Quiz

Jeremy Lin has created a time machine. Now, he knows exactly the price of $GME for the next n days, which are $p_1, p_2, \cdots, p_n$.

Suppose Jeremy can only trade $GME for at most t times.

So, what is the best trading?

Let $w_{k,t}$ be the network at k-th day using t trades.

$$w_{k,t} = \max\left(w_{k-1,t}, \max_{j<k}\left(w_{j,t-1}\frac{p_k}{p_j}\right)\right).$$

This solution technically is wrong. What is the mistake?
I omitted initial cases.

CSE 421

Dynamic Programming

RNA, Sequence Alignment

Yin Tat Lee

Announcement

- HW5 will be posted tonight. Sorry for the late.
- Swati OH is moved to Sunday (virtual). See website.
- Midterm is graded. Check your score on Canvas.
- Come to any OH for any grading mistakes.
- Come to my OH to get back the midterm (for in-person midterm)
- I will post the midterm solution tonight.

Midterm

Here is the statistics

percentile	Q1	Q2	Q3	Q4	Q5	Total
25%	12 (75%)	17.5 (72%)	6.5 (65%)	3 (20%)	18 (72%)	66 (73%)
50%	14 (88%)	21 (87.5%)	9 (90%)	8 (53%)	24 (96%)	72.8 (81%)
75%	16 (100%)	21 (87.5%)	10 (100%)	14.5 (97%)	25 (100%)	80.8 (90%)

Q4 is the hardest one.
It is modified from some problem in a programming contest.

If you get >= 50/90 in this midterm, you are on track for a 3.4 (depending on your homework)

Midterm

- $n \cdot 2^{\log^2 n}$ is not $O(n^3)$.

Note that $n \cdot 2^{\log^2 n} = n^{1+\log n}$ which is not even polynomial.

- Some write $n^{\log_2 4}$. Please write n^2 instead.

- Don’t write a large ambiguous paragraph to describe your algo.

Use pseudo code instead. We will deduct point in final.

- MST takes $O(m \log n)$ or $O(m + n \log n)$, but not $O(n \log n)$.

- Q5, you don’t need to do induction.

Knapsack Problem

Given n objects and a "knapsack."

Item i weighs $w_i > 0$ kilograms and has value $v_i > 0$.

Knapsack has capacity of W kilograms.

Goal: fill knapsack so as to maximize total value.

Ex: OPT is { 3, 4 } with value 40.

W = 11

Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

Greedy: repeatedly add item with maximum ratio v_i/w_i.

Ex: { 5, 2, 1 } achieves only value = 35 $\Rightarrow$ greedy not optimal.

Stronger DP (Strengthening Hypothesis)

Let $OPT(i, w)$ = Max value of subsets of items $1, \ldots, i$ of weight $\leq w$

Case 1: $OPT(i, w)$ selects item i

- In this case, $OPT(i, w) = v_i + OPT(i - 1, w - w_i)$

Case 2: $OPT(i, w)$ does not select item i

- In this case, $OPT(i, w) = OPT(i - 1, w)$.

Take best of the two

Therefore,

$$OPT(i, w) = \begin{cases} 0 & \text{If } i = 0 \\ OPT(i - 1, w) & \text{If } w_i > w \\ \max(OPT(i - 1, w), v_i + OPT(i - 1, w - w_i)) & \text{o.w.,} \end{cases}$$

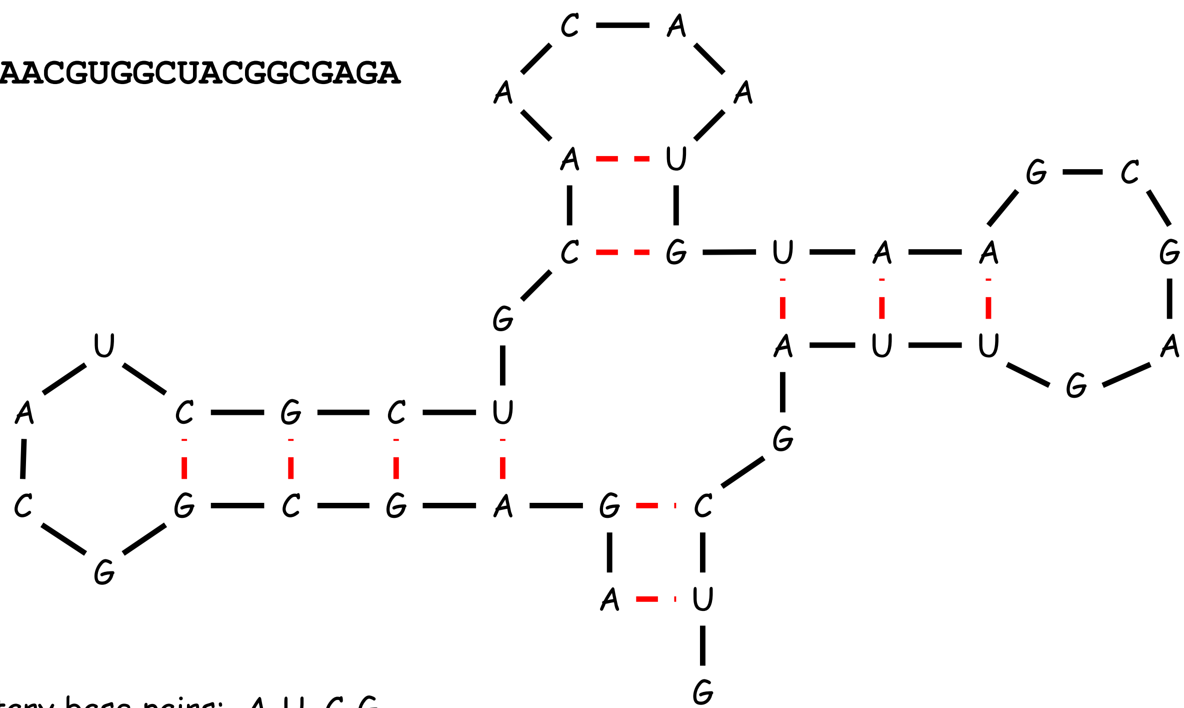
RNA Secondary Structure

RNA Secondary Structure

RNA: A String B = $b_1 b_2 \ldots b_n$ over alphabet { A, C, G, U }.

Secondary structure. RNA is single-stranded so it tends to loop back and form base pairs with itself. This structure is essential for understanding behavior of molecule.

Ex: **GUCGAUUGAGCGAAUGUAACAACGUGGCUACGGCGAGA**



complementary base pairs: A-U, C-G

RNA Secondary Structure (Formal)

Secondary structure. A set of pairs $S = \{(b_i, b_j)\}$ that satisfy:

[Matching]: S is a matching.

[Valid]: each pair in S is: $A - U$, $U - A$, $C - G$, or $G - C$.

[No sharp turns]: The ends of each pair are separated by at least 4 intervening bases. If $(b_i, b_j) \in S$, then $i < j - 4$.

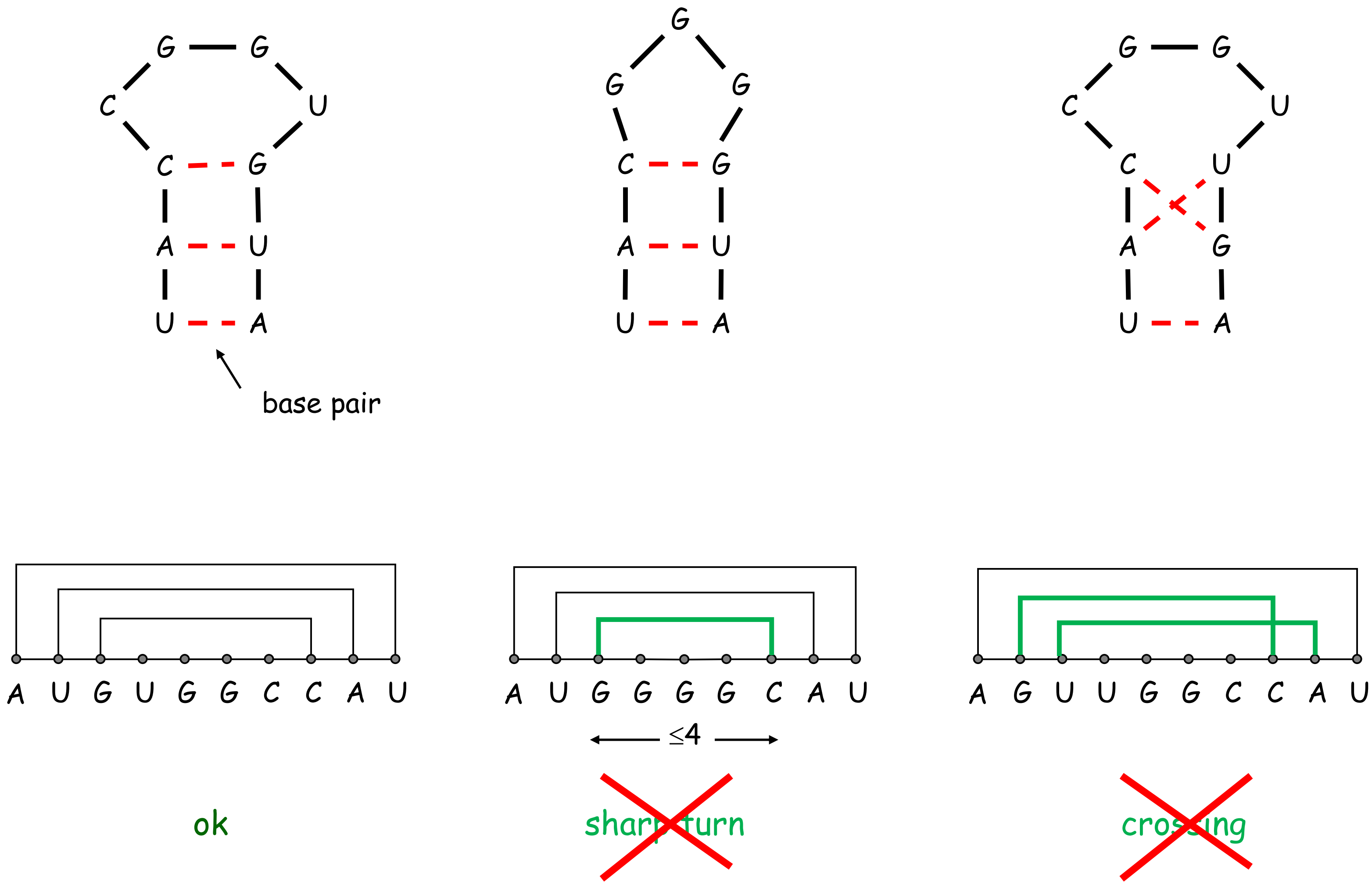
[Non-crossing] If (b_i, b_j) and (b_k, b_l) are two pairs in S, then we cannot have $i < k < j < l$.

Free energy: Usual hypothesis is that an RNA molecule will maximize total free energy.

approximate by number of base pairs

Goal: Given an RNA molecule B = $b_1 b_2 \ldots b_n$, find a secondary structure S that maximizes the number of base pairs.

Secondary Structure (Examples)

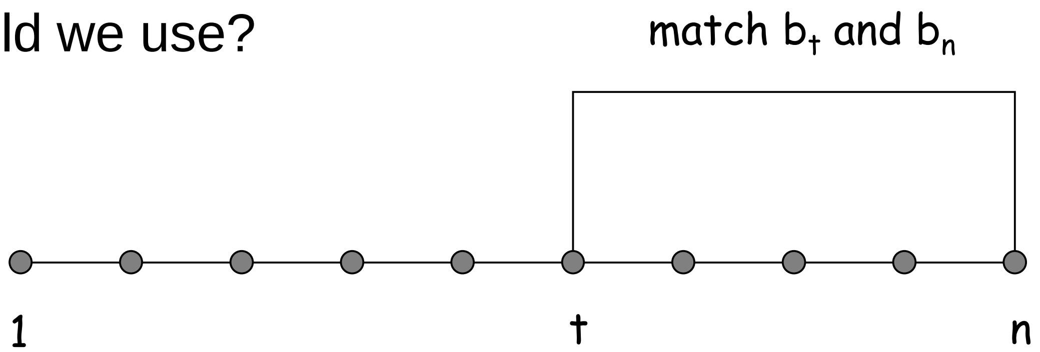


DP: First Attempt

First attempt. Let $OPT(n)$ = maximum number of base pairs in a secondary structure of the substring $b_1b_2\ldots b_n$.

Suppose b_n is matched with b_t in $OPT(n)$.

What IH should we use?



Difficulty: This naturally reduces to two subproblems

- Finding secondary structure in $b_1, \ldots, b_{t-1}$, i.e., OPT(t-1)
- Finding secondary structure in $b_{t+1}, \ldots, b_{n-1}$, ???

DP: Second Attempt

Definition: $OPT(i, j)$ = maximum number of base pairs in a secondary structure of the substring $b_i, b_{i+1}, \ldots, b_j$

The most important part of a correct DP; It fixes IH

Case 1: If $j - i \le 4$.

- $OPT(i, j) = 0$ by no-sharp turns condition.

Case 2: Base b_j is not involved in a pair.

- $OPT(i, j) = OPT(i, j - 1)$

Case 3: Base b_j pairs with b_t for some $i \le t < j - 4$

- non-crossing constraint decouples resulting sub-problems
- $OPT(i, j) = \max_{i \le t < j-4} \{ 1 + OPT(i, t - 1) + OPT(t + 1, j - 1) \}$

Recursive Code

```
Let M[i,j]=empty for all i,j.

Compute-OPT(i,j){
   if (j-i <= 4)
     return 0;
   if (M[i,j] is empty)
      M[i,j]=Compute-OPT(i,j-1)
      for t=i to j-5 do
         if (b_t,b_j is in {A-U, U-A, C-G, G-C})
           M[i,j]=max(M[i,j], 1+Compute-OPT(i,t-1) +
                      Compute-OPT(t+1,j-1))
   return M[j]
}
```

Does this code terminate?

Formal Induction

Let $OPT(i, j)$ = maximum number of base pairs in a secondary structure of the substring $b_i, b_{i+1}, \ldots, b_j$

Base Case: $OPT(i, j) = 0$ for all i, j where $|j - i| \leq 4$.

IH: For some $\ell \geq 4$, Suppose we have computed $OPT(i, j)$ for all i, j where $|i - j| \leq \ell$.

IS: Goal: We find $OPT(i, j)$ for all i, j where $|i - j| = \ell + 1$. Fix i, j such that $|i - j| = \ell + 1$.

Case 1: Base b_j is not involved in a pair.

- $OPT(i, j) = OPT(i, j - 1)$ [this we know by IH since $|i - (j - 1)| = \ell$]

Case 2: Base b_j pairs with b_t for some $i \leq t < j - 4$

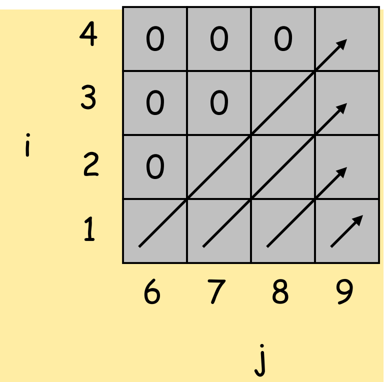
- $OPT(i, j) = \max_{i \leq t < j-4} \{ 1 + OPT(i, t - 1) + OPT(t + 1, j - 1) \}$

We know by IH since difference $\leq \ell$

Bottom-up DP

```
for ℓ = 1, 2, …, n-1
  for i = 1, 2, …, n-1
    j = i + ℓ
    if (ℓ <= 4)
      M[i,j]=0;
      else
        M[i,j]=M[i,j-1]
        for t=i to j-5 do
          if (b_t,b_j is in {A-U, U-A, C-G, G-C})
            M[i,j]=max(M[i,j], 1+ M[i,t-1] + M[t+1,j-1])

  return M[1, n]
}
```



Running Time: $O(n^3)$

(It is also okay to loop over i, j or j, i)

When poll is active, respond at **pollev.com/yintat**

Given n positive integers a1,…,an, decide whether the integers can be partitioned into 3 sets, such that each set has the same sum.

Total Results: 0

Quiz Solution

Let $A(i, x, y, z)$ be true if and only if the numbers $a_1, a_2, \cdots, a_i$ can be partitioned into three sets whose sums are x, y, z.

If $i > 0$, we have

$$A(i, x, y, z)$$
$$= A(i-1, x-a_i, y, z) \; or \; A(i-1, x, y-a_i, z) \; or \; A(i-1, x, y, z-a_i)$$

If $i = 0$, we have

$A(0, x, y, z) =$ True if $x = y = z = 0$, False otherwise.

Sequence Alignment
(Edit distance)

Word Alignment

How similar are two strings?

ocurrance

occurrence

o	c	u	r	r	a	n	c	e	-
o	c	c	u	r	r	e	n	c	e

5 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]

Cost = # of gaps + #mismatches.

How to formalize the question.

Applications.

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

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

Cost: 5

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

Cost: 3

Sequence Alignment

Given two strings $x_1, \ldots, x_m$ and $y_1, \ldots, y_n$ find an alignment with minimum number of mismatch and gaps.

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

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.

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

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 $x_i \neq y_j$ + min cost of aligning $x_1, \ldots, x_{i-1}$ 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, 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?

```
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])
```

Induction

What is the order of induction? (i.e. why there is no loop?)
We can do induction on $i + j$.
(Alternatively, we can induct on the “step” of the algorithm)

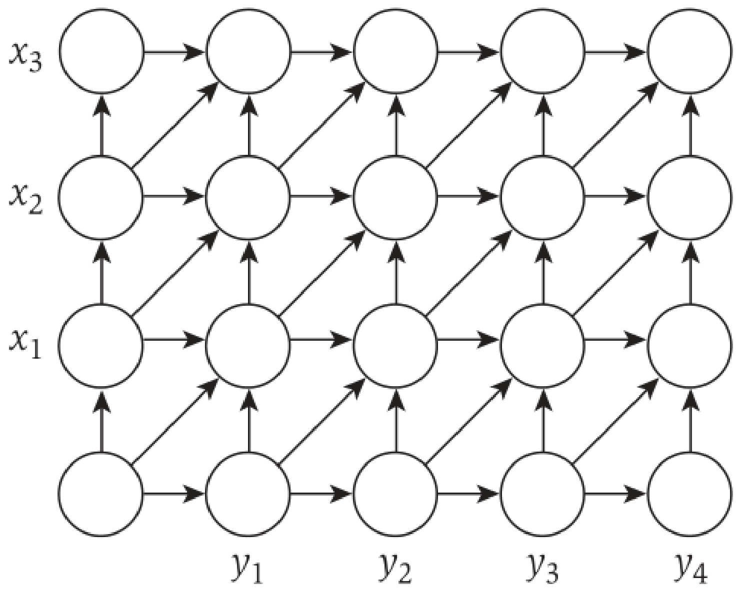


Figure 6.17 A graph-based picture of sequence alignment.

Optimizing Memory

We just need to use the last (row) of computed values.

```
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]
}
```

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

```
Sequence-Alignment(m, n, x1x2...xm, y1y2...yn) {
   for i = 0 to m
      O[i] = i
   for i = 1 to m
      N[0]=i                                       M[i-1, j-1]
      for j = 1 to n
         N[j] = min( (xi=yj ? 0:1) + O[j-1],
                     1 + O[j],                      M[i-1, j]
                     1 + N[j-1])                    M[i, j-1]
      for j = 1 to n
         O[j]=N[j]
   return N[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])
```

Shortest Path

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?

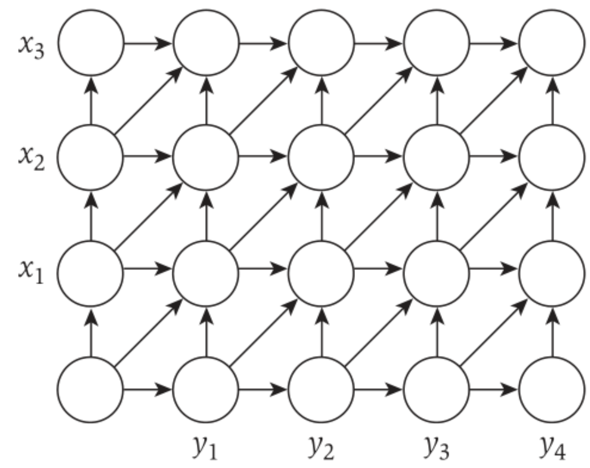
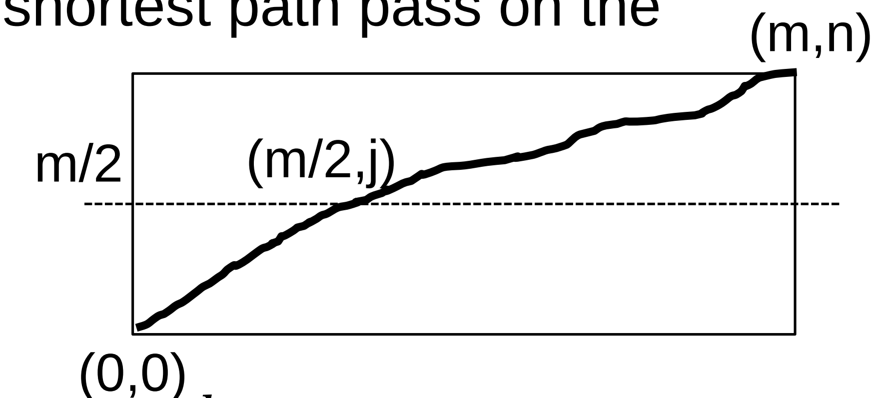


Figure 6.17 A graph-based picture of sequence alignment.

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

Find(i_1,j_1,i_2,j_2) { // Due to spacing, ignored boundary cases

  Let $\boldsymbol{k} = \lfloor(\boldsymbol{i_1} + \boldsymbol{i_2})/\mathbf{2}\rfloor$

  Compute $\boldsymbol{d_{(i_1,j_1)\to(k,j_2)}}$ **for all** $\boldsymbol{j}$ **via Sequence-Alignment.**

  Compute $\boldsymbol{d_{(k,j)\to(i_2,j_2)}}$ **for all** $\boldsymbol{j}$ **via similar algo run backward.**

  Let $\boldsymbol{j} = \mathrm{argmin}_j \boldsymbol{d_{(i_1,j_1)\to(k,j_2)}} + \boldsymbol{d_{(k,j_2)\to(i_2,j_2)}}$

  $\boldsymbol{p_1}$ = **Find(i_1,j_1,k,j)**

  $\boldsymbol{p_2}$ = **Find(k,j,i_2,j_2)**

  return $\boldsymbol{p_1}, \boldsymbol{p_2}$

}

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 $O(\frac{n^2}{\log^2 n})$ (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 $O(n^{1+\epsilon})$ (~2020).

Longest Path in a DAG

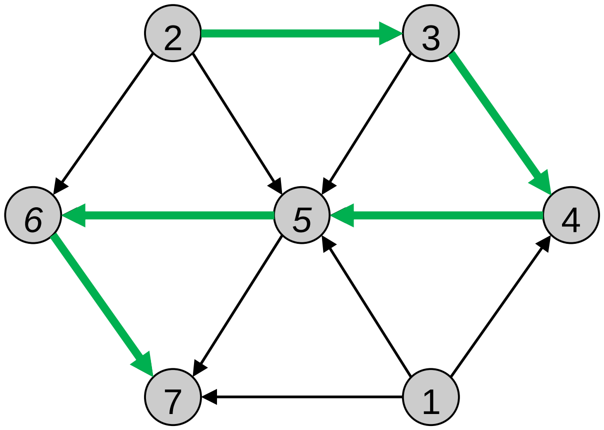
Longest Path in a DAG

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



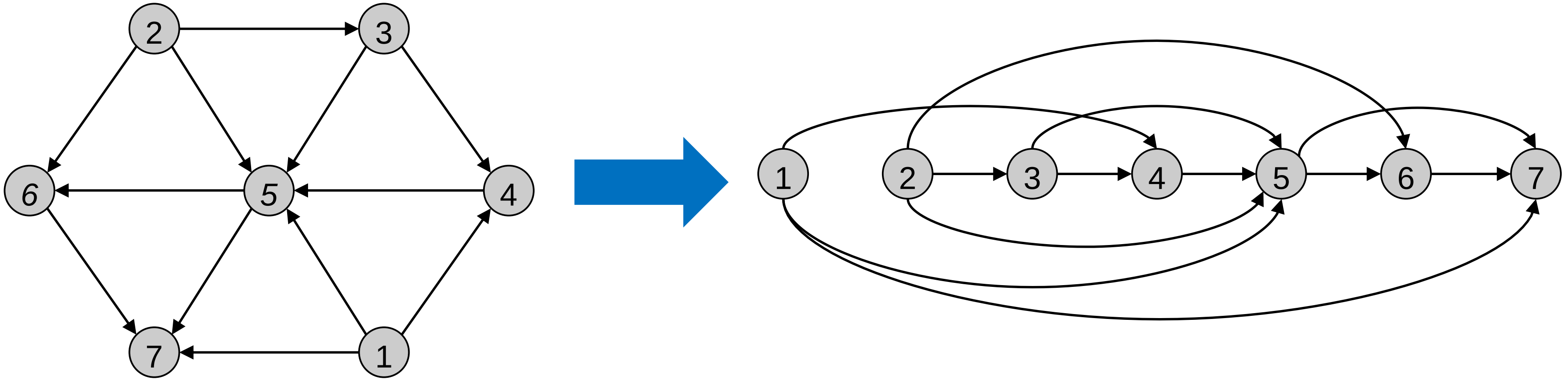
DP for Longest Path in a DAG

Q: What is the right ordering?

Remember, we have to use that G is a DAG, ideally in defining the ordering

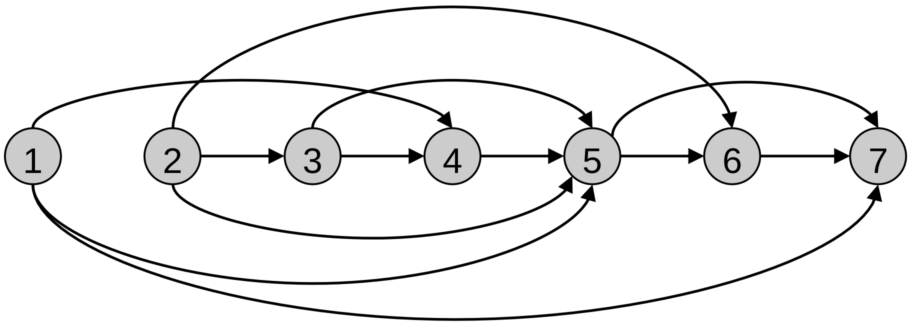
We saw that every DAG has a topological sorting

So, let's use that as an ordering.



DP for Longest Path in a DAG

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 $(i_1, i_2), (i_2, i_3), \dots, (i_{k-1}, i_k), (i_k, j)$, then

Obs 1: $i_1 \leq i_2 \leq \cdots \leq i_k \leq j$.

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

$$OPT(j) = 1 + OPT(i_k).$$

DP for Longest Path in a DAG

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

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

Outputting the Longest Path

```
Let G be a DAG given with a topological sorting: For all edges
(i,j) we have i<j.
Initialize 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
   return M[j]
}
Let M[k] be the maximum of M[1],…,M[n]
While (Parent[k]!=-1)
   Print k
   k=Parent[k]
```

Record the entry that
we used to compute OPT(j)