南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Algorithm Design and Analysis (H)

CS216

Prof. Shiqi Yu (于仕琪)

yusq@sustech.edu.cn

http://faculty.sustech.edu.cn/yusq/

The power of $O(K^n)$

Rice on a chessboard

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Dynamic Programming

3. Knapsack Problem

Knapsack Problem

- 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

#	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.

Dynamic Programming: False Start

- Def. OPT(i) = max profit subset of items 1, …, i.
  - Case 1: OPT does not select item i.
    - ✓ OPT selects best of { 1, 2, …, i-1 }
  - Case 2: OPT selects item i.
    - ✓ accepting item i does not immediately imply that we will have to reject other items
    - ✓ 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!

Dynamic Programming: Adding a New Variable

- Def. OPT(i, w) = max profit subset of items 1, …, i with weight limit w.

  - Case 1: OPT does not select item i.
    - ✓ OPT selects best of { 1, 2, …, i-1 } using weight limit w

  - Case 2: OPT selects item i.
    - ✓ new weight limit = $w - w_i$
    - ✓ OPT selects best of { 1, 2, …, i–1 } using this new weight limit

$$\mathrm{OPT}(i, w) = \max\left(OPT(i-1, w),\ v_i + OPT(i-1, w - w_i)\right)$$

Knapsack Problem: Bottom-Up

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

```
Input: n, W, w1,…,wN, v1,…,vN

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

for i = 1 to n
   for w = 1 to W
      if (wi > w)
         M[i, w] = M[i-1, w]
      else
         M[i, w] = max {M[i-1, w], vi + M[i-1, w-wi ]}

return M[n, W]
```

$$\mathrm{OPT}(i, w) = \max\left(OPT(i-1, w),\ v_i + OPT(i-1, w-w_i)\right)$$

Knapsack Algorithm

W + 1 →

n + 1 ↓

	0	1	2	3	4	5	6	7	8	9	10	11
ϕ	0	0	0	0	0	0	0	0	0	0	0	0
{ 1 }	0	$1_{\{1\}}$	$1_{\{1\}}$	$1_{\{1\}}$	$1_{\{1\}}$	$1_{\{1\}}$	$1_{\{1\}}$	$1_{\{1\}}$	$1_{\{1\}}$	$1_{\{1\}}$	$1_{\{1\}}$	$1_{\{1\}}$
{ 1, 2 }	0	$1_{\{1\}}$	$6_{\{2\}}$	$7_{\{1,2\}}$	$7_{\{1,2\}}$	$7_{\{1,2\}}$	$7_{\{1,2\}}$	$7_{\{1,2\}}$	$7_{\{1,2\}}$	$7_{\{1,2\}}$	$7_{\{1,2\}}$	$7_{\{1,2\}}$
{ 1, 2, 3 }	0	$1_{\{1\}}$	$6_{\{2\}}$	$7_{\{1,2\}}$	$7_{\{1,2\}}$	$18_{\{3\}}$	$19_{\{1,3\}}$	$24_{\{2,3\}}$	$25_{\{1,2,3\}}$	$25_{\{1,2,3\}}$	$25_{\{1,2,3\}}$	$25_{\{1,2,3\}}$
{ 1, 2, 3, 4 }	0	$1_{\{1\}}$	$6_{\{2\}}$	$7_{\{1,2\}}$	$7_{\{1,2\}}$	$18_{\{3\}}$	$22_{\{4\}}$	$24_{\{2,3\}}$	$28_{\{2,4\}}$	$29_{\{1,2,4\}}$	$29_{\{1,2,4\}}$	$40_{\{3,4\}}$
{ 1, 2, 3, 4, 5 }	0	$1_{\{1\}}$	$6_{\{2\}}$	$7_{\{1,2\}}$	$7_{\{1,2\}}$	$18_{\{3\}}$	$22_{\{4\}}$	$28_{\{5\}}$	$29_{\{1,5\}}$	$34_{\{2,5\}}$	$34_{\{2,5\}}$	$40_{\{3,4\}}$

W = 11

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

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

$$\mathrm{OPT}(i, w) = \max\left(OPT(i-1, w),\ v_i + OPT(i-1, w - w_i)\right)$$

Knapsack Problem: Running Time

- Running time. $\Theta(n\ W)$.
  - Not polynomial in input size!
  - "Pseudo-polynomial."
  - Decision version of Knapsack is NP-complete. [Chapter 8]
    - ✓ Can a value of at least V be achieved under a restriction of a certain capacity W?

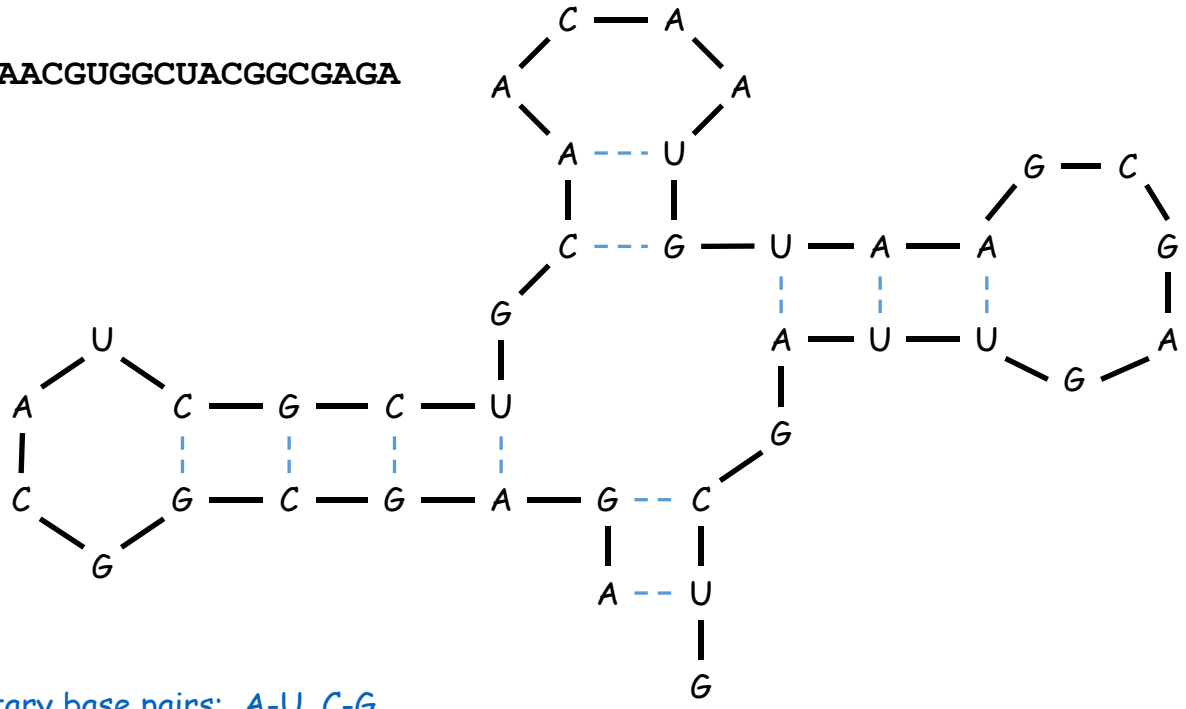
- Knapsack approximation algorithm. There exists a poly-time algorithm that produces a feasible solution that has value within 0.01% of optimum. [Section 11.8]

4 RNA Secondary Structure

RNA Secondary Structure

- RNA. 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

- 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 $i < k < j < l$.

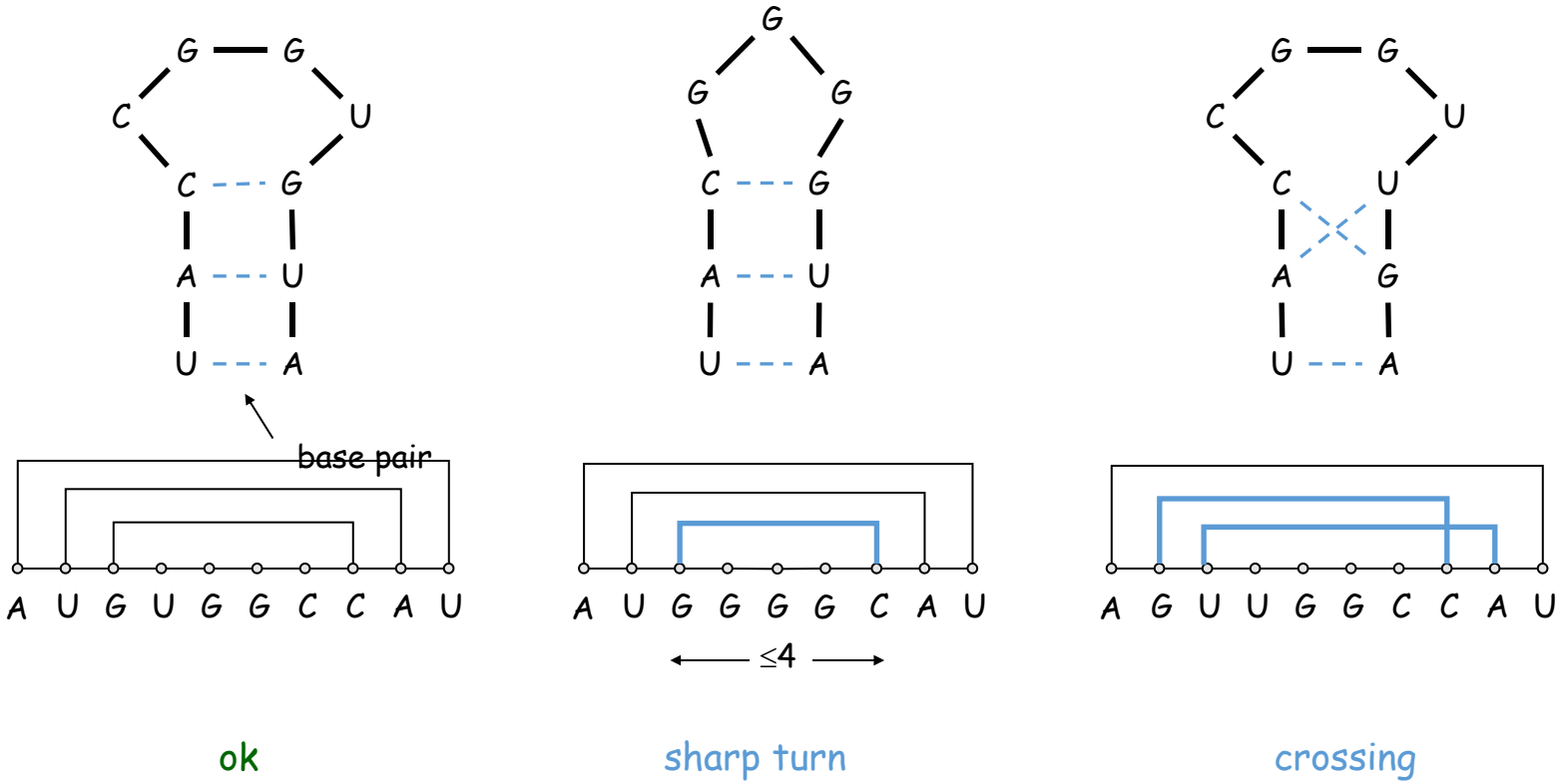
- Free energy. Usual hypothesis is that an RNA molecule will form the secondary structure with the optimum total free energy.

approximate by number of base pairs

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

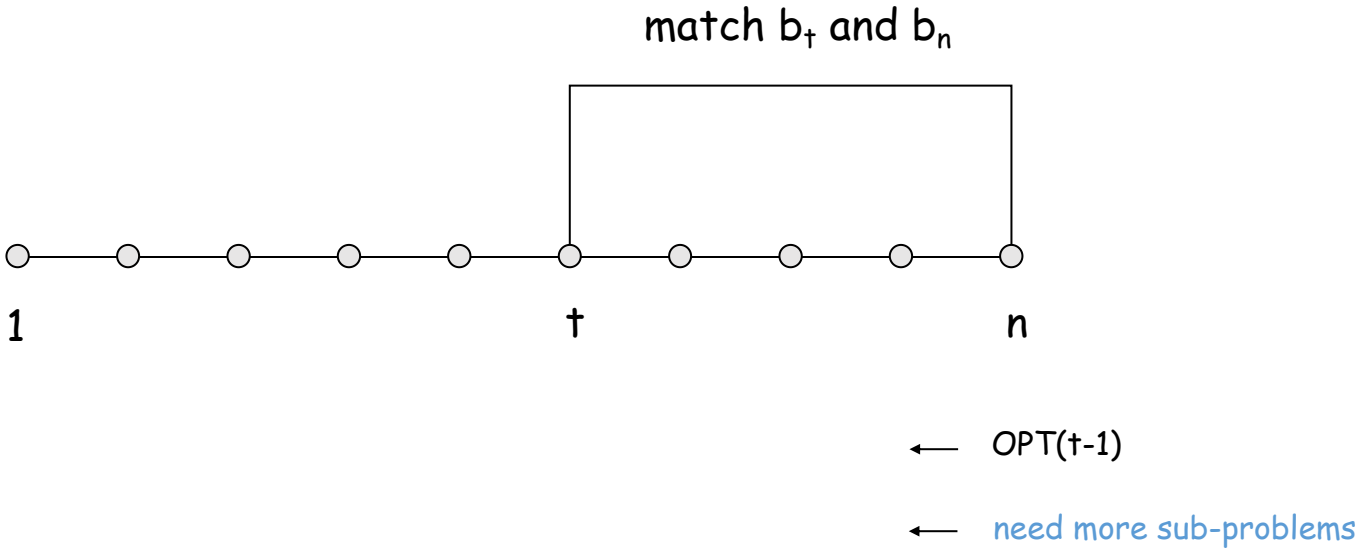
RNA Secondary Structure: Examples

- Examples.



RNA Secondary Structure: Subproblems

- First attempt. OPT(j) = maximum number of base pairs in a secondary structure of the substring $b_1 b_2 \ldots b_j$.



- Difficulty. Results in two sub-problems.
  - Finding secondary structure in: $b_1 b_2 \ldots b_{t-1}$.
  - Finding secondary structure in: $b_{t+1} b_{t+2} \ldots b_{n-1}$.

Dynamic Programming Over Intervals

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

  - Case 1. If $i \geq j - 4$.
    - ✓ 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 $i \leq t < j - 4$.
    - ✓ non-crossing constraint decouples resulting sub-problems
    - ✓ $\text{OPT}(i, j) = 1 + \max_t \{ \text{OPT}(i, t-1) + \text{OPT}(t+1, j-1) \}$

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

- Remark. Same core idea in CKY algorithm to parse context-free grammars.

Example

RNA sequence *ACCGGUAGU*

4	0	0	0	
3	0	0		
2	0			
$i = 1$				
	$j = 6$	7	8	9

Initial values

4	0	0	0	0
3	0	0	1	
2	0	0		
$i = 1$	1			
	$j = 6$	7	8	9

Filling in the values for $k = 5$

4	0	0	0	0
3	0	0	1	1
2	0	0	1	
$i = 1$	1	1		
	$j = 6$	7	8	9

Filling in the values for $k = 6$

4	0	0	0	0
3	0	0	1	1
2	0	0	1	1
$i = 1$	1	1	1	
	$j = 6$	7	8	9

Filling in the values for $k = 7$

4	0	0	0	0
3	0	0	1	1
2	0	0	1	1
$i = 1$	1	1	1	2
	$j = 6$	7	8	9

Filling in the values for $k = 8$

$$\mathrm{OPT}(i,j) = \begin{cases} 0, & \text{if } i \geq j-4 \\ \mathrm{OPT}(i,j-1), & b_j \text{ cannot be paired} \\ 1 + \max_t \{OPT(i,t-1) + OPT(t+1,j-1)\}, & b_j \text{ can be paired} \end{cases}$$

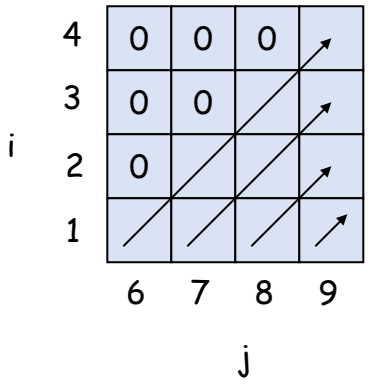
Bottom Up Dynamic Programming Over Intervals

- Q. What order to solve the sub-problems?
- A. Do shortest intervals first.

```
RNA(b1,…,bn) {
   for k = 5, 6, …, n-1
      for i = 1, 2, …, n-k
         j = i + k
         Compute M[i, j]

   return M[1, n]
}
```

using recurrence



$$\mathrm{OPT}(i,j) = \begin{cases} 0, & \text{if } i \geq j-4 \\ \mathrm{OPT}(i,j-1), & b_j \text{ cannot be paired} \\ 1 + \max_t \{OPT(i,t-1) + OPT(t+1,j-1)\}, & b_j \text{ can be paired} \end{cases}$$

- Running time. $O(n^3)$.

Dynamic Programming Summary

- Recipe.
  - Characterize structure of problem.
  - 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.
  - Adding a new variable: knapsack.
  - Dynamic programming over intervals: RNA secondary structure.

Viterbi algorithm for HMM also uses DP to optimize a maximum likelihood tradeoff between parsimony and accuracy

CKY parsing algorithm for context-free grammar has similar structure

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