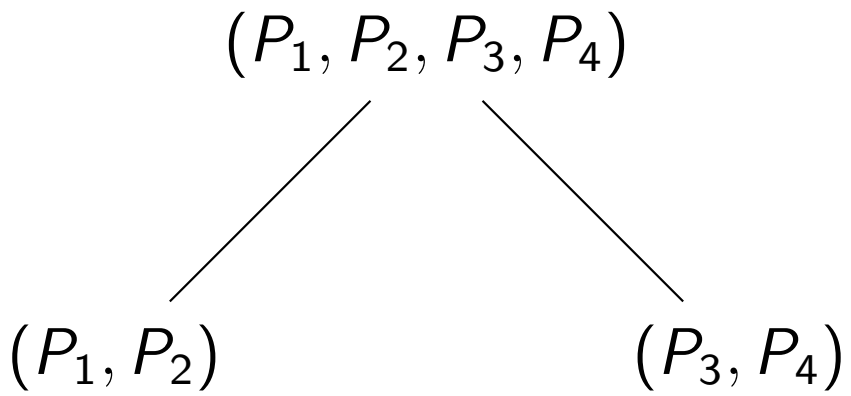
Homework 6

Yu Hsiao Yu-Hsuan Wu

Question 1

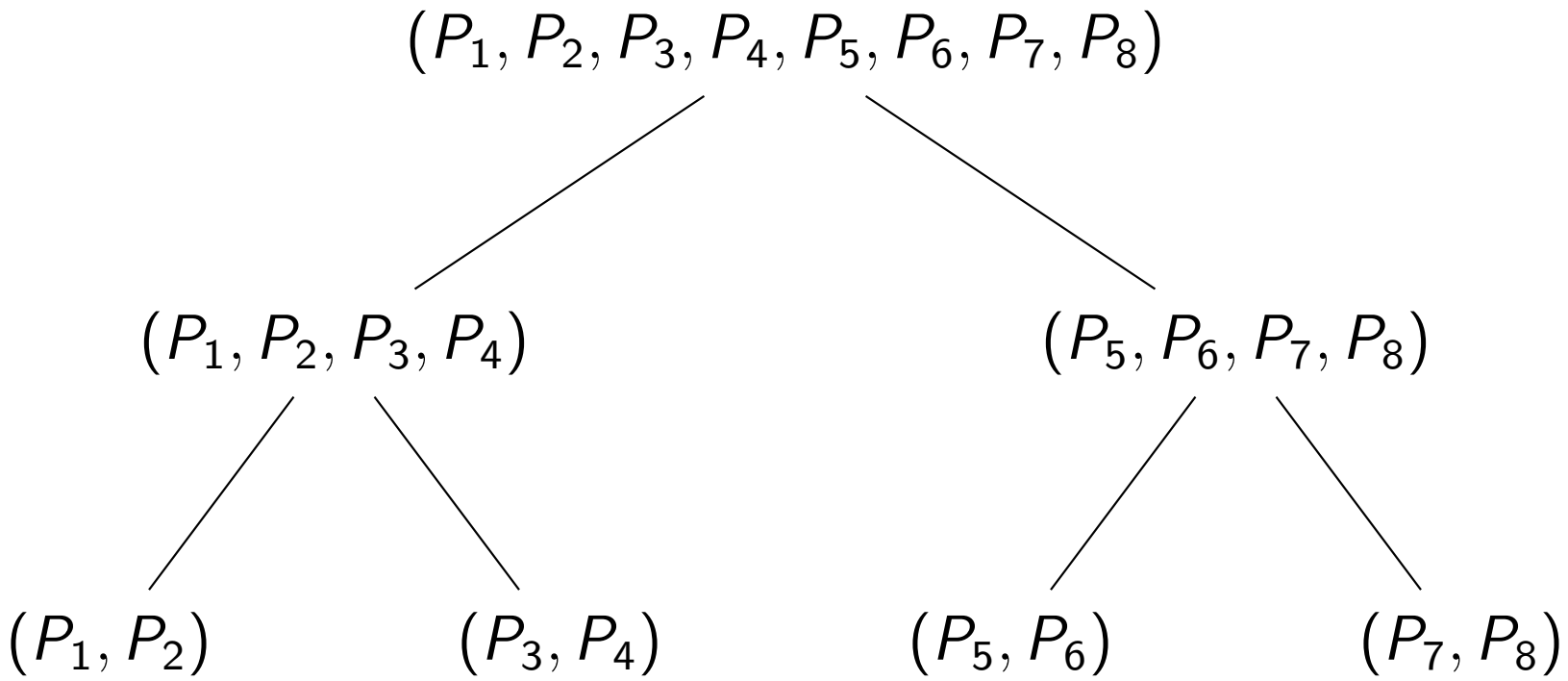
1. (6.62) You are asked to design a schedule for a round-robin tennis tournament. There are $n = 2^k$ $(k \geq 1)$ players. Each player must play every other player, and each player must paly one match per round for $n-1$ rounds. Denote the players by $P_1, P_2, \ldots, P_n$. Output the schedule for each player. (Hint: use divide and conquer in the following way. First, divide the players into two equal groups and let them play within the groups for the first $\frac{n}{2} - 1$ rounds. Then, design the games between the groups for the other $\frac{n}{2}$ rounds.)

Question 1



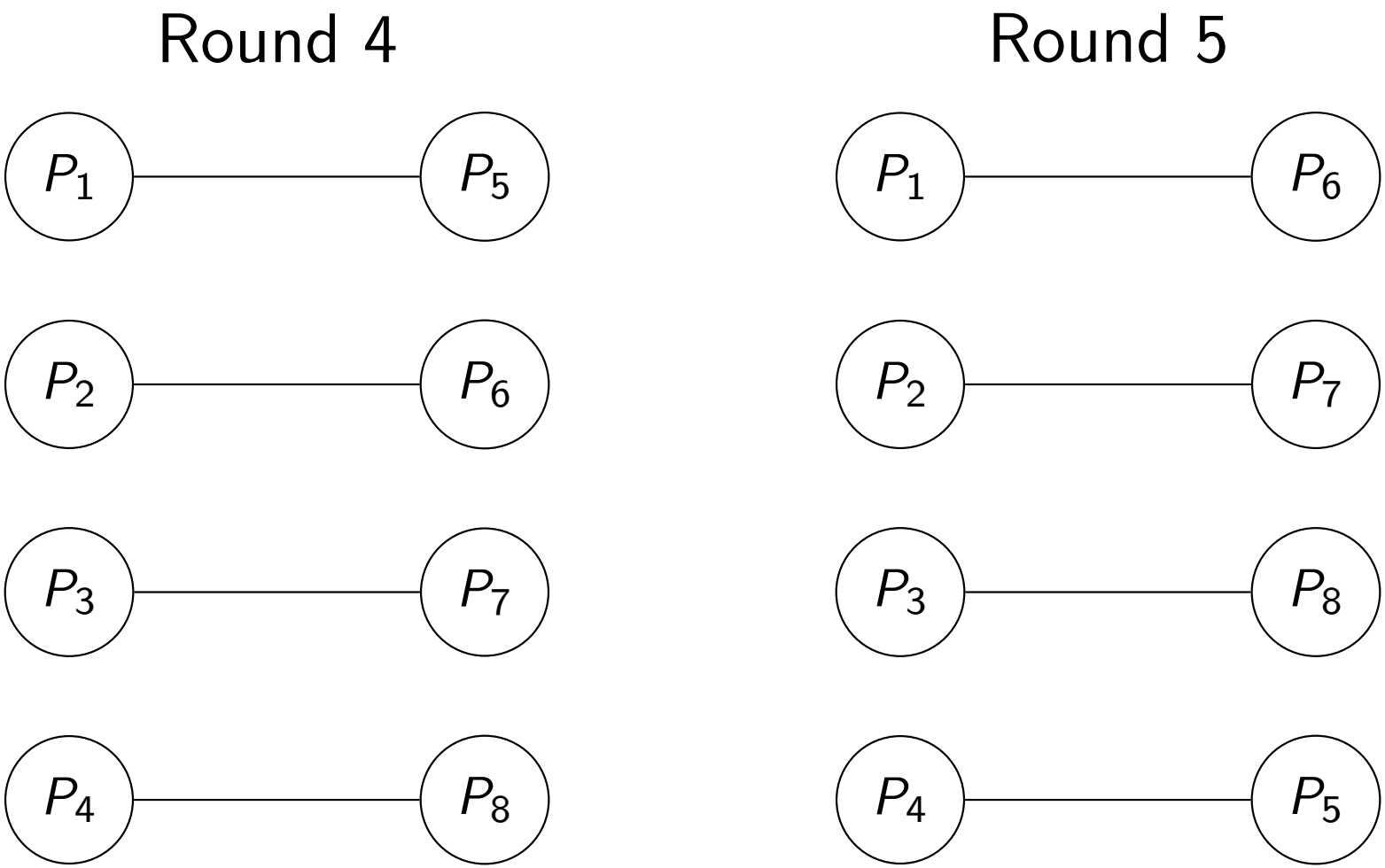
- Matches within groups
  - Round-1: P_1 vs. P_2
  - Round-1: P_3 vs. P_4
- Matches between groups
  - Round-2: P_1 vs. P_3
  - Round-2: P_2 vs. P_4
  - Round-3: P_1 vs. P_4
  - Round-3: P_2 vs. P_3

Question 1



You may observe that each subgroup with n players holds round $\frac{n}{2}$ to round $n-1$. Since $\frac{n}{2}-1$ rounds has been played in within groups matches.

Question 1



Question 1

function ROUND-ROBIN SCHEDULE(L, R, *playerCnt*)
  if $R - L = 1$ **then**
    $print$(Round 1 : L vs.R)
  else
    $M := (L + R)/2$
    Round-Robin Schedule($L, M,$ *playerCnt*$/2$)
    Round-Robin Schedule($M + 1, R,$ *playerCnt*$/2$)
  end if
  for $r :=$ *playerCnt*$/2$ **to** *playerCnt* $- 1$ **do**
    shift $:= r\%($*playerCnt*$/2)$
    for $i := 0$ **to** $M - L$ **do**
      $P_1 := L + i$
      $P_2 := M + 1 + [(i +$ *shift*$)\%($*playerCnt*$/2)]$
      $print$(Round r : P_1 vs.P_2)
    end for
  end for
end function

Question 1

For simplicity:

function ROUND-ROBIN SCHEDULE(L, R, $playerCnt$)
  if $R - L = 1$ **then**
    $print$(Round 1 : L vs.R)
  else
    $M := (L + R)/2$
    Round-Robin Schedule($L, M, playerCnt/2$)
    Round-Robin Schedule($M + 1, R, playerCnt/2$)
  end if
  for $r := playerCnt/2$ **to** $playerCnt - 1$ **do**
    for $i := 0$ **to** $M - L$ **do**
      $P_1 := L + i$
      $P_2 := M + 1 + [(i + r)\%(playerCnt/2)]$
      $print$(Round r : P_1 vs.P_2)
    end for
  end for
end function

Question 2

2. (6.32) Prove that the sum of the heights of all nodes in a complete binary tree with n nodes is at most $n - 1$. (A complete binary tree with n nodes is one that can be compactly represented by an array A of size n, where the root is stored in $A[1]$ and the left and the right children of $A[i]$, $1 \leq i \leq \lfloor \frac{n}{2} \rfloor$, are stored respectively in $A[2i]$ and $A[2i + 1]$. Notice that, in Manber's book a complete binary tree is referred to as a balanced binary tree and a full binary tree as a complete binary tree. Manber's definitions seem to be less frequently used. Do not let the different names confuse you. "Balanced binary tree" in the original problem description is the same as "complete binary tree")

Question 2

Prove *by induction* that the sum of the heights of all nodes in a complete binary tree with n nodes is at most $n - 1$.

You may assume it is known that the sum of the heights of all nodes in a *full binary tree* of height h is $2^{h+1} - h - 2$.

Question 2

There are several patterns of complete binary tree:

1. a single-node tree of height 0.
2. a two-node tree of height 1 where the root has a left child.
3. composed from a full binary tree of height h and a complete (possibly full) binary tree of height h as the left and the right subtrees of the root, resulting in a tree of height $h + 1$.
4. composed from a complete (possibly full) binary tree of height h and a full binary tree of height $h - 1$ as the left and the right subtrees of the root, resulting also in a tree of height $h + 1$.

Question 2

Let $G(n)$ denote the sum of the heights of all nodes in a complete binary tree with n nodes.
For a full binary tree (a special case of complete binary tree) of height h, we already know that:

$$\begin{aligned} n &= 2^{h+1} - 1 \\ G(n) &= 2^{h+1} - (h+2) \\ &= n - (h+1) \leq n - 1 \end{aligned}$$

With this as a basis, we prove that $G(n) \leq n - 1$ for the general case of arbitrary complete binary trees by induction on the number n of nodes.

Question 2

Base case ($n = 1$ or $n = 2$):

When $n = 1$, $G(n) = 0 \leq 1 - 1 = n - 1$.
When $n = 2$, $G(n) = 1 \leq 2 - 1 = n - 1$.

Inductive step ($n > 2$):

If n happens to be equal to $2^{h+1} - 1$ for some $h \geq 1$, the tree is a full binary tree, then we are done.

Otherwise, suppose $2^{h+1} - 1 < n < 2^{h+2} - 1$ ($h \geq 1$), the tree is a "proper" complete binary tree with height $h + 1 \geq 2$, There are two cases to consider:

Question 2

Case 1:

The left subtree is full of height h with n_l nodes, and the right one is complete also of height h with n_r nodes (such that $n_l + n_r + 1 = n$). From the special case of full binary tree and the induction hypothesis:

$$\begin{aligned} G(n_l) &= 2^{h+1} - (h+2) = n_l - (h+1) \\ G(n_r) &\leq n_r - 1, \end{aligned}$$

we have

$$\begin{aligned} G(n) &= G(n_l) + G(n_r) + (h+1) \\ &\leq (n_l - (h+1)) + (n_r - 1) + (h+1) \\ &= (n_l + n_r + 1) - 2 \\ &\leq n - 1. \end{aligned}$$

Question 2

Case 2:

The left subtree is complete of height h with n_l nodes and the right one is full of height $h-1$ with n_r nodes.
From the induction hypothesis and the special case of full binary trees:

$$\begin{aligned} G(n_l) &\leq n_l - 1 \\ G(n_r) &= 2^h - (h+1) = n_r - h, \end{aligned}$$

we have

$$\begin{aligned} G(n) &= G(n_l) + G(n_r) + (h+1) \\ &\leq (n_l - 1) + (n_r - h) + (h+1) \\ &= (n_l + n_r + 1) - 1 \\ &\leq n - 1. \end{aligned}$$

Question 3

3. (6.40 adapted) Design an algorithm that, given a set of integers $S = \{x_1, x_2, \ldots, x_n\}$, finds a nonempty subset $R \subseteq S$, such that

$$\sum_{x_i \in R} x_i \equiv 0 \pmod{n}.$$

Please present your algorithm in adequate pseudocode and make assumptions wherever necessary. Give also an analysis of its time complexity. The more efficient your algorithm is, the more points you will be credited for this problem.

Before presenting your algorithm, please argue why such a nonempty subset must exist.

Question 3

First, we must prove that "There must exist a nonempty subset $R \subseteq S$ such that $\sum_{x_i \in R} x_i \equiv 0 \pmod{n}$." is true.

Let S_i be the subset sum of $x_1 + \cdots + x_i$ for $1 \leq i \leq n$. We have

$$
\begin{aligned}
S_1 &= x_1 \\
S_2 &= x_1 + x_2 \\
&\vdots \\
S_k &= x_1 + x_2 + \cdots + x_k \\
&\vdots \\
S_n &= x_1 + x_2 + \cdots + x_k + \cdots + x_n.
\end{aligned}
$$

Question 3

Case 1: There exists some k such that $S_k \equiv 0 \pmod{n}$.

If there exists some k with $1 \leq k \leq n$ such that $S_k \equiv 0 \pmod{n}$, then we are done.

Case 2: There does not exist any k such that $S_k \equiv 0 \pmod{n}$.

In this case, we know that there must exist two subset sums S_i, S_j with $1 \leq i < j \leq n$ having the same remainder when divided by n since there should be at most n remainders for n sums, but at most $n-1$ distinct remainders appear (excluding the zero remainder).

Recall that $S_i = x_1 + \cdots + x_i$ and $S_j = x_1 + \cdots + x_j$. By subtracting S_j with S_i, we have $S_{j-i} = x_{i+1} + \cdots + x_j$, where $S_{j-i} \equiv 0 \pmod{n}$.

In conclusion, a nonempty subset must exist.

Question 3

```
function FindSubset(S, n)
    Sum := 0
    for i := 1 to n − 1 do
        R[i] := 0
    end for
    for k := 1 to n do
        Sum := Sum + x_k
        Remainder := Sum%n
        if Remainder = 0 then
            Return {x_1, ··· , x_k}
        end if
        if R[Remainder] ≠ 0 then
            i := R[Remainder] + 1
            Return {x_i, ··· , x_k}
        else
            R[Remainder] := k
        end if
    end for
end function
```

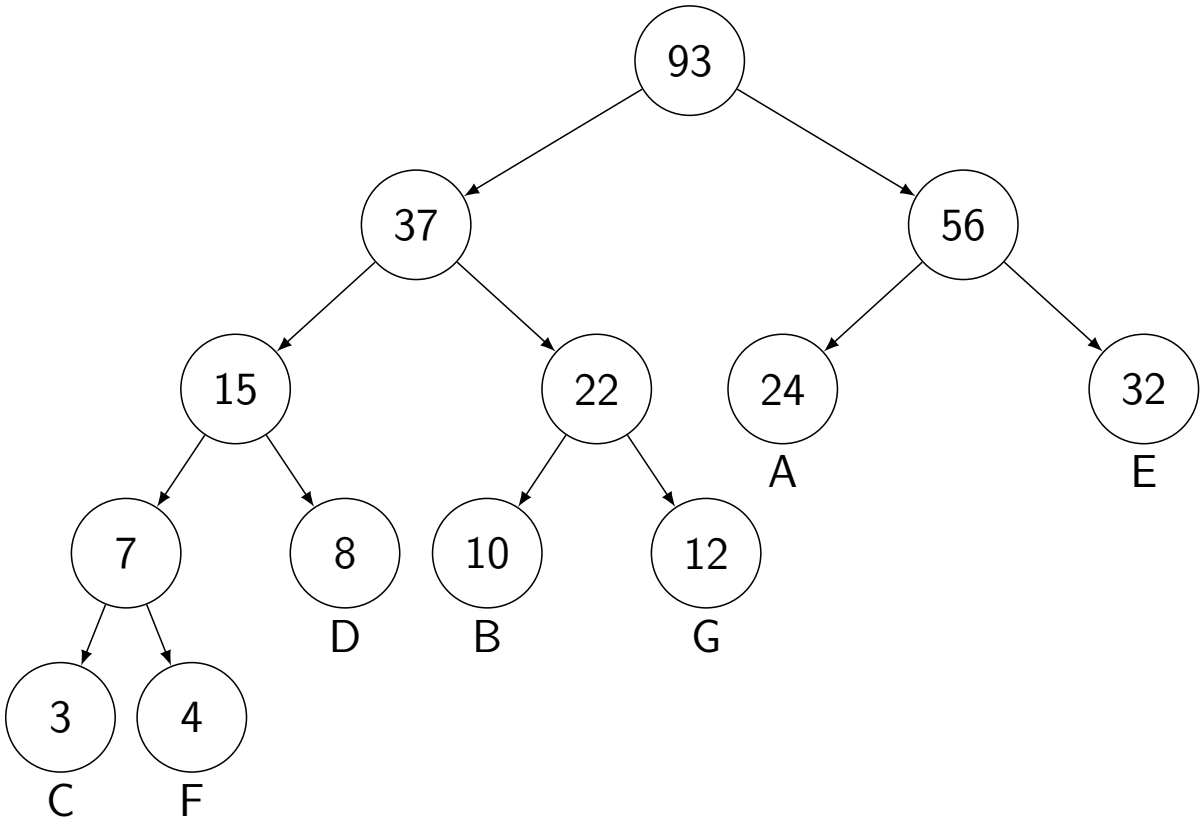
Notice that $R[i] = k$ means the subset sum S_k has remainder i when divided by n.

The time complexity is $O(n)$.

Question 4

4. Construct a Huffman code tree for a text composed from seven characters A, B, C, D, E, F, and G with frequencies 24, 10, 3, 8, 32, 4, and 12 respectively. And then, list the codes for all the characters according to the code tree.

Question 4



Character	Frequency	Code
A	24	10
B	10	010
C	3	0000
D	8	001
E	32	11
F	4	0001
G	12	011

Question 5

5. Consider the *next* table as in the KMP algorithm (the version presented in class) for string $B[1..9] = abaababaa$.

1	2	3	4	5	6	7	8	9
a	b	a	a	b	a	b	a	a
-1	0	0	1	1	2	3	2	3

Suppose that, during an execution of the KMP algorithm, $B[6]$ (which is an a) is being compared with a letter in A, say $A[i]$, which is not an a and so the matching fails. The algorithm will next try to compare $B[next[6]+1]$, i.e., $B[3]$ which is also an a, with $A[i]$. The matching is bound to fail for the same reason. This comparison could have been avoided, as we know from B itself that $B[6]$ equals $B[3]$ and, if $B[6]$ does not match $A[i]$, then $B[3]$ certainly will not, either. $B[5]$, $B[8]$, and $B[9]$ all have the same problem, but $B[7]$ does not.

Please adapt the computation of the *next* table so that such wasted comparisons can be avoided. Also, please give, for a new string $B[1..9] = babbabbba$, the values of the original *next* table and those of the new *next* table according to the adaptation.

Question 5

The original next table of string B:

1	2	3	4	5	6	7	8	9
a	b	a	a	b	a	b	a	a
-1	0	0	1	1	2	3	2	3

$next[9]$ represents: if comparison between $A[i]$ and $B[9]$ fails, we are going to compare $A[i]$ with $B[next[9]+1](B[4])$ next.

However, $B[9]$ and $B[4]$ has the same string so the comparison is doomed to fail. To avoid wasted comparisons, we can directly compare $A[i]$ with $B[next[4]+1](B[2])$ if we fail matching $A[i]$ with $B[9]$.

Question 5

```
function COMPUTE_NEXT(B, m)
    next[1] := −1; next[2] := 0;
    for i := 3 to m do
        j := next[i − 1] + 1;
        while B[i − 1] ≠ B[j] and j > 0 do
            j := next[j] + 1;
        end while
        next[i] := j;
    end for
end function
```

Question 5

function COMPUTE_NEXT_ADAPTED(B, m)
    $next[1] := -1$; $next[2] := 0$;
    for $i := 3$ **to** m **do**
        $j := next[i-1] + 1$;
        while $B[i-1] \neq B[j]$ and $j > 0$ **do**
            $j := next[j] + 1$;
        end while
        $next[i] := j$;
    end for
    // Add the following lines for optimization but less efficient.
    for $i := m$ **down to** 2 **do**
        $j := next[i] + 1$;
        while $B[i] = B[j]$ and $j > 0$ **do**
            $j := next[j] + 1$;
        end while
        $next[i] := j - 1$;
    end for
end function

Question 5

```
function COMPUTE_NEXT_ADAPTED(B, m)
    next[1] := −1; next[2] := 0;
    for i := 3 to m do
        j := next[i − 1] + 1;
        while B[i − 1] ≠ B[j] and j > 0 do
            j := next[j] + 1;
        end while
        next[i] := j;
    end for
    // Add the following lines for optimization.
    for i := 2 to m do
        j := next[i] + 1;
        if B[i] = B[j] and j > 0 then
            next[i] := next[j];
        end if
    end for
end function
```

Question 5

For string $B[1..9] = abbaabbaa$, the original *next* table:

1	2	3	4	5	6	7	8	9
a	b	a	a	b	a	b	a	a
-1	0	0	1	1	2	3	2	3

New *next* table:

1	2	3	4	5	6	7	8	9
a	b	a	a	b	a	b	a	a
-1	0	-1	1	0	-1	3	-1	1