

# *Algorithm Design and Analysis*

---

**CSE  
565**

## **LECTURE 19**

### **Dynamic Programming**

- Knapsack problem
- RNA Secondary Structure

**Adam Smith**

# Review questions

- Shortest-path( $G,s,t$ ) = shortest route from  $s$  to  $t$ .
- Longest-path( $G,s,t$ ) = longest *simple* path from  $s$  to  $t$
- **Question:** Do Shortest Path and Longest Path have optimal substructure that can be used for a poly-time dynamic programming algorithm?
  - What if the graph is a DAG?

# 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:  $\{ 3, 4 \}$  has value 40.

$W = 11$

- Many “packing” problems fit this model
  - Assigning production jobs to factories
  - Deciding which jobs to do on a single processor with bounded time
  - Deciding which problems to do on an exam

#	value	weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

# 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:  $\{ 3, 4 \}$  has value 40.

$W = 11$

- **Greedy:** repeatedly add item with maximum ratio  $v_i / w_i$
- Example:  $\{ 5, 2, 1 \}$  achieves only value = 35  $\Rightarrow$  greedy not optimal.

#	value	weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

# 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:  $\{ 3, 4 \}$  has value 40.

$W = 11$

- **Greedy algorithm?**

#	value	weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

# Dynamic programming: attempt 1

- **Definition:**  $\text{OPT}(i) =$ 
  - maximum profit subset of items  $1, \dots, i$ .
- **Case 1:** OPT does not select item  $i$ .
  - OPT selects best of  $\{ 1, 2, \dots, i-1 \}$
- **Case 2:** OPT selects item  $i$ .
  - without knowing what other items were selected before  $i$ , we don't even know if we have enough room for  $i$
- **Conclusion.** Need more sub-problems!

# Adding a new variable

- **Definition:**  $OPT(i, w)$  = max profit subset of items  $1, \dots, i$  **with weight limit  $w$ .**
  - **Case 1:**  $OPT$  does not select item  $i$ .
    - $OPT$  selects best of  $\{ 1, 2, \dots, i-1 \}$  with weight limit  $w$
  - **Case 2:**  $OPT$  selects item  $i$ .
    - new weight limit =  $w - w_i$
    - $OPT$  selects best of  $\{ 1, 2, \dots, i-1 \}$  with new weight limit

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

# Bottom-up algorithm

- Fill up an  $n$ -by- $W$  array.

```
Input:  $n, W, w_1, \dots, w_N, v_1, \dots, v_N$ 

for  $w = 0$  to  $W$ 
     $M[0, w] = 0$ 

for  $i = 1$  to  $n$ 
    for  $w = 1$  to  $W$ 
        if ( $w_i > w$ )
             $M[i, w] = M[i-1, w]$ 
        else
             $M[i, w] = \max \{M[i-1, w], v_i + M[i-1, w-w_i]\}$ 

return  $M[n, W]$ 
```

# Knapsack table

		W →											
		0	1	2	3	4	5	6	7	8	9	10	11
n ↓	$\phi$	0	0	0	0	0	0	0	0	0	0	0	0
	{1}	0	1	1	1	1	1	1	1	1	1	1	1
	{1, 2}	0	1	6	7	7	7	7	7	7	7	7	7
	{1, 2, 3}	0	1	6	7	7	18	19	24	25	25	25	25
	{1, 2, 3, 4}	0	1	6	7	7	18	22	24	28	29	29	40
	{1, 2, 3, 4, 5}	0	1	6	7	7	18	22	28	29	34	34	40

W = 11

OPT: {4, 3}  
value = 22 + 18 = 40

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

# How do we make turn this into a proof of correctness?

- Dynamic programming (and divide and conquer) lends itself directly to induction.
  - Base cases:  $\text{OPT}(i,0)=0$ ,  $\text{OPT}(0,w)=0$  (no items!).
  - Inductive step: explaining the correctness of recursive formula
    - If the following values are correctly computed:
      - $\text{OPT}(0,w-1), \text{OPT}(1,w-1), \dots, \text{OPT}(n,w-1)$
      - $\text{OPT}(0,w), \dots, \text{OPT}(i-1,w)$
    - Then the recursive formula computes  $\text{OPT}(i,w)$  correctly
      - Case 1: ..., Case 2: ... (from previous slide).

# About proofs

- Proof is a rigorous argument about a mathematical statement's truth
  - Should convince you
  - Should not feel like a shot in the dark
  - What makes a proof “good enough” is social
  - (Truly rigorous, machine-readable proofs exist but are too painful for human use).
- In real life, you have to check your own work
  - Ideal: all problems should have grades 100% or 15%

# Time and Space Complexity

- 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.

What is the input size?

- In words?
- In bits?

$W = 11$

#	value	weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

# Time and space complexity

- **Time and space:**  $\Theta(n W)$ .
  - Not polynomial in input size!
  - "Pseudo-polynomial."
  - Decision version of Knapsack is NP-complete.

[KT, chapter 8]

- **Knapsack approximation algorithm.** There is a poly-time algorithm that produces a solution with value within 0.01% of optimum. [KT, section 11.8]



# RNA Secondary Structure

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

- [Watson-Crick.]  $S$  is a matching and each pair in  $S$  is a Watson-Crick complement: 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  $i < k < j < l$ .

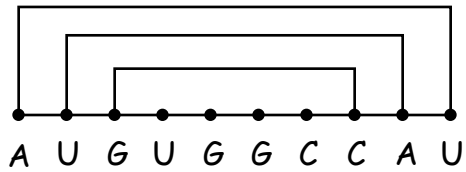
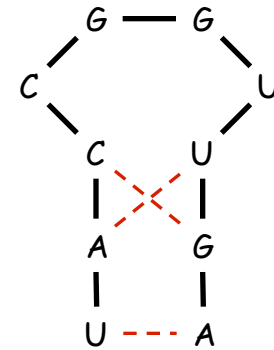
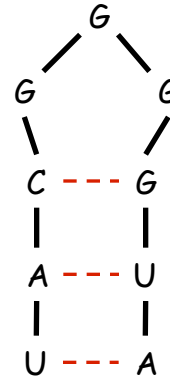
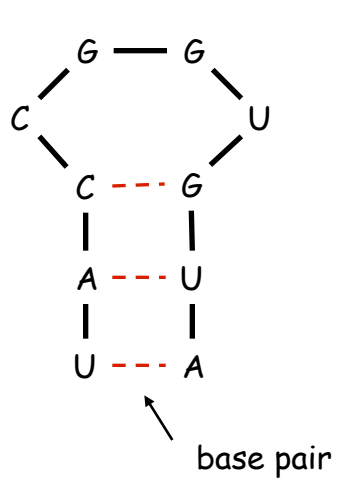
Free energy. Usual hypothesis is that an RNA molecule will form the secondary structure with the optimum total free energy.

↑  
approximate by number of base pairs

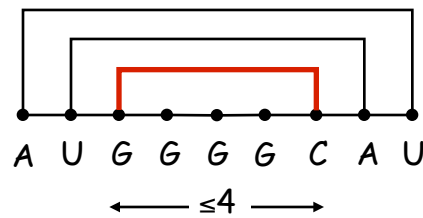
Goal. Given an RNA molecule  $B = b_1b_2\dots b_n$ , find a secondary structure  $S$  that maximizes the number of base pairs.

# RNA Secondary Structure: Examples

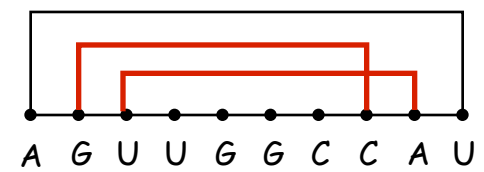
Examples.



ok



sharp turn

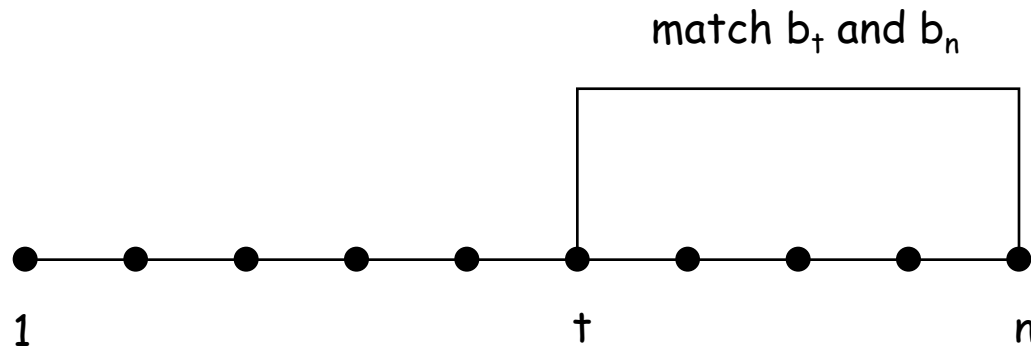


crossing

# RNA Secondary Structure: Subproblems

First attempt.  $OPT(j) =$

maximum number of base pairs in a secondary structure of the substring  $b_1b_2\dots b_j$ .



Difficulty. Results in two sub-problems.

- Finding secondary structure in:  $b_1b_2\dots b_{t-1}$ . ←  $OPT(t-1)$
- Finding secondary structure in:  $b_{t+1}b_{t+2}\dots b_{n-1}$ . ← need more sub-problems

# Dynamic Programming Over Intervals

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

- Base case. If  $i \geq j - 4$ .
  - $OPT(i, j) = 0$  by no-sharp turns condition.
- Case 1. Base  $b_j$  is not involved in a pair.
  - $OPT(i, j) = OPT(i, j-1)$
- Case 2. Base  $b_j$  pairs with  $b_t$  for some  $i \leq t < j - 4$ .
  - non-crossing constraint decouples resulting sub-problems
  - $OPT(i, j) = 1 + \max_t \{ OPT(i, t-1) + OPT(t+1, j-1) \}$

↑  
take max over  $t$  such that  $i \leq t < j-4$  and  $b_t$  and  $b_j$  are Watson-Crick complements

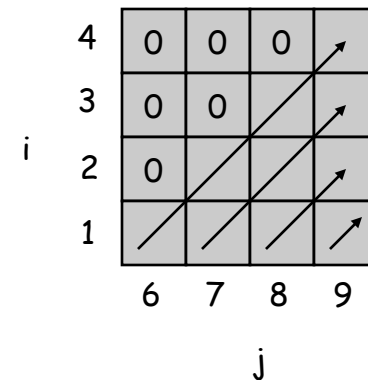
## Bottom Up Dynamic Programming Over Intervals

Q. What order to solve the sub-problems?

A. Do shortest intervals first.

```
RNA( $b_1, \dots, b_n$ ) { %Ignoring base cases
  for  $k = 5, 6, \dots, n-1$ 
    for  $i = 1, 2, \dots, n-k$ 
       $j = i + k$ 
      Compute  $M[i, j]$ 
    }
  return  $M[1, n]$ 
}
```

using recurrence



Time:  $O(n^3)$ .

Space:  $O(n^2)$ .

**Exercise:** Find the secondary structure that achieves the max value.

# Dynamic Programming Summary

## Recipe.

- Recursively define value of optimal solution.
- Compute value of optimal solution.
- Construct optimal solution from computed information.

## Dynamic programming techniques.

- Binary choice: weighted interval scheduling.
- Multi-way choice: segmented least squares. [KT, section 6.3]
- Adding a new variable: LCS, knapsack.
- Dynamic programming over intervals: RNA secondary structure.

↖ parsing algorithm for context-free grammar has similar structure

Top-down (memoization) vs. bottom-up: different people have different intuitions.