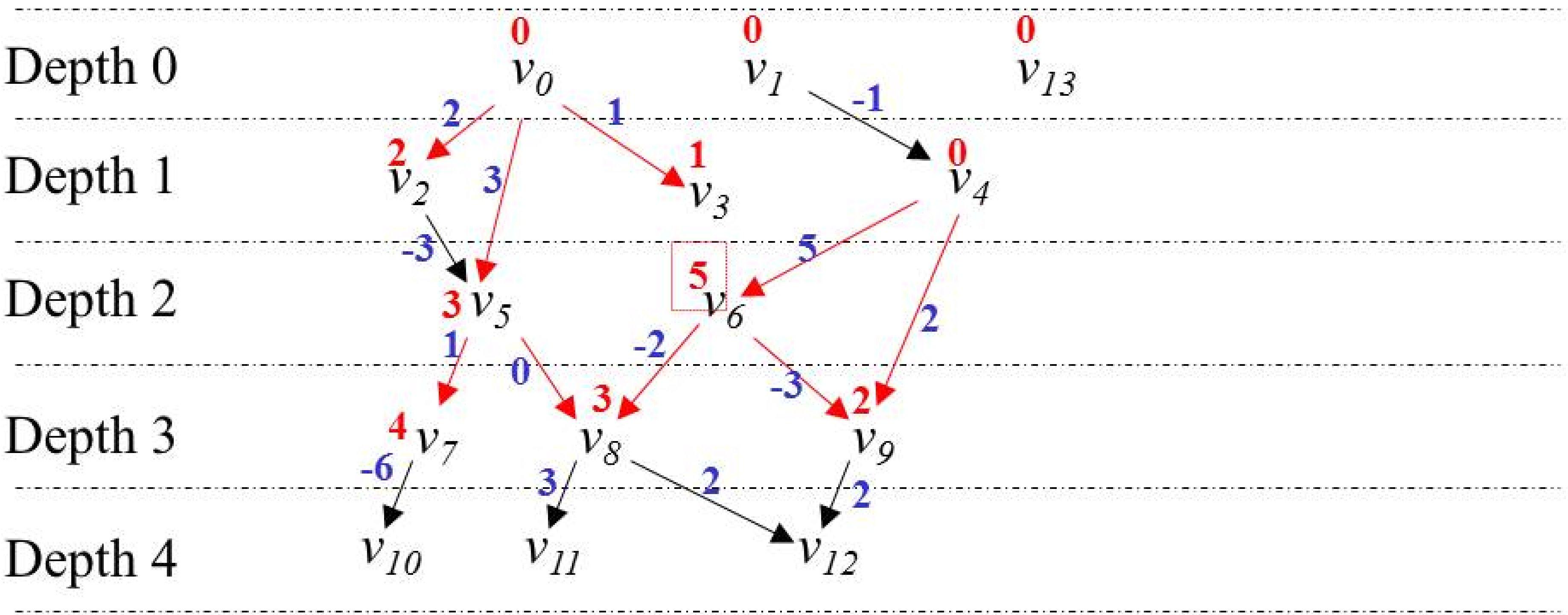
Discussion section #2

- HW1 questions?
- HW2: maximum-weight path on a DAG
- Shortest (minimum-weight) path algorithms
- Memoization

HW1 questions?

HW2: maximum-weight path on a DAG



HW2: maximum-weight path on a DAG

- Create input file with one line for each vertex and edge

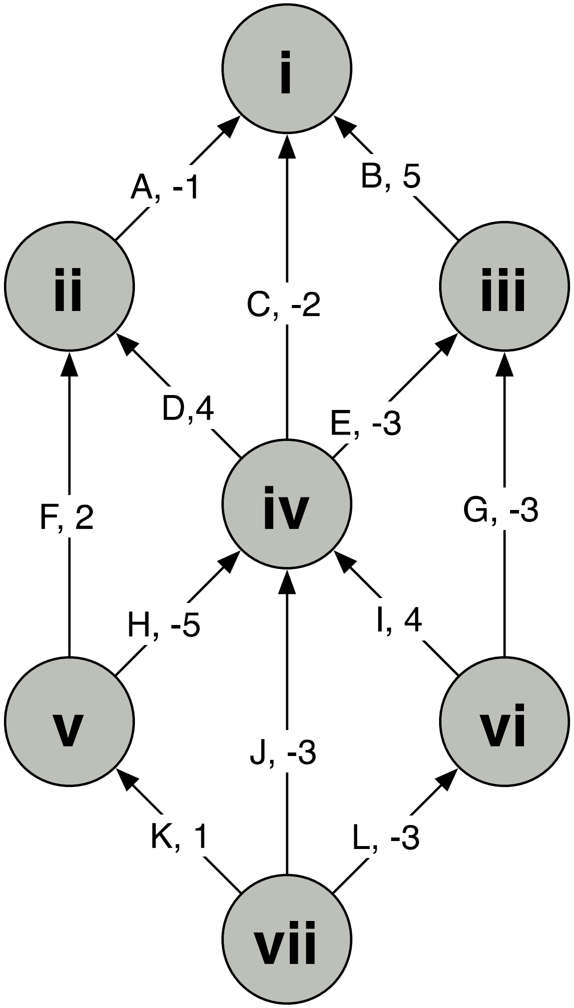
HW2: maximum-weight path on a DAG

- Create input file with one line for each vertex and edge
- Find the maximum-weight path on the graph

HW2: maximum-weight path on a DAG

- Create input file with one line for each vertex and edge
- Find the maximum-weight path on the graph
- Output:
  - Path length
  - Beginning and end vertex labels (positions)
  - The labels of all the edges on the path, in order

HW2: maximum-weight path on a DAG



Minimum-weight path on a DAG

- Similar to the homework
  - Looking for the minimum instead of the maximum
  - If no negative weights, then shortest path is technically weight 0
  - Otherwise, update vertex weights in depth order as normal

Minimum-weight path with cycles?

- No depth order to follow

Minimum-weight path with cycles?

- No depth order to follow
- Bellman-Ford algorithm (for a given start vertex)

Minimum-weight path with cycles?

- No depth order to follow
- Bellman-Ford algorithm (for a given start vertex)
  - Set start vertex distance to 0

Minimum-weight path with cycles?

- No depth order to follow
- Bellman-Ford algorithm (for a given start vertex)
  - Set start vertex distance to 0
  - All other vertex distances are infinity

Minimum-weight path with cycles?

- No depth order to follow
- Bellman-Ford algorithm (for a given start vertex)
  - Set start vertex distance to 0
  - All other vertex distances are infinity
  - For each edge (u, v), if v's distance can be reduced by taking that edge, update v's distance

Minimum-weight path with cycles?

- No depth order to follow
- Bellman-Ford algorithm (for a given start vertex)
  - Set start vertex distance to 0
  - All other vertex distances are infinity
  - For each edge (u, v), if v's distance can be reduced by taking that edge, update v's distance
  - Repeat |V| - 1 times

Minimum-weight path with cycles?

- No depth order to follow
- Bellman-Ford algorithm (for a given start vertex)
  - Set start vertex distance to 0
  - All other vertex distances are infinity
  - For each edge (u, v), if v's distance can be reduced by taking that edge, update v's distance
  - Repeat |V| - 1 times
  - How would you check for a negative cycle?

Minimum-weight path with cycles?

- No depth order to follow
- Bellman-Ford algorithm (for a given start vertex)
  - Set start vertex distance to 0
  - All other vertex distances are infinity
  - For each edge (u, v), if v's distance can be reduced by taking that edge, update v's distance
  - Repeat |V| - 1 times
  - How would you check for a negative cycle?
  - What about checking all paths?

Minimum-weight path with no negative edges?

Minimum-weight path with no negative edges?

- Djikstra's algorithm (for a given start vertex)

Minimum-weight path with no negative edges?

- Djikstra's algorithm (for a given start vertex)
  - Set start vertex distance to 0

Minimum-weight path with no negative edges?

- Djikstra's algorithm (for a given start vertex)
  - Set start vertex distance to 0
  - All other vertex distances are infininty

Minimum-weight path with no negative edges?

- Djikstra's algorithm (for a given start vertex)
  - Set start vertex distance to 0
  - All other vertex distances are infininty
  - Which vertex do we know the minimum-weight path to?

Minimum-weight path with no negative edges?

- Djikstra's algorithm (for a given start vertex)
  - Set start vertex distance to 0
  - All other vertex distances are infininty
  - Which vertex do we know the minimum-weight path to?
  - Do we ever need to update a vertex more than once?

Memoization
(similar to dynamic programming)

Memoization
(similar to dynamic programming)

- Dynamic programming
  - Can imagine filling a table of values
  - Bottom-up approach

Memoization
(similar to dynamic programming)

- Dynamic programming
  - Can imagine filling a table of values
  - Bottom-up approach

- Memoization

Memoization (similar to dynamic programming)

- Dynamic programming
  - Can imagine filling a table of values
  - Bottom-up approach
- Memoization
  - Makes sense from a recursive standpoint
  - Top-down approach

Memoization
(similar to dynamic programming)

- Dynamic programming
  - Can imagine filling a table of values
  - Bottom-up approach

- Memoization
  - Makes sense from a recursive standpoint
  - Top-down approach
  - Some scenarios where more intuitive

RNA folding

AUGCUAUAUAAACGCGAUACUAUACGCGAUAAUCGCGCGAGA

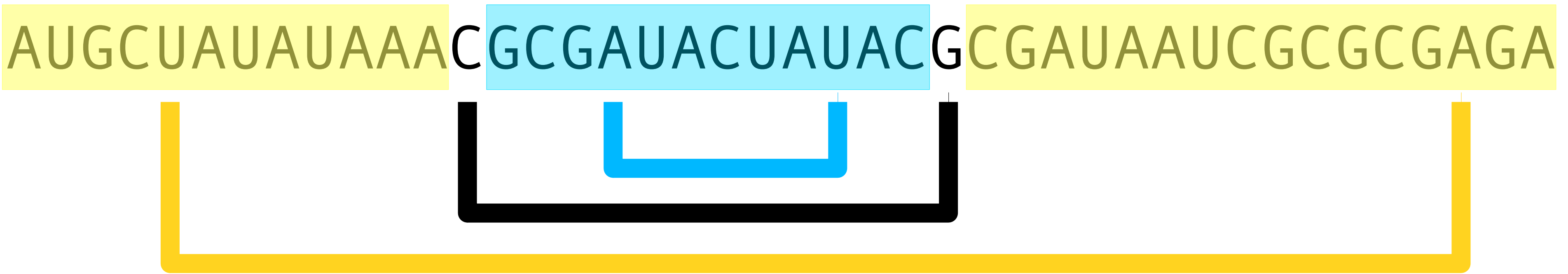
RNA folding

AUGCUAUAUAAACGCGAUACUAUACGCGAUAAUCGCGCGAGA

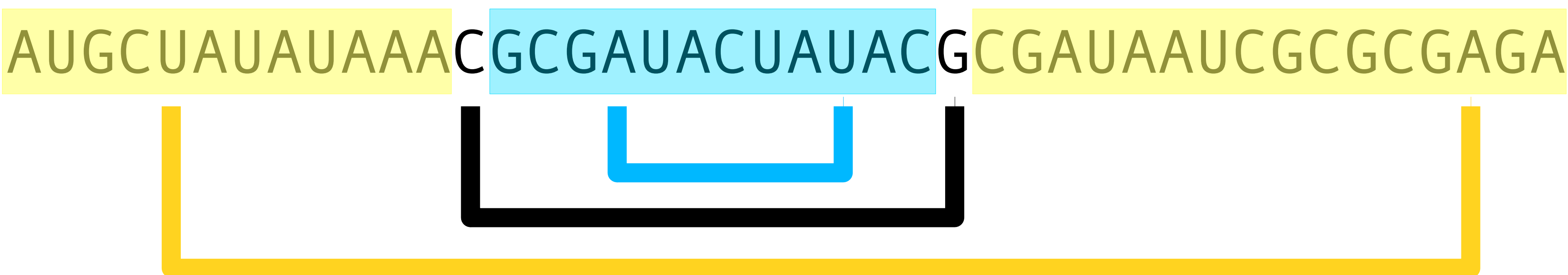
RNA folding

AUGCUAUAUAAACGCGAUACUAUACGCGAUAAUCGCGAGA

RNA folding

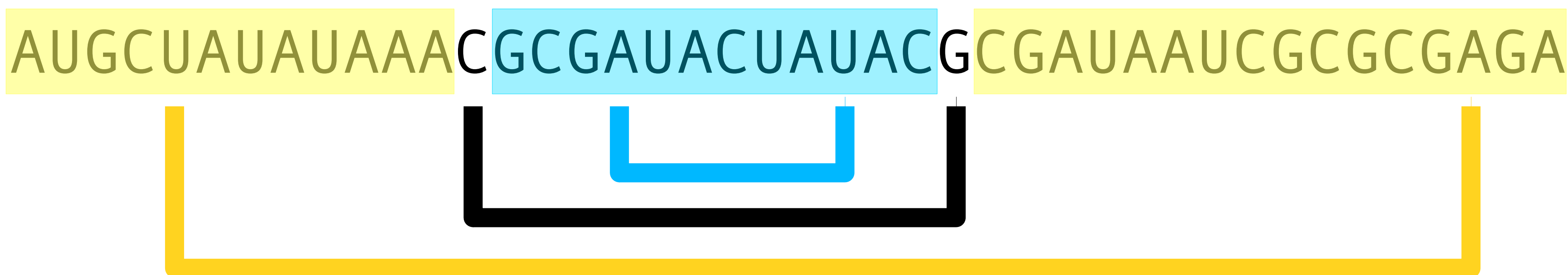


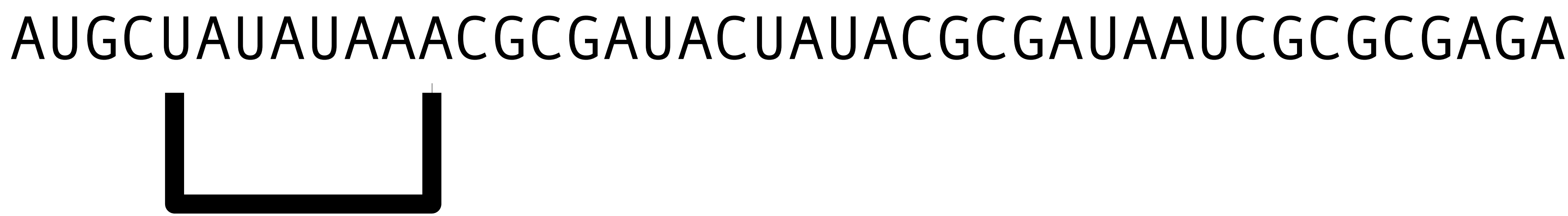
RNA folding



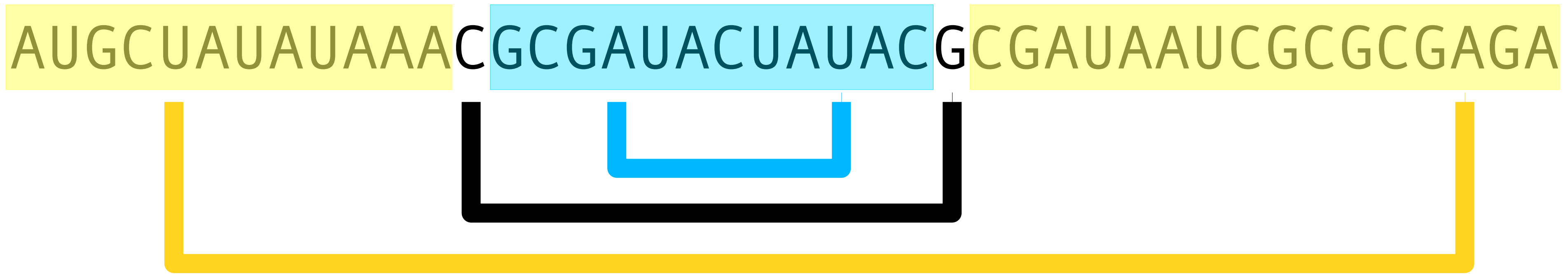
AUGCUAUAUAAACGCGAUACUAUACGCGAUAAUCGCGCGAGA

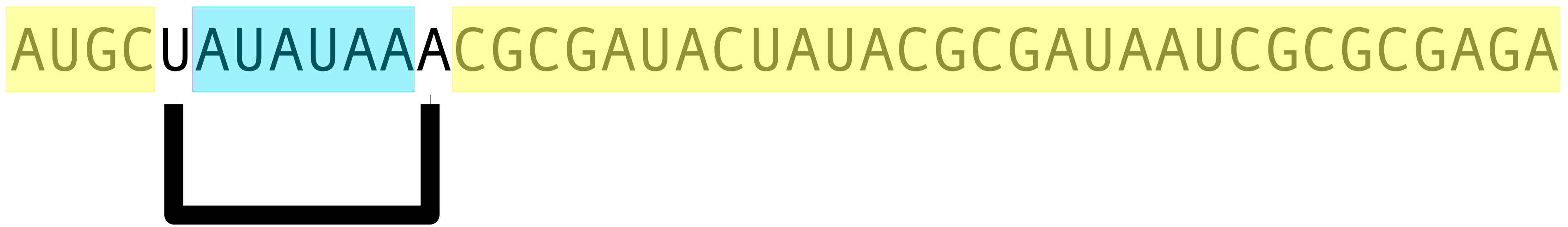
RNA folding


AUGCUAUAUAAACGCGAUACUAUACGCGAUAAUCGCGCGAGA

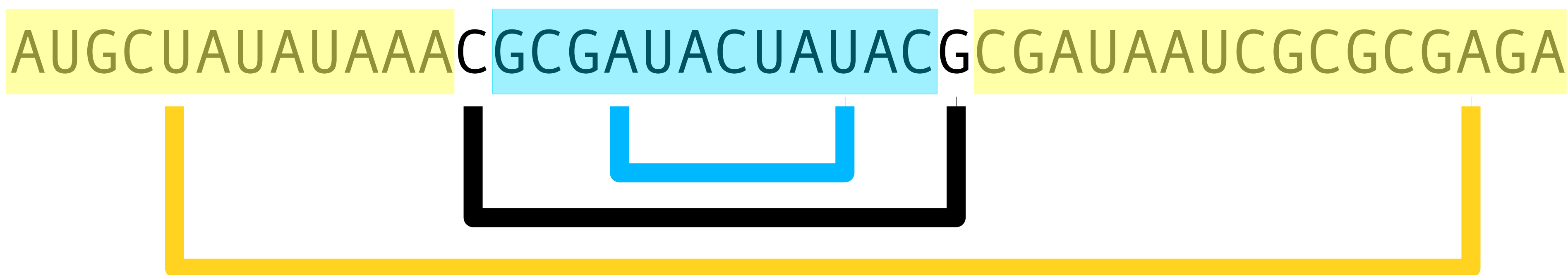

AUGCUAUAUAAACGCGAUACUAUACGCGAUAAUCGCGCGAGA

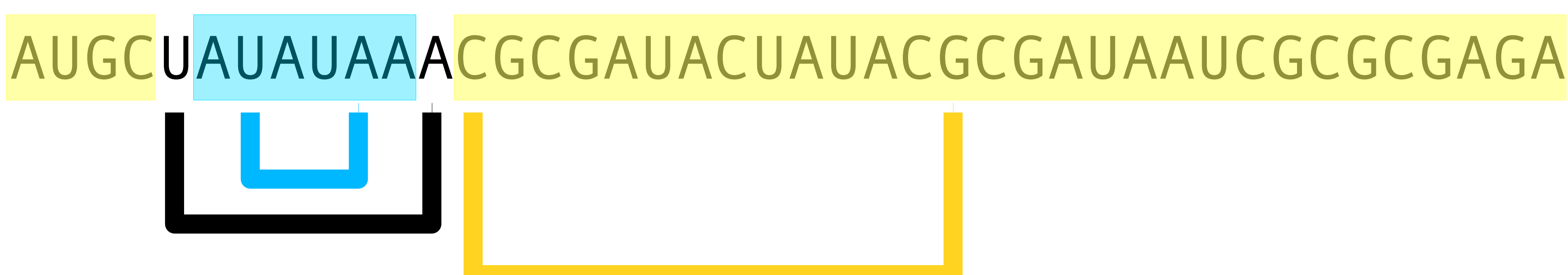
RNA folding


AUGCUAUAUAAACGCGAUACUAUACGCGAUAAUCGCGCGAGA


AUGCUAUAUAAACGCGAUACUAUACGCGAUAAUCGCGCGAGA

RNA folding


AUGCUAUAUAAACGCGAUACUAUACGCGAUAAUCGCGCGAGA


AUGCUAUAUAAACGCGAUACUAUACGCGAUAAUCGCGCGAGA

RNA folding

```
function max_folds(seq)


    if length(seq) <= 1
        return 0
    current_max = 0
    for i in 1..length(seq)
         for j    in i..length(seq)
            if complement(seq[i], seq[j])
                left = seq[1:i-1]
                middle = seq[i:j]
                right = seq[j+1:length(seq)]
                num_folds = 1 + max_folds(left + right)
                               + max_folds(middle)
                if num_folds > current_max
                    current_max = num_folds

    return current_max
```

RNA folding

```
function max_folds(seq, solutions)
    if seq in solutions
     return solutions[seq]
    if length(seq) <= 1
        return 0
    current_max = 0
    for i in 1..length(seq)
         for j    in i..length(seq)
            if complement(seq[i], seq[j])
                left = seq[1:i-1]
                middle = seq[i:j]
                right = seq[j+1:length(seq)]
                num_folds = 1 + max_folds(left + right, memo)
                               + max_folds(middle, memo)
                if num_folds > current_max
                    current_max = num_folds
    solutions[seq] = current_max
    return current_max
```

A more efficient all minimum-weight paths with cycles algorithm?

- Floyd-Warshall
  - Calculates the minimum-weight path between all pairs of vertices simultaneously

A more efficient all minimum-weight paths with cycles algorithm?

- Floyd-Warshall
  - Calculates the minimum-weight path between all pairs of vertices simultaneously
  - Basic idea
    - Given the shortest path between vertices u and v that only uses vertices 1 to k

A more efficient all minimum-weight paths with cycles algorithm?

- Floyd-Warshall
  - Calculates the minimum-weight path between all pairs of vertices simultaneously
  - Basic idea
    - Given the shortest path between vertices u and v that only uses vertices 1 to k
      - What is the shortest path between u and v that only uses vertices 1 to k + 1?

A more efficient all minimum-weight paths with cycles algorithm?

- Floyd-Warshall
  - Calculates the minimum-weight path between all pairs of vertices simultaneously
  - Basic idea
    - Given the shortest path between vertices u and v that only uses vertices 1 to k
      - What is the shortest path between u and v that only uses vertices 1 to k + 1?
  - How can we reconstruct a path?