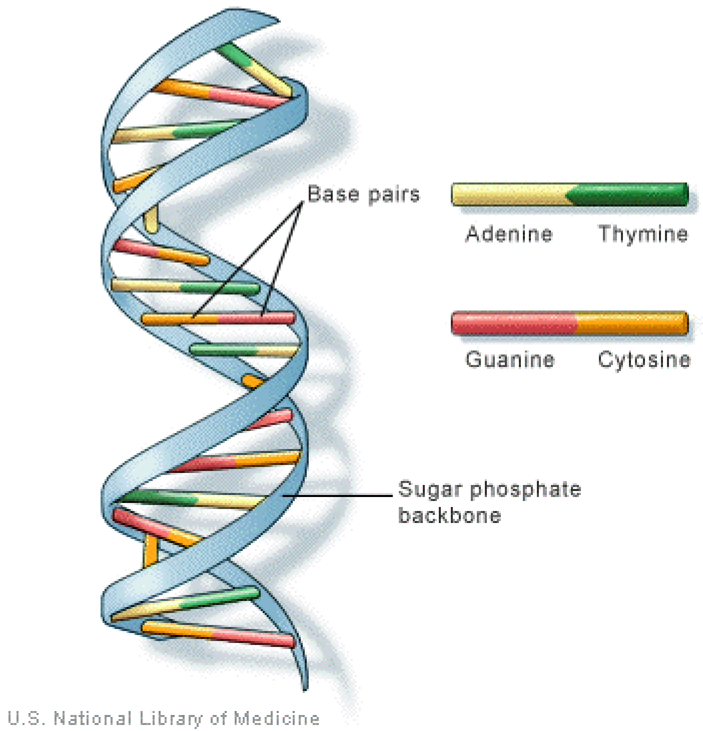
A problem from Computational Biology

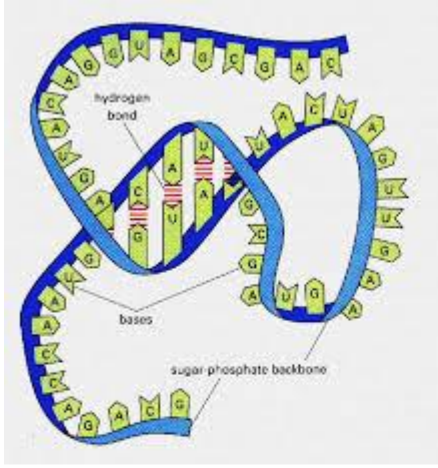
Basic Biology

- Watson and Crick gave the double stranded DNA model where two strands are zipped together by complementary base-pairing
- Bases {A,C, G, T}
- Pairing A-T, C-G

Courtesy:
ttp://www.chemguide.co.uk/organicprops/aminoacids/dna1.html



RNA



- RNA is a basic biological molecule. It is single stranded.
- RNA molecules fold into complex secondary structures.
- Secondary structure often governs the behaviour of an RNA molecule.
- Various rules govern secondary structure formation:

Problem

- Pairs of bases match up; each base matches with 1 other base.
- **A**denine always matches with **U**racil.
- **C**ytosine always matches with **G**uanine.
- There are no kinks in the folded molecule.
- Structures are knot-free.

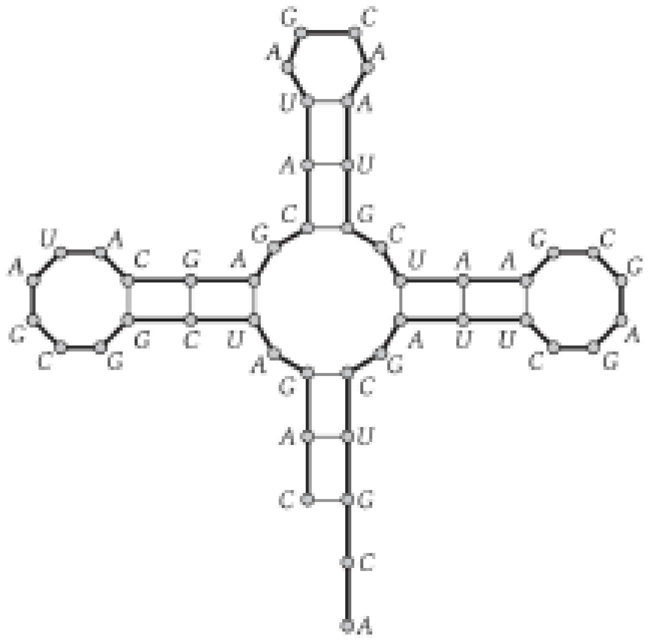


Figure 6.13 An RNA secondary structure. Thick lines connect adjacent elements of the sequence; thin lines indicate pairs of elements that are matched.

Problem: given an RNA molecule, predict its secondary structure.

Formulation

- An RNA molecule is a string B = $b_1b_2 \dots b_n$; each

$$b_i \in \{A, C, G, U\}$$

- An secondary structure on B is a set of pairs S = {(i , j)}, where $1 \le i , j \le n$ and satisfies the following rules.

Rules

- (No sharp turns) The ends of each pair are separated by at least 4 intervening bases i.e. if $(i , j) \in S$, then $i < j - 4$.
- The elements in each pair in S consist of either {A,U} or {C,G} (in either order).
- S is a matching: no base appears in more than one pair.
- (No knots) If (i,j) and (k,l) are two pairs in S, then we cannot have $i < k < j < l$.

The energy of a secondary structure is proportional to the number of base pairs in it.

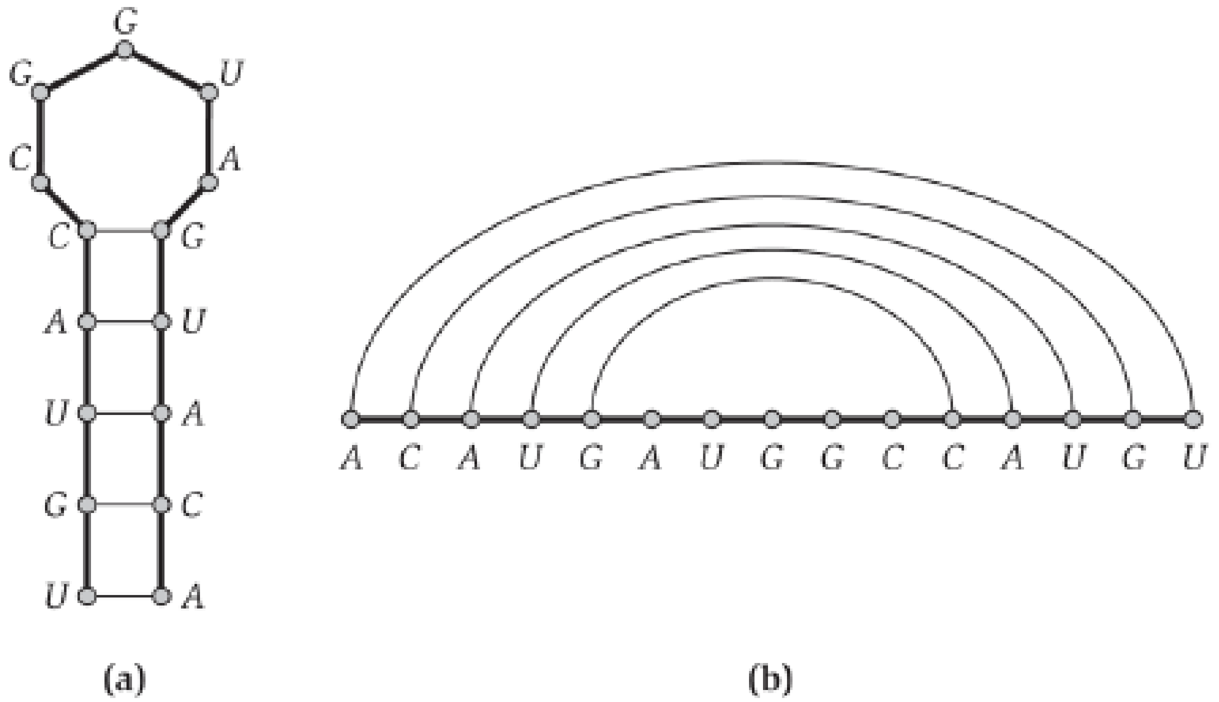


Figure 6.14 Two views of an RNA secondary structure. In the second view, (b), the string has been "stretched" lengthwise, and edges connecting matched pairs appear as noncrossing "bubbles" over the string.

First approach

- OPT(j) is the maximum number of base pairs in a secondary structure for $b_1b_2 \dots b_j$. OPT(j) = 0, if $j \leq 5$.
- In the optimal secondary structure on $b_1b_2 \dots b_j$
  - if j is not a member of any pair, use OPT(j-1).
  - if j pairs with some $t < j - 4$, knot condition yields two independent sub-problems! OPT(t -1) and ???

Pictorially

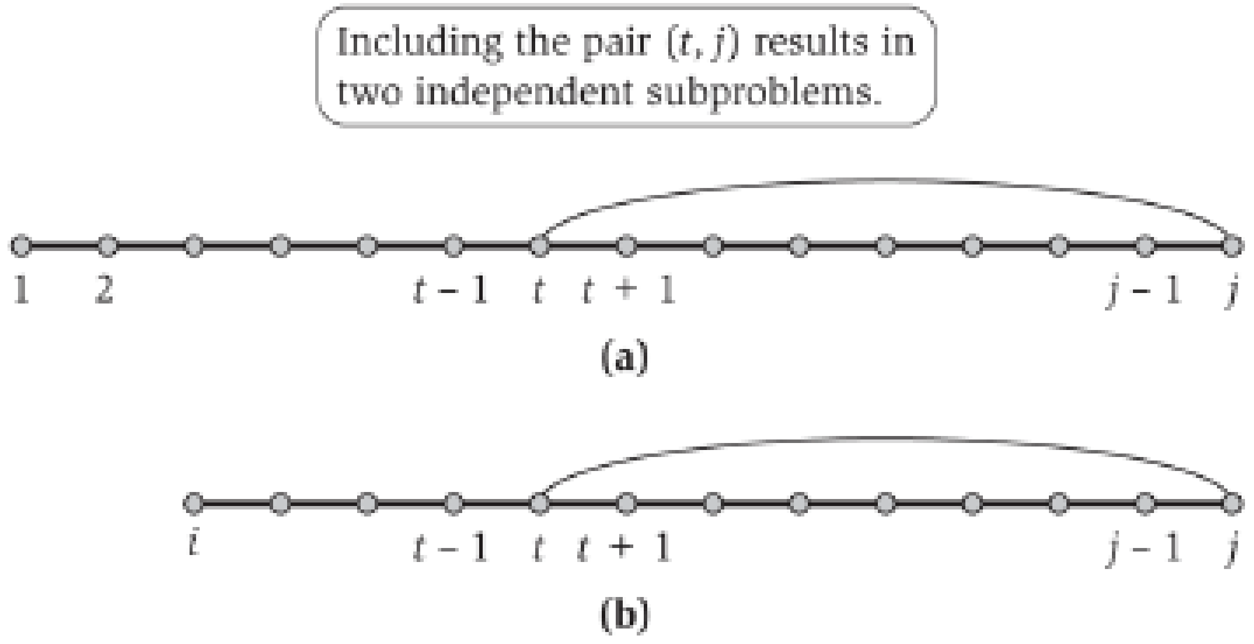

Including the pair (t, j) results in two independent subproblems.
1
2
$t-1$
t
$t+1$
$j-1$
j
(a)
i
$t-1$
t
$t+1$
$j-1$
j
(b)

Figure 6.15 Schematic views of the dynamic programming recurrence using (a) one variable, and (b) two variables.

Correct approach

- OPT(i,j) is the maximum number of base pairs in a secondary structure for $b_i b_{i+1} \ldots b_j$. OPT(i,j) = 0, if $i \geq j-4$.
- In the optimal secondary structure on $b_i b_{i+1} \ldots b_j$
  - if j is not a member of any pair, use OPT(i,j-1).
  - if j pairs with some $t < j - 4$, knot condition yields two independent sub-problems! OPT(i,t -1) and OPT(t+1,j-1)

Recurrence

$$OPT(i, j) = \max \begin{cases} OPT(i, j-1) \\ \max\limits_{t} (1 + OPT(i, t-1) + OPT(t+1, j-1)) \end{cases}$$

t ranges from I to j-5 such that it is allowed to pair with j

There are O(n^2) sub-problems.
How do we order them from “smallest" to “largest"?
Note that computing OPT(i , j) involves sub-problems OPT(l ,m) where m - l < j - i .

Recurrence

$$OPT(i, j) = \max \begin{cases} OPT(i, j-1) \\ \max_t (1 + OPT(i, t-1) + OPT(t+1, j-1)) \end{cases}$$

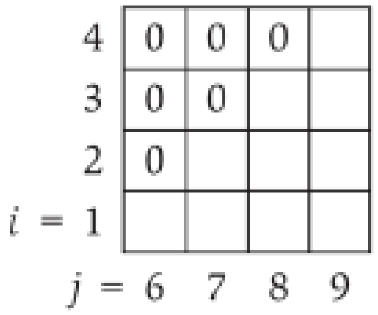
There are O(n^2) sub-problems.
How do we order them from “smallest" to “largest"?
Note that computing OPT(i , j) involves sub-problems OPT(l ,m)
where m - l < j - i .

Algorithm

```
Initialize $\text{OPT}(i,j)=0$ whenever $i \ge j-4$
For $k=5,\ 6,\ldots,n-1$
  For $i=1,2,\ldots n-k$
    Set $j=i+k$
    Compute $\text{OPT}(i,j)$ using the recurrence in (6.13)
  Endfor
Endfor
Return $\text{OPT}(1,n)$
```

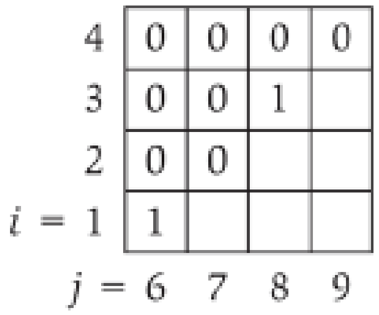
Lets do an example

- Input ACCGGUAGU



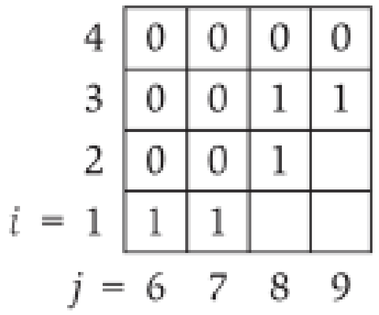
4	0	0	0	
3	0	0		
2	0			
i = 1				
	j = 6	7	8	9

Initial values



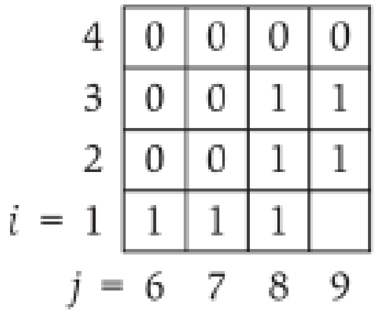
4	0	0	0	0
3	0	0	1	
2	0	0		
i = 1	1			
	j = 6	7	8	9

Filling in the values for k = 5



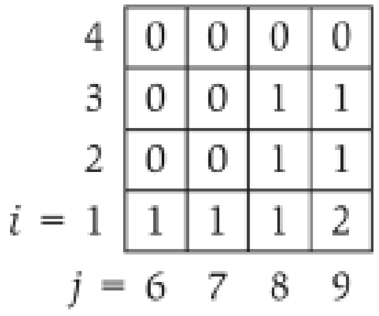
4	0	0	0	0
3	0	0	1	1
2	0	0	1	
i = 1	1	1		
	j = 6	7	8	9

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

Reference

- Algorithm Design by Eva Tardos and Jon Kleinberg.