Page 292, exercise 7.
I view the matrix as distances between nodes 0, 1, 2, 3, 4. A matrix $R^k$ allows intermediate nodes $0, ..., k - 1$.

\[
\begin{bmatrix}
0 & 2 & \cdot & 1 & 8 \\
6 & 0 & 3 & 2 & \cdot \\
\cdot & 0 & 4 & \cdot \\
\cdot & 2 & 0 & 3 \\
\cdot & \cdot & \cdot & \cdot & 0
\end{bmatrix}
\rightarrow
\begin{bmatrix}
0 & 2 & \cdot & 1 & 8 \\
6 & 0 & 3 & 2 & 14 \\
\cdot & \cdot & 0 & 4 & \cdot \\
\cdot & \cdot & 2 & 0 & 3 \\
\cdot & \cdot & \cdot & \cdot & 0
\end{bmatrix}
\rightarrow
\begin{bmatrix}
0 & 2 & 5 & 1 & 8 \\
6 & 0 & 3 & 2 & 14 \\
\cdot & \cdot & 0 & 4 & \cdot \\
\cdot & \cdot & 2 & 0 & 3 \\
\cdot & \cdot & \cdot & \cdot & 0
\end{bmatrix}
\rightarrow
\begin{bmatrix}
0 & 2 & 3 & 1 & 4 \\
6 & 0 & 3 & 2 & 5 \\
\cdot & \cdot & 0 & 4 & 7 \\
\cdot & \cdot & 2 & 0 & 3 \\
\cdot & \cdot & \cdot & \cdot & 0
\end{bmatrix}
\rightarrow
\begin{bmatrix}
0 & 2 & 3 & 1 & 4 \\
6 & 0 & 3 & 2 & 5 \\
10 & 12 & 0 & 4 & 7 \\
6 & 8 & 2 & 0 & 3 \\
3 & 5 & 6 & 4 & 0
\end{bmatrix}
\]

Page 292, exercise 9.
Floyd’s algorithm gives incorrect results if there exists a negative cost cycle. In that case, the minimum cost path does not exist, but Floyd returns an answer. E.g. we may have two nodes, $d(0, 1) = 1$, $d(1, 0) = -2$.

Page 298, exercise 5.
False. For example, when the nodes have very similar search probability then the optimum shape is not changed. We may have nodes 0, 1, 2 with probabilities 0.34, 0.33, 0.33. With 0 at the root the average search cost is 1.99 and with 1 at the root the average is 1.67.

Page 299, exercise 10.

a. Let $A_1$ be 2000×1 matrix, $A_2$ be 1×2000 matrix, and $A_3$ be 2000×1 matrix.
   Evaluating $(A_1 \times A_2) \times A_3$ costs $2000 \times 1 \times 2000 + 2000 \times 2000 \times 1 = 8,000,000$.
   Evaluating $A_1 \times (A_2 \times A_3)$ costs $1 \times 2000 \times 1 + 2000 \times 1 \times 1 = 4,000$.

b. A “way” to compute the product $n$ matrices is a binary tree with $n$ leaves. As discussed on page 294, this is the $n$-th Catalan number,

\[
c(n) = \binom{2n}{n} \frac{1}{n+1}
\]

with order of growth $\Theta(4^n n^{-1.5})$.

c. Design a dynamic programming solution.
   The input will be the array of matrix dimensions, $A_i$ is a $D[i-1] \times D[i]$ matrix.
   We define a recursive subproblem: to find the minimum cost $C[i, j]$ of multiplying matrices $A_i \times \cdots \times A_j$, where $i \leq j$.
   The basic case is $C[i, i] = 0$, no cost to multiply when there is only one matrix.
   The recursive relationship holds when $i < j$:
\[ C[i, j] = \min_{k=i}^{j-1} C[i, k] + C[k + 1, j] + D[i - 1] \times D[k] \times D[j] \]

Now we can fill a matrix iteratively, in order of increasing \( j - i \):

```java
for (i = 1; i <= n; i++)
    C[i][i] = 0;
for (s = 1; s <= n; s++)
    for (i = 1; i+s <= n; i++) {
        j = i+s;
        C[i][j] = infty;
        for (k = i; k < j; k++) {
            new = C[i][k]+C[k+1][j]+D[i-1]*D[k]*D[j];
            if (new < C[i][j])
                B[i][j] = k,
                C[i][j] = new;
        }
    }
```

We can use array \( B \) to establish the actual optimal order of multiplications.

Page 314, exercise 7.

a.

Black numbers are priorities, 99 is infinity, red numbers are deleted from the queue.
b.

Black numbers are priorities, red numbers are deleted from the queue.

In the last three steps the blue edges are selected, no better candidates appear.
EXTRA CREDIT

Page 292, exercise 5.

Warshall’s algorithm can be written as follows. We initialize array $R$ with the adjacency matrix of the graph and then we run this iteration:

```c
for (k = 0; k < n; k++)
    for (i = 0; i < n; i++)
        // innermost loop
            for (j = 0; j < n; j++)
                $R[i][j] ||= R[i][k] && R[k][j]$;
```

In the innermost loop the value of $R[i][k]$ is fixed, so we can skip doing it if it is 0 (false), and “ignore it” if it is 1 (true):

```c
for (k = 0; k < n; k++)
    for (i = 0; i < n; i++) {
        if (!$R[i][k])
            continue;
        // innermost loop
        for (j = 0; j < n; j++)
            $R[i][j] ||= R[k][j]$;
    }
```

Now the innermost loop replaces $i$-th row with a position-wise “or” with the $k$-th row. If we represent rows as bit strings, then the task of the innermost loop can be achieved using bitwise and.