INDIAN STATISTICAL INSTITUTE

MTech(CS) I year 2025-2026

Subject: Computing Laboratory

Lab Test 4 (December 1, 2025)

Total: 60 marks Duration: 4 hours

GENERAL INSTRUCTIONS

1. All programs should take the required input via the standard input (terminal/keyboard), and print the desired output to the terminal.
2. Please make sure that your programs adhere strictly to the specified input and output format. Do not print extra strings asking the user for input, debugging messages, etc. These will cause the automatic checking system to fail.
3. Please make sure that the programs are free from memory errors and leaks, you will lose marks if they are not.

Q1. (20 marks)

(a) *Breadth first search (BFS).* Implement a function that traverses a simple, undirected, unweighted graph in breadth-first order, starting from a specified vertex. [5]

(b) *Number of distinct shortest paths.* Assume that all edges are of unit length. Modify the above function to compute the **number of distinct shortest paths** from a specified vertex u to another specified vertex v. [15]

For full credit, your algorithm should run in $O(|V| + |E|)$ time.

Input format:

A graph in the usual format supported by `read_graph()`, i.e., n, the number of vertices in the graph, followed by the adjacency list for each vertex ($v_0, v_1, \ldots, v_{n-1}$)

p ← non-negative integer

x_0
x_1 ← indices of p starting vertices for BFS
$\vdots$
x_{p-1}

q ← non-negative integer

u_0 v_0
u_1 v_1 ← source vertex, target vertex pairs
$\vdots$
u_{q-1} v_{q-1}

Output format: $p + q$ lines

List of vertices reachable from x_0, in breadth-first order (one line)

List of vertices reachable from x_1, in breadth-first order (one line)

$\vdots$

List of vertices reachable from x_{p-1}, in breadth-first order (one line)

u_0 v_0 number of distinct shortest paths from u_0 to v_0

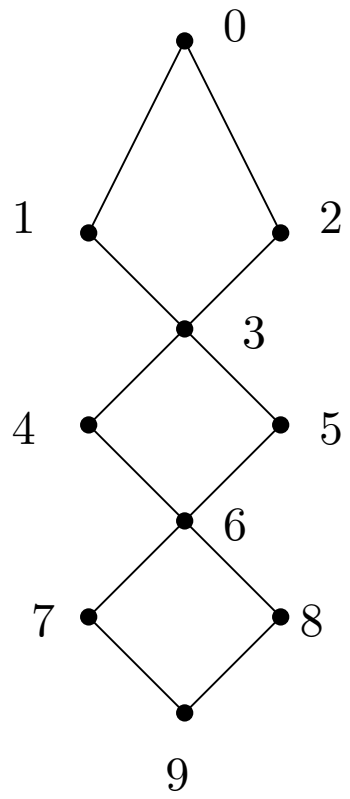
u_1 v_1 number of distinct shortest paths from u_1 to v_1

$\vdots$

NOTE: If v and v' are visited from u, then v should be listed before v' iff $v < v'$. Please see the provided test cases for more details.

Example: This is a simplified version of `input1-1.txt`.

```
10          # number of vertices
2 1 2       # 0
2 0 3       # 1
2 0 3       # 2
4 1 2 4 5   # 3
2 3 6       # 4
2 3 6       # 5
4 4 5 7 8   # 6
2 6 9       # 7
2 6 9       # 8
2 7 8       # 9
1           # p
1           # x_0
1           # q
0 9         # u_o  v_0
```



Q2. (20 marks) If you are unable to solve this problem, you may do the additional problem given below for partial credit (5 marks).

A *vertex cover* of a graph $G = (V, E)$ is a subset of vertices $S \subseteq V$ that includes at least one endpoint of every edge in E. Write a program that computes the size of the smallest vertex cover (SVC) of a given undirected **tree** T.

See the sketch of a dynamic programing algorithm for the *Maximum Independent Set in Trees* problem discussed in class for ideas. Consider 3 quantities for each vertex u:

- size of SVC that definitely includes u,
- size of SVC that definitely excludes u,
- size of SVC either including or excluding u.

Use a recursive postorder traversal to simplify your implementation (your algorithm will then run in $O(|V|)$ time).

Input format: The tree T will be given to you in the usual format supported by `read_graph()`.

Output format: The size of your minimum vertex cover (a single integer).

Additional problem (if you can't solve the above):

Write a program that takes a given graph G and a set of vertices S, and determines whether S is a vertex cover of G. [5]

Input format: A graph G will be given to you in the usual format supported by `read_graph()`. This will be followed by a single non-negative integer N, denoting the number of test cases, and N more lines corresponding to the test cases. Each test case will consist of a positive integer n, followed by the indices of n vertices.

Output format: For each test case, you should print one line consisting of the serial number of the test case (starting from 1), followed by `TRUE` or `FALSE`, depending on whether the given set is a vertex cover of G or not.

Q3. (20 marks) Even if you cannot write a complete, correct program, you will get partial credit if you can write correct functions for the parts written in blue.

Consider a grid that has 4 columns and n rows, as shown in the example below, with an integer written in each square. Write a program that marks **at most** $2n$ squares (out of the $4n$ squares that are present) with a **X**, so as to maximize the sum of the integers in the marked squares. There is one constraint: no two horizontally or vertically adjacent squares may be marked **X** (it is acceptable for diagonally adjacent squares to be marked).

HINT: Start with $n = 1$, i.e., consider a single row in isolation. This row can contain only positive numbers, only negative numbers, or some combination. Enumerate **all** possible permissible *patterns* in which the squares in this column can be marked; you should get 8 distinct valid ways ($p_0, p_1, p_2, \ldots, p_7$, say) to mark a single row. Devise a suitable representation for these patterns in your program. Write a function to determine which of these 8 patterns is the 'best' for a given row in isolation, i.e., maximizes the sum of the integers in the marked squares.

Next, consider $n = 2$. Call two patterns p_i and p_j *compatible* if they can occur in adjacent rows without violating the specified constraint. Construct a compatibility matrix for all $\langle p_i, p_j \rangle$ pairs.

Now suppose we have solved the problem for the first k rows, and need to consider the final $k+1$-th row.

Suppose I want to use pattern p_5 to mark the last row. Use the compatibility matrix that you have computed to determine which patterns (compatible with p_5) may be used to mark the second last row. Which of these options forms the best combination with p_5?

Similarly, consider the other patterns for marking the last row.

Now, for each of the 8 possible ways to mark the last row, we know the best we can do in combination with all the previous rows. Which of these 8 possible ways is the best considering **all the rows**?

More formally, we consider subproblems consisting of only the first k rows (for $0 \leq k \leq n-1$). The possible solutions to each subproblem can be assigned a type, corresponding to the type of the pattern occurring in the last row. Using the notions of compatibility and type, design an $O(n)$-time dynamic programming algorithm for computing the maximum sum obtainable by an optimal marking.

Input format: The value of n, followed by n rows, each containing 4 integers.

Output format: The maximum sum of the integers in the marked squares.

Example:

-66	-54	15	-11
82	26	36	7
94	-10	-31	77

Consider just the first row. We can place up to 2 marks, but since 3 out of the 4 numbers are negative, the maximum is obtained by placing only 1 **X** on 15.

Consider just the second row in isolation. The maximum sum is obtained by marking 82 and 36 (total = 118).

Now consider the first two rows. If we choose 15 from the first row, we can't choose 36 from the second row, and the best we can get is 15+82+7 = 104. Thus, we are better off choosing *nothing* from the first row, and 82 and 36 from the second row, i.e., we place 2 out of the 4 permitted marks, and get a score of 118.

When the third row is also included, the maximum obtainable sum turns out to be 212 (= 15 + 26 + 94 + 77).