# National University of Computer and Emerging Sciences

**FAST School of Computing** **Summer-2023** **Islamabad Campus**

# CS-2009: Design and Analysis of Algorithms

Serial No:

## Final Examination
## Total Time: 3 Hours
## Total Marks: 150

Friday, 11th August, 2023

## Course Instructors

Nirmal Tariq

_____

Signature of Invigilator

| _____ | _____ | _____ | _____ |
|---|---|---|---|
| Student Name | Roll No. | Course Section | Student Signature |

**DO NOT OPEN THE QUESTION BOOK OR START UNTIL INSTRUCTED.**

**Instructions:**

1. Attempt on question paper. Attempt all of them. Read the question carefully, understand the question, and then attempt it.
2. No additional sheet will be provided for rough work. Use the back of the last page for rough work.
3. If you need more space write on the back side of the paper and clearly mark question and part number etc.
4. After asked to commence the exam, please verify that you have **seventeen (17)** different printed pages including this title page. There are a total of **9** questions.
5. Calculator sharing is strictly prohibited.
6. Use permanent ink pens only. Any part done using soft pencil will not be marked and cannot be claimed for rechecking.

| | Q-1 | Q-2 | Q-3 | Q-4 | Q-5 | Q-6 | Q-7 | Q-8 | Q-9 | Total |
|---|---|---|---|---|---|---|---|---|---|---|
| **Marks Obtained** | | | | | | | | | | |
| **Total Marks** | **20** | **15** | **15** | **30** | **15** | **15** | **10** | **15** | **15** | **150** |

## Question 1 [5+5+10= 20 Marks]

a) Write an algorithm in pseudocode form to find the number of distinct elements in an array A[1...n] in O(n) time; where "n" is the array size. For example, if the array is {3,1,3,8,2,1,8,2}, the number of distinct elements is 4 (to be returned from the procedure) as the distinct elements are {1,2,3,8}. All the elements of the array are in the range [1,100]. Also, n >> 100 (n is significantly greater than 100). You are not allowed to use more than constant extra space i.e. use of a few variables is allowed. There will be no credit for a solution that will take more than linear time or more than constant extra space.

Note: Use of any implicit procedure call (for procedure not made by you) is not allowed.

```
Procedure DistinctElementsInArray
BEGIN
Inputs: List A[1…n]  and n
Output: count    {Number of  Distinct Elements}
        count = 0
        counts_Array[101]    {for storing the frequencies of distinct elements}
        FOR (i = 1 to 100)
                counts_Array[i] = 0
        END FOR

        FOR (i = 1 to n)
                counts_Array [ array[i] ] = counts_Array [ array[i] ] + 1
        END FOR
        FOR (i = 1 to 100)
                IF ( counts_Array[i] >  0) THEN
                        count = count + 1
                END IF
        END FOR
        RETURN count
END PROCEDURE
```

b) Which famous problem is solved by the following C++ code? Write the time complexity of the code in terms of Big-O notation.

```
int abc(int n)
{
    int prev1=0, prev2=1;
        for(int i=0; i<n; i++)
        {
                int savePrev1 = prev1;
                prev1 = prev2;
                prev2 = savePrev1 + prev2;
        }
  return prev1;
}
```

Finding n-th term of Fibonacci series. O(n)

c) The following information is based on a piece of text using a set of five different symbols. The frequencies of the symbols in the text are given below:

| Symbol | Frequency |
|---|---|
| B | 24 |
| D | 12 |
| A | 30 |
| E | 10 |
| C | 8 |

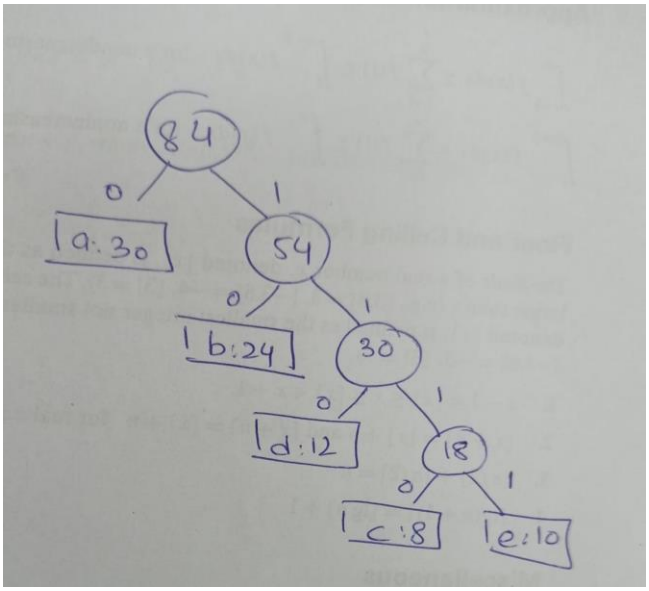
i. What is the minimum number of bits required to store the text using fixed-length coding scheme? Justify your answer **(2)**.

With a fixed-length encoding, you could represent each character in that string using 3 bits. There are 24+12+30+10+8= 84 total characters, so when using this method, the compressed string would be 84*3=252 bits long.

ii. What is the minimum number of bits required to store the text using a variable-length coding scheme? You are required to use the Huffman's algorithm learnt in the class. Justify your answer by showing all steps **(6)**.

186 bits are required to store text.

iii. Show the final Huffman's Tree **(2)**.




## Question 2 [10+2+3=15 Marks]
## Sorting Algorithms

a) Perform counting sort on the following array and show that it is a stable sort: Stable sort is described as a sorting algorithm that maintains the position of two equal elements relative to one another. In other words, whenever there are two items R and S with the same key and R appearing before S in the original list, R appears before S in sorted list then the sorting algorithm is labelled stable. For the following array provide a complete dry run of Counting sort and demonstrate if it is a stable sorting algorithm.

| | | | | | | |
|---|---|---|---|---|---|---|
| $1_A$ | 3 | $2_A$ | 8 | $5_A$ | $1_B$ | $5_B$ |

```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  // C[i] now contains the number of elements equal to i.
 7  for i = 1 to k
 8      C[i] = C[i] + C[i − 1]
 9  // C[i] now contains the number of elements less than or equal to i.
10  for j = A.length downto 1
11      B[C[A[j]]] = A[j]
12      C[A[j]] = C[A[j]] − 1
```

b) Counting sort is a linear sorting algorithm, But why don't we always use counting sort?

Because of the extra space required for B and C.

c) In which scenario Counting sort will be best suited? Is quick sort stable or not? Why/Why not?

Counting sort algorithm work best if k is not significantly larger than n. In this case the complexity becomes close to O(n) or linear. Also, larger the range of elements in the given array, larger is the space complexity. ____

QuickSort is an unstable algorithm because we do swapping of elements according to pivot's position (without considering their original positions)

**Question 3 [4+7+4=15 Marks]**
**Divide and Conquer**

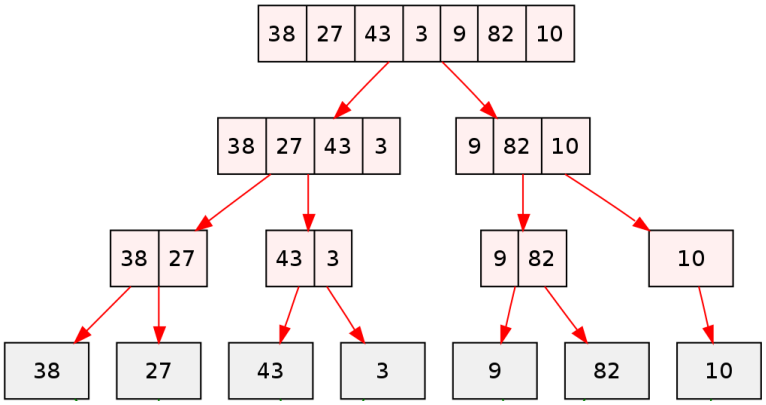
a) Explain under what circumstances dynamic programming technique is better than divide and conquer technique?

Divide-and-conquer algorithms partition the problem into disjoint subproblems, solve the subproblems recursively, and then combine their solutions to solve the original problem. In contrast, dynamic programming applies when the subproblems overlap—that is, when subproblems share subsubproblems. In this context, a divide-and-conquer algorithm does more work than necessary, repeatedly solving the common subsubproblems. A dynamic-programming algorithm solves each subsubproblem just once and then saves its answer in a table, thereby avoiding the work of recomputing the answer every time it solves each subsubproblem.

b) Show why divide and conquer technique is a suitable approach instead of a dynamic programming technique in order to sort numbers using merge sort? Provide your reasoning with the help of a complete example.

Merge Sort cannot use Dynamic Programming, because the subproblems are not overlapping in any way

Following merge sort tree shows each sub problem is distinct




c) What happens when dynamic programming technique is applied to this problem? Discuss in terms of time and space complexities.

DP uses the memorization technique; it always stores the previously calculated values. Due to this, the time complexity is decreased but the space complexity is increased.

**Question 4 [10+10+5+5=30 Marks]**
**String Matching Algorithms**

a) Suppose that all characters in the pattern P are different. Show how to accelerate NAIVE-STRING-MATCHER to run in time *O(n)* on an n-character text T.

We know that one occurrence of P in T cannot overlap with another, so we don't need to double-check the way the naive algorithm does. If we find an occurrence of $P_k$ in the text followed by a nonmatch, we can increment s by k instead of 1.

```
NAIVE-STRING-MATCHER(T, P)
1  n = T.length
2  m = P.length
3  for s = 0 to n − m
4      if P[1..m] == T[s + 1..s + m]
5          print "Pattern occurs with shift" s
```

b) Construct the string-matching automaton for the pattern P = aabab and illustrate its operation on the text string T = aaababaabaababaab.

| | Inputs | |
|---|---|---|
| States | a | b |
| 0 | 1 | 0 |
| 1 | 2 | 0 |
| 2 | 2 | 3 |
| 3 | 4 | 0 |
| 4 | 2 | 5 |
| 5 | 1 | 0 |

| i | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| T[i] | | a | a | a | b | a | b | a | a | b | a | a | b | a | b | a | a | b |
| State | 0 | 1 | 2 | 2 | 3 | 4 | 5 | 1 | 2 | 3 | 4 | 2 | 3 | 4 | 5 | 1 | 2 | 3 |

Valid shifts s=1 and s=9.

c) Compute the prefix function π for the pattern ababbabbabbababbabb.

| i | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P[i] | a | b | a | b | b | a | b | b | a | b | b | a | b | a | b | b | a | b | b |
| π[i] | 0 | 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

d) Give brief answers:

i. What is the worst case time complexity of KMP algorithm for pattern searching (m = length of text, n = length of pattern)? O(m)

ii. The naive pattern searching algorithm is an in place algorithm (True/False). True

iii. Rabin Karp algorithm and naive pattern searching algorithm have the same worst case time complexity (True/False). True

iv. How does KMP improve the brute-force-method?

By computing prefix function and avoid testing useless shifts that the naïve algorithm does

v. If pattern = aab, then

σ (ccacab) = _____0_____

σ (cabaab) =_____3_____

## Question 5 [6+4+5=15 Marks]
## Dynamic Programming (Matrix-Chain Multiplication)

a) Find an optimal parenthesization of a matrix-chain product whose sequence of dimensions is {5, 10, 2, 3, 4}. Dimensions of matrices are: A (5x10), B (10x2), C (2x3), D(3x4).

Show step by step computation to fill up table 'M' and 'S' given below.

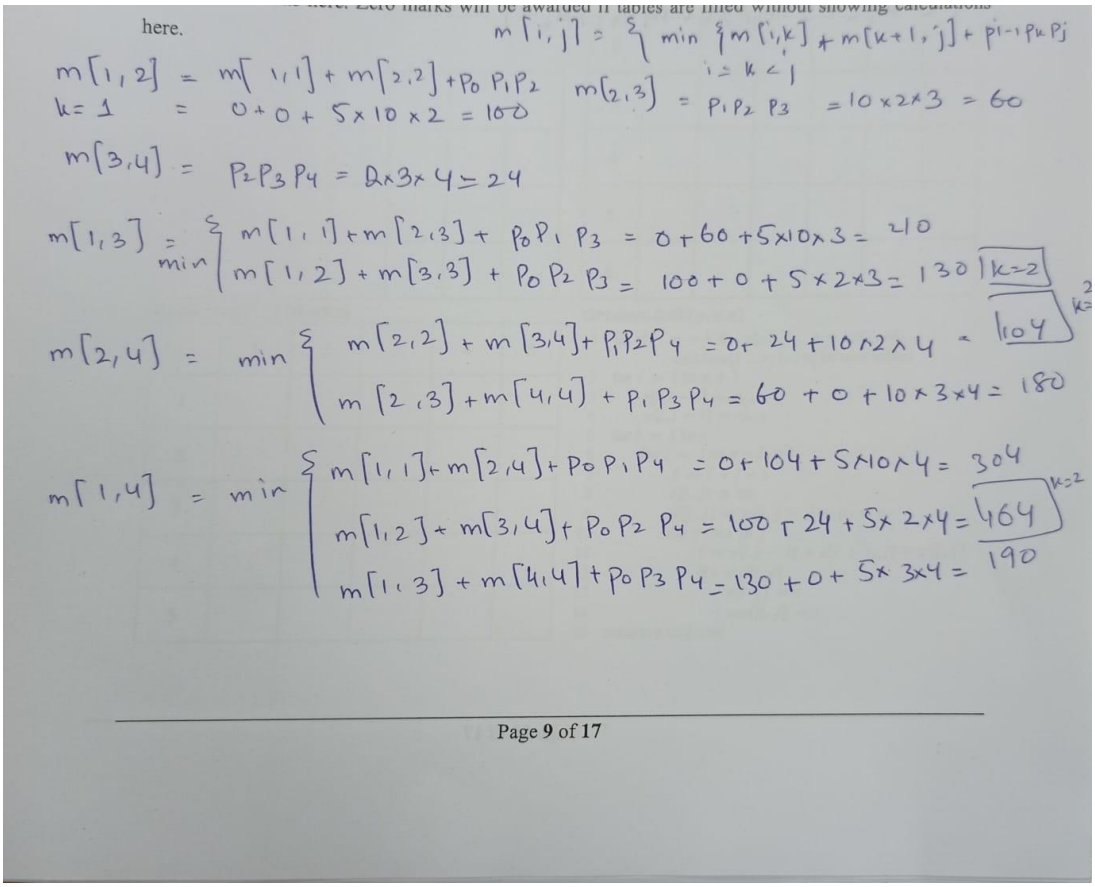
**Table 'M' (6 Marks)**

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | **0** | **100** | **130** | **164** |
| 2 | | **0** | **60** | **104** |
| 3 | | | **0** | **24** |
| 4 | | | | **0** |

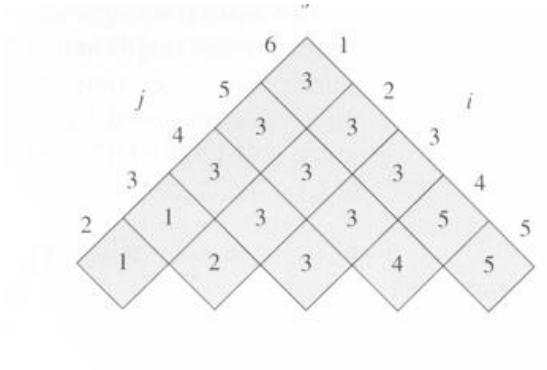
**Table 'S' (4 Marks)**

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | **-** | **1** | **2** | **2** |
| 2 | | **-** | **2** | **2** |
| 3 | | | **-** | **3** |
| 4 | | | | **-** |

Show calculations here. Zero marks will be awarded if tables are filled without showing calculations here.



b) Use the following 'S' table to compute the optimal solution and show optimal parenthesization.




```
PRINT-OPTIMAL-PARENS(s, i, j)
1  if i == j
2       print "A"_i
3  else print "("
4       PRINT-OPTIMAL-PARENS(s, i, s[i, j])
5       PRINT-OPTIMAL-PARENS(s, s[i, j] + 1, j)
6       print ")"
```

$((A_1(A_2A_3))((A_4A_5)A_6))$

## Question 6 [5+5+5=15 Marks]
## Dynamic Programming (OBST)

Determine the cost and structure of an optimal binary search tree for a set of n =5 keys with the following probabilities:

| $i$ | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| $p_i$ | | 0.15 | 0.10 | 0.05 | 0.10 | 0.20 |
| $q_i$ | 0.05 | 0.10 | 0.05 | 0.05 | 0.05 | 0.10 |

You are required to fill in the following tables of **'w', 'e'** and **'root'** and show the calculations in the given space below.

**Table 'e' (5 Marks)**

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| **1** | | | | | | |
| **2** | | | | | | |
| **3** | | | | | | |
| **4** | | | | | | |
| **5** | | | | | | |
| **6** | | | | | | |

**Table 'w' (5 Marks)**

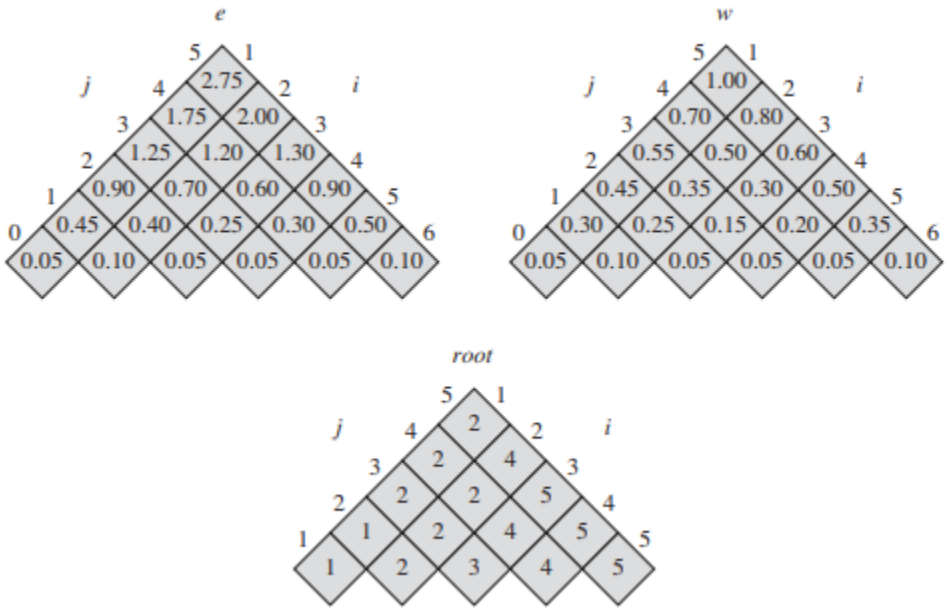
| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| **1** | | | | | | |
| **2** | | | | | | |
| **3** | | | | | | |
| **4** | | | | | | |
| **5** | | | | | | |
| **6** | | | | | | |

**Table 'root' (5 Marks)**

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| **1** | | | | | |
| **2** | | | | | |
| **3** | | | | | |
| **4** | | | | | |
| **5** | | | | | |

```
OPTIMAL-BST(p, q, n)
 1  let e[1..n+1, 0..n], w[1..n+1, 0..n],
        and root[1..n, 1..n] be new tables
 2  for i = 1 to n + 1
 3      e[i, i-1] = q_{i-1}
 4      w[i, i-1] = q_{i-1}
 5  for l = 1 to n
 6      for i = 1 to n - l + 1
 7          j = i + l - 1
 8          e[i, j] = ∞
 9          w[i, j] = w[i, j-1] + p_j + q_j
10          for r = i to j
11              t = e[i, r-1] + e[r+1, j] + w[i, j]
12              if t < e[i, j]
13                  e[i, j] = t
14                  root[i, j] = r
15  return e and root
```

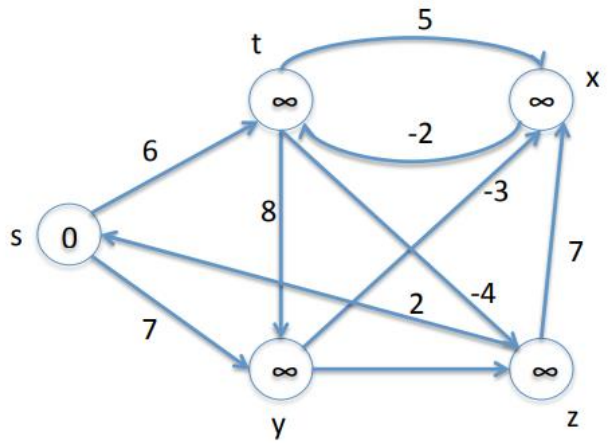
Show calculations here and note that no marks will be awarded without complete calculations.



## Question 7 [10 Marks]

## Single Source shortest path problem

Provide complete dry run of Bell-man Ford algorithm on the following graph and in the given order of edges.




$d[s] \leftarrow 0$
**for** each $v \in V - \{s\}$
  **do** $d[v] \leftarrow \infty$

**for** $i \leftarrow 1$ **to** $|V| - 1$
  **do for** each edge $(u, v) \in E$
    **do if** $d[v] > d[u] + w(u, v)$
      **then** $d[v] \leftarrow d[u] + w(u, v)$

**for** each edge $(u, v) \in E$
  **do if** $d[v] > d[u] + w(u, v)$
    **then** report that a negative-weight cycle exists

Edges considered in this order:

(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)

Fill the table below that shows vertex weights after each iteration.

| | **Iterations** | | | | |
|---|---|---|---|---|---|
| **Node** | **0** | **1** | **2** | **3** | **4** |
| **s** | **0** | **0** | **0** | **0** | **0** |
| **t** | **∞** | **6** | **6** | **6->2** | **2** |
| **y** | **∞** | **7** | **7** | **7** | **7** |
| **x** | **∞** | **∞** | **11->4** | **4** | **4** |
| **z** | **∞** | **∞** | **2** | **2** | **-2** |

Calculations for Bellman Ford algorithm:

**Question 8 [7.5+7.5= 15 Marks]**

**All pair shortest path problems**

You are given a directed graph $G = (V, E)$ with edge weights $w : E \rightarrow R$. In addition, each edge of the graph is either red or blue. The shortest red/blue path from vertex $i \in V$ to vertex $j \in V$ is defined as the shortest path from $i$ to $j$ among those paths that go through exactly one red edge (if there are no such paths, the length of the shortest red/blue path is $\infty$).

We can represent this graph with two $n$ x $n$ matrices of edge weights, $W_r$ and, $W_b$ , where $W_r$ contains the weights of all red edges, and $W_b$ contains the weights of all blue edges.

a) Given the Floyd-Warshall algorithm below, how would you modify the algorithm to obtain the lengths of the shortest paths that only go through blue edges?

```
FLOYD-WARSHALL(W):
1  n = W.rows
2  D^(0) = W
3  for k = 1 to n
4      let D^(k) = (d_ij^(k)) be a new n × n matrix
5      for i = 1 to n
6          for j = 1 to n
7              d_ij^(k) = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
8  return D^(n)
```

**Solution:** [6 points] In order to find shortest paths going through only blue edges, it suffices to ignore the red edges and run Floyd-Warshall on only the blue edges of the graph. We make the following changes:

- Replace each occurrence of $W$ with $W_b$ in lines 1 and 2.
- Replace the matrices $D^{(k)}$ with blue versions $D_b^{(k)}$ in lines 2, 4, and 8.
- Replace the matrix elements $d_{ij}^{(k)}$ with blue versions $d_{b,ij}^{(k)}$ in lines 4 and 7.

While the second and third changes above are unnecessary for this part, they lay the groundwork for future parts below.

b) How would you modify your algorithm from part (a) to keep track not only of shortest paths with only blue edges, but also those with exactly one red edge, and to output the lengths of the shortest red/blue paths for all pairs of vertices in this graph?

**Solution:** [6 points] Add a new set of matrices $D_r^{(k)}$ that give lengths of shortest paths with exactly one red edge and intermediate vertices up to $k$. The resulting pseudocode is as follows:

RED-BLUE-FLOYD-WARSHALL$(W_r, W_b)$:

```
1  n = W_r.rows
2  D_b^(0) = W_b
3  D_r^(0) = W_r
4  for k = 1 to n
5      let D_b^(k) = (d_{b,ij}^(k)), D_r^(k) = (d_{r,ij}^(k)) be new n × n matrices
6      for i = 1 to n
7          for j = 1 to n
8              d_{b,ij}^(k) = min(d_{b,ij}^(k-1), d_{b,ik}^(k-1) + d_{b,kj}^(k-1))
9              d_{r,ij}^(k) = min(d_{r,ij}^(k-1), d_{r,ik}^(k-1) + d_{b,kj}^(k-1), d_{b,ik}^(k-1) + d_{r,kj}^(k-1))
10 return D_r^(n)
```

**Question 9 [15 Marks]**

**Hashing**

Analyze the given hash function based on the four properties of a good hash function *(sequence of hash properties does not matter but property should be mentioned explicitly)*. Provide a justification against each of the four properties and also show a complete step by step dry run of the string "ab". Assume that there is no error in the code. The ASCII value of character a is 97.

```
int fun(char *p)
{
    char  m = 7;
    char h = 0x00;
    char g;
    while ( *p != NULL )
    {
        h = ( h << 1 ) + ( *p );
        if ( g = h & 0xF0 )
        {
            h = h ^ g;
        }
        p = p+1;
    }
    return h % m;
}
```

Following are the details of the function:

- Each number is of one byte and 0x.. represents hex number.
- 'm' is the size of hash table, and it is always a prime number.
- "<<" is a bitwise left shift operator.
- "^" is bitwise XOR operator.
- p is the received actual key.

| | |
|---|---|
| **Rule 1:**<br>The hash value is fully determined by the data being hashed.<br>Satisfied because hash depends on the input string.<br>**Rule 2:**<br>The hash function uses all the input data.<br>Satisfied because all input characters are used to make a hash value.<br>**Rule 3:**<br>The hash function uniformly distributes the data across the entire set of possible hash values.<br>Satisfied because left shift operation mutates the previous value of 'h' in each iteration and hence the distribution is approximately uniform due to almost equal probability of occurrence of each h value.<br>**Rule 4:**<br>The hash function generates very different hash values for similar strings.<br>Satisfied because h(ab) is not equal to h(ba) and slight change in strings mostly produces a different hash value. | **Rough Work:** |

**Dry Run:**

Test ab and ba

ab:

h= 00000000b + 01100001b = 01100001b
g= 01100001b & 11110000b = 01100000b
h= 01100001b XOR 01100000b = 00000001b

h= 00000010b + 01100010b = 01100100b
g= 01100100b & 11110000b = 01100000b
h= 01100100b XOR 01100000b = 00000100b
return 4d % 7d

h(ab)= 4

ba:

h= 00000000b + 01100010b = 01100010b
g= 01100010b & 11110000b = 01100000b
h= 01100010b XOR 01100000b = 00000010b

h= 00000100b + 01100001b = 01100101b
g= 01100101b & 11110000b = 01100000b
h= 01100101b XOR 01100000b = 00000101b

return 5d % 7d

h(ba)= 5