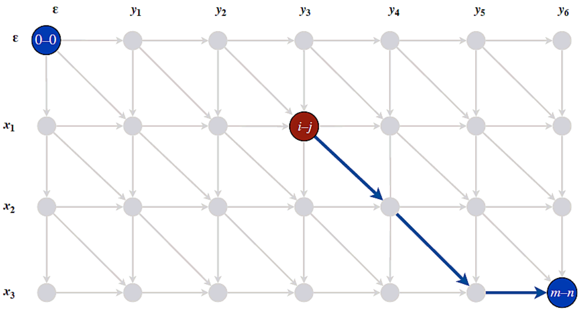
- Let $f(i, j)$ denote length of shortest path from $(0, 0)$ to (i, j).
- Lemma: $f(i, j) = OPT(i, j)$ for all i and j.
- Can compute $f(\cdot, j)$ for any j in $O(mn)$ time and $O(m)$ space.



Hirschberg's Algorithm

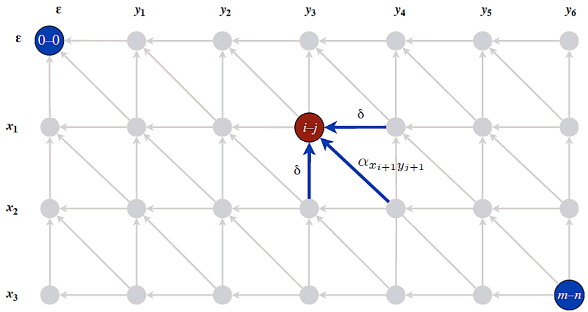
Edit distance graph.

- Let $g(i, j)$ denote length of shortest path from (i, j) to (m, n).



Edit distance graph.

- Let $g(i, j)$ denote length of shortest path from (i, j) to (m, n).
- Can compute $g(i, j)$ by reversing the edge orientations and inverting the roles of $(0, 0)$ and (m, n).

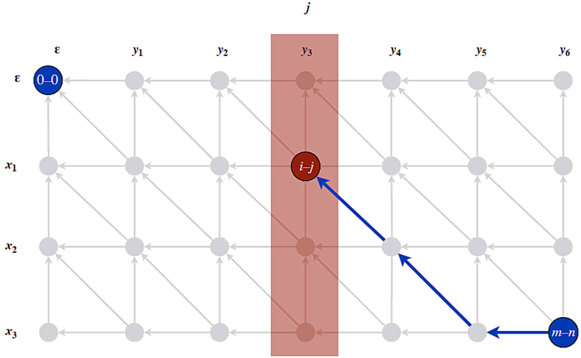


Hirschberg's Algorithm

SHANGHAI JIAO TONG UNIVERSITY

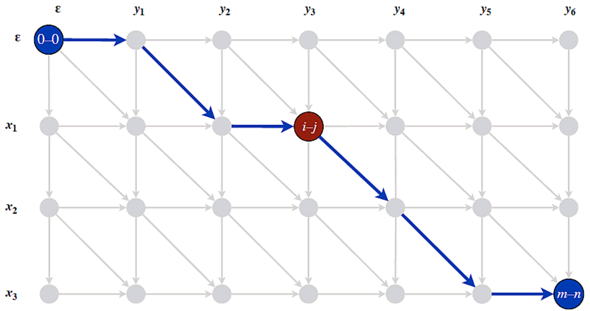
Edit distance graph.

- Let $g(i, j)$ denote length of shortest path from (i, j) to (m, n).
- Can compute $g(\cdot, j)$ for any j in $O(mn)$ time and $O(m)$ space.



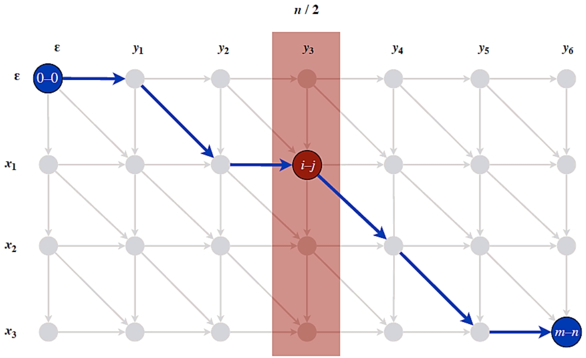
Hirschberg's Algorithm

Observation 1. The length of a shortest path that uses (i, j) is $f(i, j) + g(i, j)$.



Hirschberg's Algorithm

Observation 2. let q be an index that minimizes $f(q, n/2) + g(q, n/2)$. Then, there exists a shortest path from $(0, 0)$ to (m, n) that uses $(q, n/2)$.

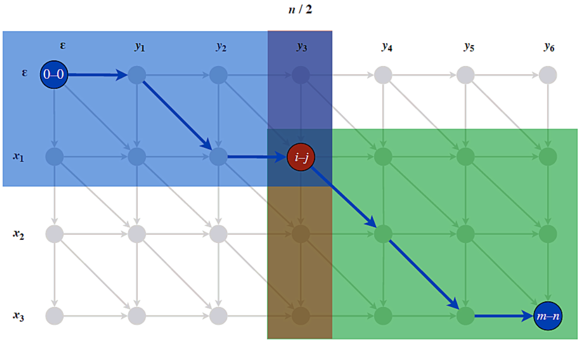


Hirschberg's Algorithm

Shanghai Jiao Tong University

Divide. Find index q that minimizes $f(q, n/2) + g(q, n/2)$; save node i-j as part of solution.

Conquer. Recursively compute optimal alignment in each piece.



Hirschberg's Algorithm: Space Analysis

SHANGHAI JIAO TONG UNIVERSITY

Theorem

Hirschberg's algorithm uses $\Theta(m+n)$ *space.*

Hirschberg's Algorithm: Space Analysis

SHANGHAI JIAO TONG UNIVERSITY

Theorem

Hirschberg's algorithm uses $\Theta(m+n)$ *space.*

Proof.

Hirschberg's Algorithm: Space Analysis

Theorem

Hirschberg's algorithm uses $\Theta(m+n)$ space.

Proof.

- Each recursive call uses $\Theta(m)$ space to compute $f(\cdot, n/2)$ and $g(\cdot, n/2)$.
- Only $\Theta(1)$ space needs to be maintained per recursive call.
- Number of recursive calls $\leq n$.

Exercise

What is the worst-case running time of Hirschberg's algorithm?

A. $O(mn)$

B. $O(mn \log m)$

C. $O(mn \log n)$

D. $O(mn \log m \log n)$

Running Time Analysis Warmup

Theorem

Let $T(m,n)$ be max running time of Hirschberg's algorithm on strings of lengths at most m and n. Then, $T(m,n) = O(mn\log n)$.

Running Time Analysis Warmup

Theorem

Let $T(m,n)$ be max running time of Hirschberg's algorithm on strings of lengths at most m and n. Then, $T(m,n) = O(mn \log n)$.

Proof.

- $T(m,n)$ is monotone nondecreasing in both m and n.
-

$$\begin{aligned} T(m,n) &\le 2T(m,n/2) + O(mn) \\ &\Rightarrow T(m,n) = O(mn \log n) \end{aligned}$$

Running Time Analysis Warmup

Theorem

Let $T(m,n)$ be max running time of Hirschberg's algorithm on strings of lengths at most m and n. Then, $T(m,n) = O(mn \log n)$.

Proof.

- $T(m,n)$ is monotone nondecreasing in both m and n.
-
$$\begin{aligned} T(m,n) &\le 2T(m,n/2) + O(mn) \\ &\Rightarrow T(m,n) = O(mn \log n) \end{aligned}$$

Remark. Analysis is not tight because two subproblems are of size $(q, n/2)$ and $(m-q, n/2)$. Next, we prove $T(m,n) = O(mn)$.

Running Time Analysis

Theorem

Let $T(m,n)$ be max running time of Hirschberg's algorithm on strings of lengths at most m and n. Then, $T(m,n) = O(mn)$.

Running Time Analysis

Theorem

Let $T(m, n)$ be max running time of Hirschberg's algorithm on strings of lengths at most m and n. Then, $T(m, n) = O(mn)$.

Proof.

Running Time Analysis

Shanghai Jiao Tong University

Theorem

Let $T(m, n)$ be max running time of Hirschberg's algorithm on strings of lengths at most m and n. Then, $T(m, n) = O(mn)$.

Proof. [by strong induction on $m + n$]

Running Time Analysis

SHANGHAI JIAO TONG UNIVERSITY

Theorem

Let $T(m,n)$ be max running time of Hirschberg's algorithm on strings of lengths at most m and n. Then, $T(m,n) = O(mn)$.

Proof. [by strong induction on $m+n$]

- $O(mn)$ time to compute $f(\cdot, n/2)$ and $g(\cdot, n/2)$ and find index q.
- $T(q, n/2) + T(m-q, n/2)$ time for two recursive calls.
- Choose constant c so that:

$$T(m,2) \leq cm$$
$$T(2,n) \leq cn$$
$$T(m,n) \leq cmn + T(q,n/2) + T(m-q,n/2)$$

Running Time Analysis

Shanghai Jiao Tong University

Claim

$T(m, n) \leq 2cmn$

Running Time Analysis

SHANGHAI JIAO TONG UNIVERSITY

Claim

$T(m,n) \le 2cmn$

- Base cases: $m = 2$ and $n = 2$.
- Inductive hypothesis: $T(m,n) \le 2cmn$ for all (m', n') with $m' + n' < m + n$.

$$\begin{aligned} T(m,n) &\le T(q, n/2) + T(m-q, n/2) + cmn \\ &\le 2cqn/2 + 2c(m-q)n/2 + cmn \\ &= cqn + cmn - cqn + cmn \\ &= 2cmn \end{aligned}$$

Longest common subsequence

Problem. Given two strings $x_1 x_2 \ldots x_m$ and $y_1 y_2 \ldots y_n$, find a common subsequence that is as long as possible.

Longest common subsequence

Problem. Given two strings $x_1x_2\dots x_m$ and $y_1y_2\dots y_n$, find a common subsequence that is as long as possible.

Alternative viewpoint. Delete some characters from x; delete some character from y; a common subsequence if it results in the same string.

Longest common subsequence

Problem. Given two strings $x_1 x_2 \ldots x_m$ and $y_1 y_2 \ldots y_n$, find a common subsequence that is as long as possible.

Alternative viewpoint. Delete some characters from x; delete some character from y; a common subsequence if it results in the same string.

Example. **LCS(GGCACCACG, ACGGCGGATACG) = GGCAACG**.

Longest common subsequence

Problem. Given two strings $x_1 x_2 \ldots x_m$ and $y_1 y_2 \ldots y_n$, find a common subsequence that is as long as possible.

Alternative viewpoint. Delete some characters from x; delete some character from y; a common subsequence if it results in the same string.

Example. **LCS(GGCACCACG, ACGGCGGATACG) = GGCAACG**.

Applications. Unix diff, git, bioinformatics.

Quiz

How about the longest common string?

Referred Materials

Referred Materials

- Content of this lecture comes from Section 6.1 and 6.2 in [DPV07], Section 6.6 and 6.7 in [KT05].
- Suggest to read Section 6.2 in [KT05].