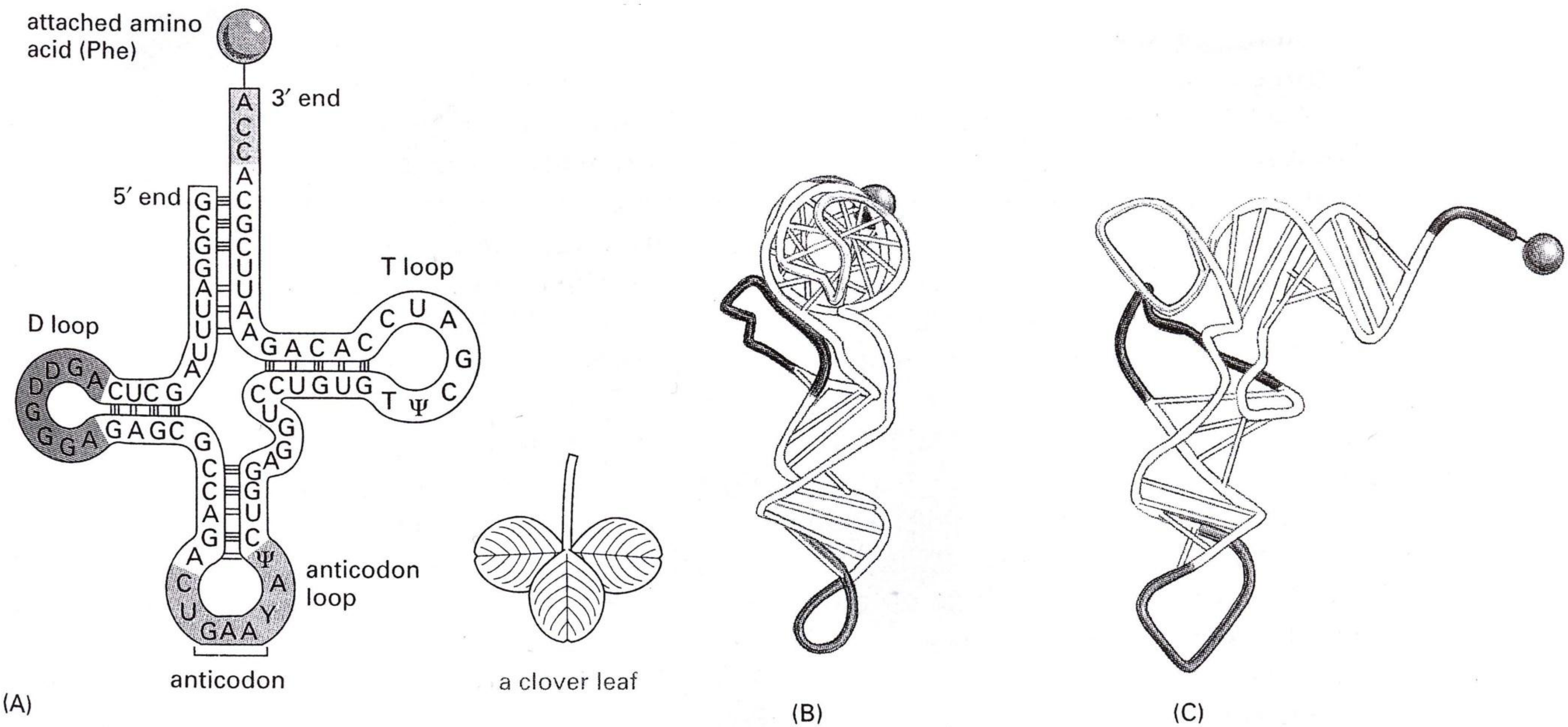
6.5 RNA Secondary Structure



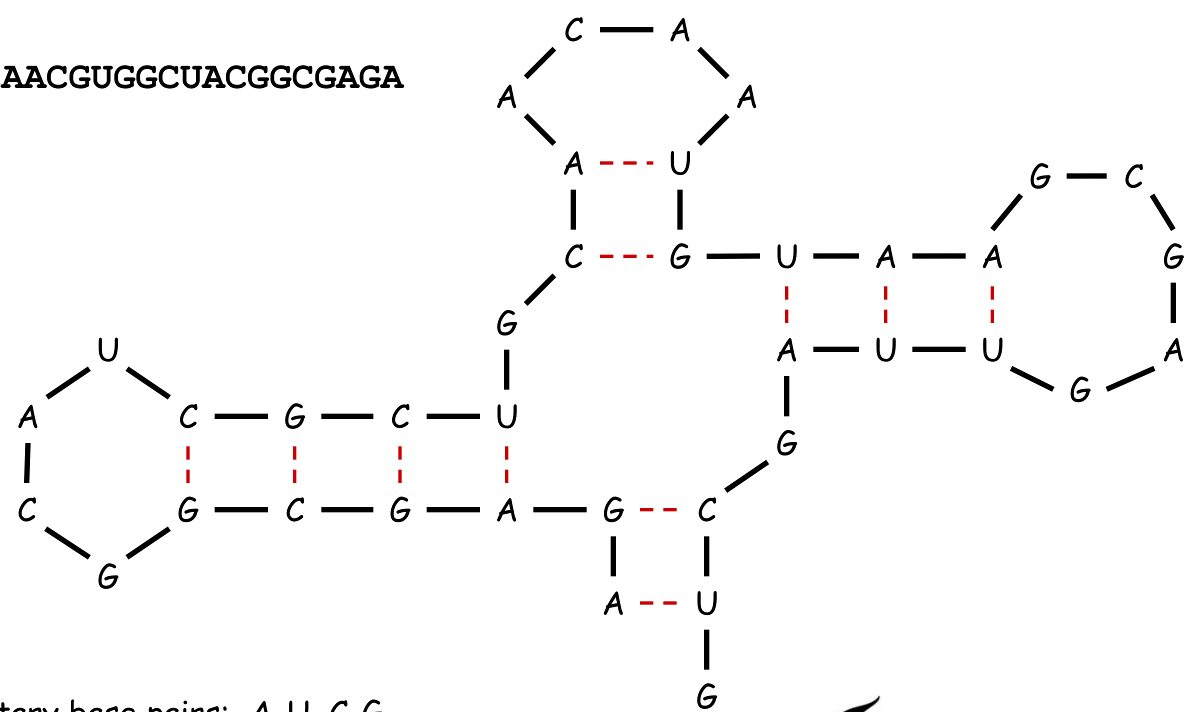
http://xray.bmc.uu.se/henke/Practical_4/practical_4.html

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 $i < k < j < l$.

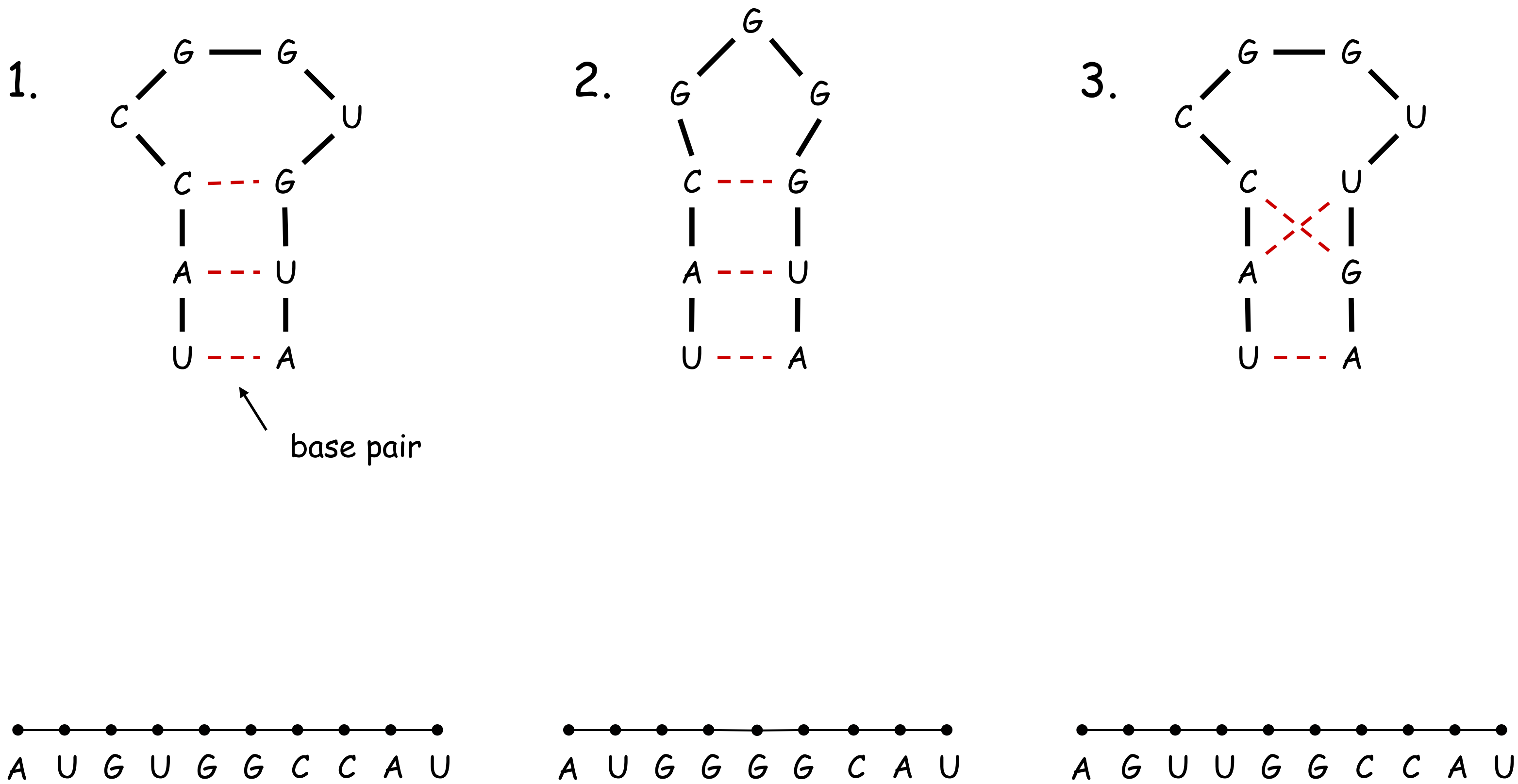
Free energy. Usual hypothesis is that an RNA molecule will form the secondary structure with the optimum total free energy.

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.

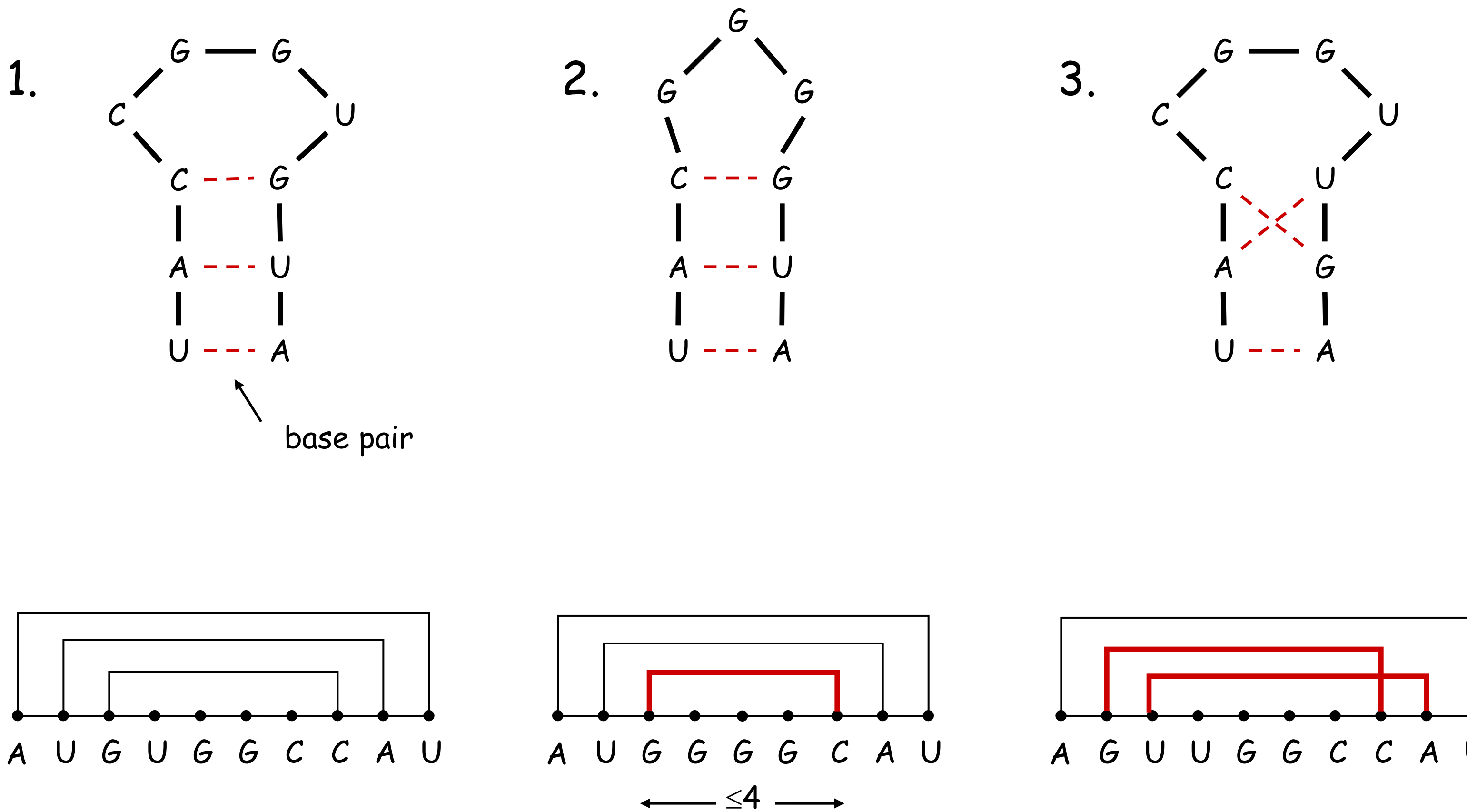
RNA Secondary Structure: Examples

Q. Are the following structures OK and why (not)?



RNA Secondary Structure: Examples

Q. Are the following structures OK and why (not)?



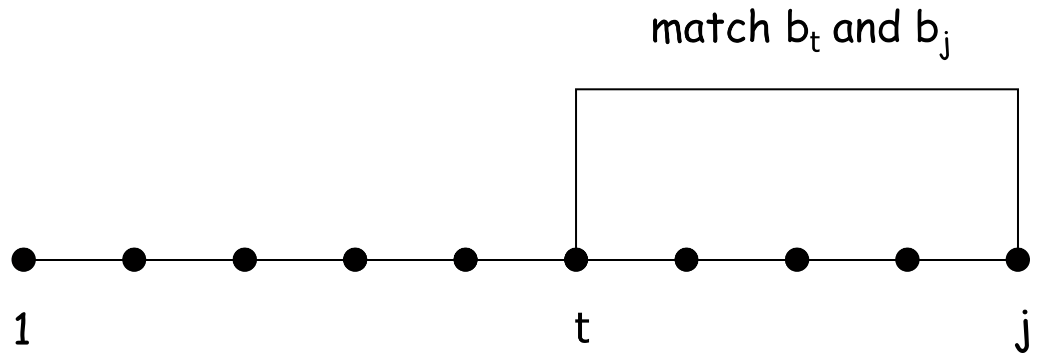
ok

sharp turn

crossing

RNA Secondary Structure: Subproblems

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



Q. What are our sub-problems?

- Finding secondary structure in: $b_1 b_2 \ldots b_{t-1}$. ← OPT(t-1)
- Finding secondary structure in: $b_{t+1} b_{t+2} \ldots b_{j-1}$. ← other type of sub-problem

So just a formula for OPT(j) is not enough!

Q. Which parameters do you need to express *any* sub-problem?

Q. And how to express the maximum number of pairs in terms of these sub-problems? (1 min)

Dynamic Programming Over Intervals

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

Q. What cases can we distinguish?

Dynamic Programming Over Intervals

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

Q. What cases can we distinguish?

A.

1. j cannot be involved in a pair, because i and j are too close
2. we choose to not pair j
3. we choose to pair j with another base t (which is its Watson-Crick complement and is more than 4 bases away)

Dynamic Programming Over Intervals

Recursively define value of optimal solution:

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 $i \geq j - 4$ (i and j too close)
  Q. How many base pairs are possible in this case?

- Case 2. We choose to let base b_j not be involved in a pair.

- Case 3. We choose to let base b_j pair with b_t for some $i \leq t < j - 4$.

Dynamic Programming Over Intervals

Recursively define value of optimal solution:

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

- Case 1. If $i \geq j - 4$ (i and j too close)
  - OPT(i, j) = 0 by no-sharp turns condition.

- Case 2. We choose to let base b_j not be involved in a pair.
  - Q. How many base pairs are possible in this case?

- Case 3. We choose to let base b_j pair with b_t for some $i \leq t < j - 4$.

TU Delft

Dynamic Programming Over Intervals

Recursively define value of optimal solution:

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 $i \geq j - 4$ (i and j too close)
  - OPT(i, j) = 0 by no-sharp turns condition.

- Case 2. We choose to let base b_j not be involved in a pair.
  - OPT(i, j) = OPT(i, j-1)

- Case 3. We choose to let base b_j pair with b_t for some $i \leq t < j - 4$.
  - Q. How many base pairs are possible in this case?

Dynamic Programming Over Intervals

Recursively define value of optimal solution:

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 $i \geq j - 4$ (i and j too close)
  - OPT(i, j) = 0 by no-sharp turns condition.

- Case 2. We choose to let base b_j not be involved in a pair.
  - OPT(i, j) = OPT(i, j-1)

- Case 3. We choose to let base b_j pair 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.

Dynamic Programming Over Intervals

Recursively define value of optimal solution:

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 $i \geq j - 4$ (i and j too close)
  - OPT(i, j) = 0 by no-sharp turns condition.
- Case 2. We choose to let base b_j not be involved in a pair.
  - OPT(i, j) = OPT(i, j-1)
- Case 3. We choose to let base b_j pair with b_t for some $i \leq t < j - 4$.
  - non-crossing constraint decouples resulting sub-problems
  - $OPT(i, j) = 1 + \max_t \{ OPT(i, t-1) + OPT(t+1, j-1) \}$

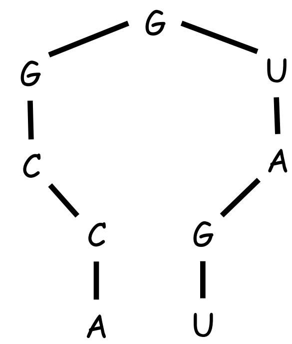
take max over t such that $i \leq t < j-4$ and b_t and b_j are Watson-Crick complements

$$OPT(i, j) = \begin{cases} 0 & \text{if } i \geq j - 4 \\ \max \left\{ OPT(i, j-1), 1 + \max\limits_{i \leq t < j-4 \text{ such that } b_t : b_j \text{ is a pair}} \{ OPT(i, t-1) + OPT(t+1, j-1) \} \right\} & \text{otherwise} \end{cases}$$

Bottom Up Dynamic Programming Over Intervals

Q. In what order to solve the sub-problems? (1 min)

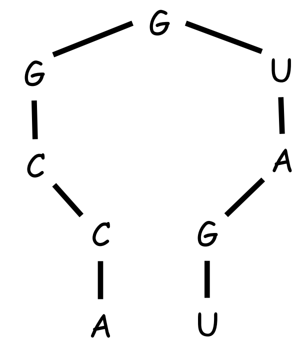
A.



j

$$OPT(i,j) = \begin{cases} 0 & \text{if } i \geq j-4 \\ \max\left\{ OPT(i, j-1), 1 + \max\limits_{i \leq t < j-4 \text{ such that } b_t : b_j \text{ is a pair}} \{OPT(i, t-1) + OPT(t+1, j-1)\} \right\} & \text{otherwise} \end{cases}$$

Bottom Up Dynamic Programming Over Intervals



Q. What order to solve the sub-problems? (1 min)

A. OPT(i,j) requires

- OPT(i,i) until OPT(i,j-1)
- OPT(i+1,j-1) until OPT(j-3,j-1)

Compute value of optimal solution iteratively.

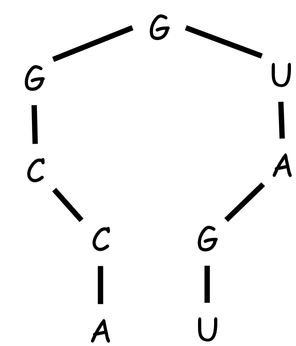
$$OPT(i,j) = \begin{cases} 0 & \text{if } i \geq j-4 \\ \max\left\{ OPT(i,j-1),\ 1 + \max\limits_{i \leq t < j-4 \text{ such that } b_t : b_j \text{ is a pair}} \{ OPT(i,t-1) + OPT(t+1,j-1) \} \right\} & \text{otherwise} \end{cases}$$

Bottom Up Dynamic Programming Over Intervals

Q. What order to solve the sub-problems? (1 min)

A. OPT(i,j) requires

- OPT(i,i) until OPT(i,j-1)
- OPT(i+1,j-1) until OPT(j-3,j-1)

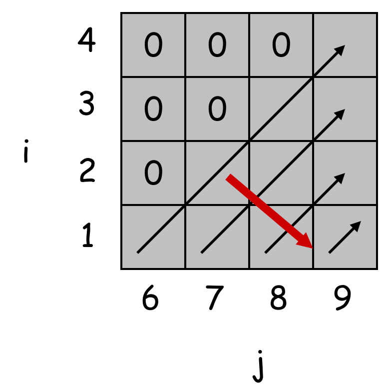


Compute value of optimal solution iteratively.

```
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



Q. What is the running time?

A.

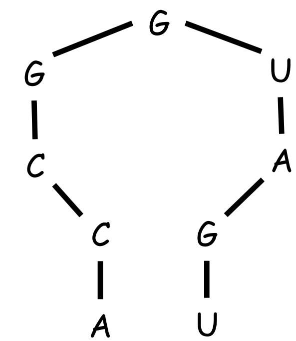
$$OPT(i, j) = \begin{cases} 0 & \text{if } i \geq j-4 \\ \max\left\{ OPT(i, j-1), 1 + \max\limits_{i \leq t < j-4 \text{ such that } b_t : b_j \text{ is a pair}} \{OPT(i, t-1) + OPT(t+1, j-1)\} \right\} & \text{otherwise} \end{cases}$$

Bottom Up Dynamic Programming Over Intervals

Q. What order to solve the sub-problems? (1 min)

A. OPT(i,j) requires

- OPT(i,i) until OPT(i,j-1)
- OPT(i+1,j-1) until OPT(j-3,j-1)

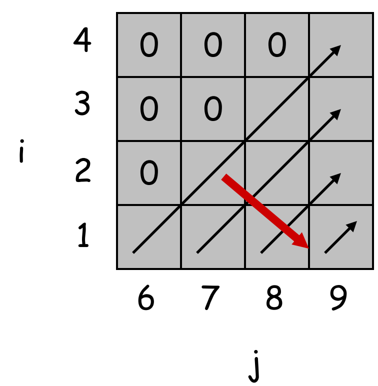


Compute value of optimal solution iteratively.

```
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



Q. What is the running time?

A. $O(n^3)$.

$$OPT(i,j) = \begin{cases} 0 & \text{if } i \geq j-4 \\ \max\left\{ OPT(i, j-1), 1 + \max\limits_{i \leq t < j-4 \text{ such that } b_t : b_j \text{ is a pair}} \{OPT(i, t-1) + OPT(t+1, j-1)\} \right\} & \text{otherwise} \end{cases}$$

Dynamic Programming Over Intervals: Finding a Solution

Construct optimal solution from computed information.

```
Run RNA()
Run Find-Solution(1,n)

Find-Solution(i,j) {
   if (i = 0 or w = 0)
      output nothing
   else if ( M[i,w] = M[i-1, w] )
         Find-Solution(i-1,w)
   else
         print i
         Find-Solution(i-1,w-wi)
}
```

Dynamic Programming Summary

Recipe.

1. Characterize structure of problem.
2. Recursively define value of optimal solution.
3. Compute value of optimal solution.
4. 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 (more subproblems): RNA secondary structure.

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

TU Delft