

Family Name:..... Other Names: .....

Student ID:..... Signature .....

## COMP 261 : Test 1

2 April 2026

### Instructions

- Time allowed: **120 minutes**
- Attempt **all** the questions. There are 120 marks in total.
- Write your answers in this test paper and hand in all sheets.
- If you think a question is unclear, ask for clarification.
- This test contributes 30% of your final grade.
- You may use dictionaries and calculators.
- You may write notes and working on this paper, but make sure your answers are clear.

### Questions

### Marks

1. Grammars and Parse Trees

[17]

2. Coding a Parser

[23]

3. Ambiguous Grammars

[20]

4. Minimum Spanning Trees

[20]

5. Disjoint Sets (Union-Find)

[12]

6. Network Flow Analysis

[28]

TOTAL:

**Question 1. Grammars and Parse Trees****[17 marks]**

Consider the following grammar that describes a simple language for recording sensor readings.

In this grammar

- Non-terminals are in uppercase; terminals are enclosed in quotation marks,
- [...] means one or more repetitions of what is in the brackets.
- NUM matches any terminal that is a non-negative integer.

```
LOG ::= [ RECORD ]+
RECORD ::= "(" ID ":" VALUES ")"
ID ::= "sensor" NUM
VALUES ::= NUM [ "," NUM ]+
NUM ::= matches "[0-9]+"
```

Please note that space is used as the delimiter in Scanner.

**(a) [12 marks]**

Determine whether each sentence is valid or invalid according to the grammar above, and circle either "valid" or "invalid."

valid / invalid :	[ sensor 1 : 1 , 2 , 3 ]
valid / invalid :	( sensor 1 : 10 , 20 , 30 )
valid / invalid :	( sensor 4 : 5 3 2 )
valid / invalid :	( ( sensor 5 : 1 , 2 ) )
valid / invalid :	( sensor 3 : 8 , 9 ) , ( sensor 4 : 10 , 11 )
valid / invalid :	( sensor 2 : 3 ) ( sensor 1 : 2 )

(Question 1 continued on next page)

**(Question 1 continued)**

(b) [5 marks] Select a **valid** sentence from (a) and draw its Concrete Parse Tree according to the grammar. If none of them is valid, you may write your own sentence and draw its tree.



**Question 2. Coding a Parser****[23 marks]**

Suppose you are writing a parser for the grammar in Question 1.

```
LOG ::= [ RECORD ]+
RECORD ::= "(" ID ":" VALUES ")"
ID ::= "sensor" NUM
VALUES ::= NUM [ "," NUM ]+
NUM ::= matches "[0-9]+"
```

Your parser program (below) already includes some constants, three parse methods, and utility methods (require(..) and fail(..)).

The two sub-questions are:

- (a) to write more parse methods so that the top parse method should return an Abstract Syntax Tree of TNodes, or throw an exception if a sentence is invalid. We provide methods for `parseLog(..)`, `parseID(..)`, and `parseNum(...)`, please read them carefully.
- (b) to write the `LogNode`, `RecordNode` classes defining the different kinds of TNodes.

You are to complete the methods in Java, not pseudocode.

---

```
//----- constants (patterns) -----
static final Pattern LEFT_PAT = Pattern.compile("\\\\(");
static final Pattern RIGHT_PAT = Pattern.compile("\\\\)");
static final Pattern NUM_PAT = Pattern.compile("[0-9]+");
static final Pattern COMMA_PAT = Pattern.compile("\\\\,");
static final Pattern COLON_PAT = Pattern.compile("\\\\:");
static final Pattern SENSOR_PAT = Pattern.compile("sensor");

//----- parse... methods -----
/** Parses the rule: LOG ::= [ RECORD ]+ */
public TNode parseLog(Scanner s){
    List<TNode> logs = new ArrayList<TNode>();
    do {
        logs.add(parseRecord(s));
    } while (s.hasNext());
    return new LogNode(logs);
}
/** Parses the rule: ID ::= "sensor" NUM      Please note it returns an integer */
public int parseID(Scanner s){
    require(SENSOR_PAT, s);
    int num = parseNum(s);
    return num;
}
public int parseNum(Scanner s){
    if (s.hasNext("[0-9]")) {return s.nextInt();}
    fail("Expecting integer"); return -1;
}
//----- Utility methods -----
public void require(Pattern pat, Scanner s){
    if (s.hasNext(pat)) {s.next(); return;}
    fail("expecting "+ pat);
}
public void fail(String msg){ System.out.println(msg); throw new RuntimeException(msg);}
```

---

(Question 2 continued on next page)

**(Question 2 continued)**

(a) [13 marks] Complete the parseRecord(..), parseValues(..) methods below.

```
/** Parses the rule: RECORD ::= "(" ID ":" VALUES ")" */
public TNode parseRecord(Scanner s){

}

/** Parses the rule: VALUES ::= NUM [ "," NUM ]+ */
/** Please note it returns an Integer List */
public List<Integer> parseValues(Scanner s){

}

}
```

(b) [10 marks] Define the TNode classes

The parser in the previous sub-question needs the LogNode, RecordNode classes to create the Abstract Syntax Tree of TNodes. Suppose we want to print out "the total value of the sensor readings" for a valid sentence, for example,

---

```
Parser parser = new Parser();
Scanner s = new Scanner("( sensor 2 : 4 , 3 ) ( sensor 6 : 2 , 5 , 4 , 1 )")
TNode tree = parser.parseLog(s);
if (tree!=null) {System.out. println ( tree .getSensorTotal ());}
```

---

should print out 19, which is calculated as:  $4 + 3 + 2 + 5 + 4 + 1$

You will need to define a getSensorTotal() method for each kind of TNodes.

The TNode interface is provided as follows:

---

```
interface TNode{
    public int getSensorTotal ();
}
```

---

Grammar copied from previous question:

```
LOG ::= [ RECORD ]+
RECORD ::= "(" ID ":" VALUES ")"
ID ::= "sensor" NUM
VALUES ::= NUM [ "," NUM ]+
NUM ::= matches "[0-9]+"
```

```
//----- Node classes -----
```

```
class LogNode implements TNode{
```

```
}
```

```
class RecordNode implements TNode{
```

```
}
```

**Question 3. Ambiguous Grammars****[20 marks]**

Ambiguous grammars (where a text can have multiple different parse trees) cannot be parsed by deterministic top-down recursive descent parsers, like the parsers described in the lectures.

Consider the following grammar for a simple string manipulation language:

```
EXPR ::= EXPR "append" EXPR | EXPR "delete" NUM | STR
NUM  ::= matches "[0-9]+"
STR  ::= matches "\"[a-z]+\""
```

(a) **[8 marks]** Draw two alternative parse trees of the following expression according to this grammar:

"abc" append "ing" delete 2

Tree #1

Tree #2

(Question 3 continued on next page)

**(Question 3 continued)**

(b) [6 marks] Rewrite the grammar to remove ambiguity by ensuring that delete has higher precedence than append.

For example,

"aaa" append "bbb" delete 1

should be

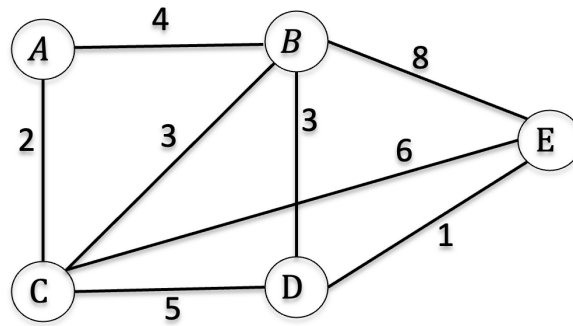
"aaa" append ("bbb" delete 1).

(c) [6 marks] Rewrite the grammar in (b) to use **right recursion only** and make sure it can be parsed by a one-token lookahead recursive descent parser, such as those introduced in the lectures.

**Question 4. Minimum Spanning Trees**

**[20 marks]**

Consider the following weighted undirected graph.



(a) [8 marks] Use **Prim's Algorithm** to find the Minimum Spanning Tree (MST), starting at node **A**. List the edges in the exact order they are added to the tree. If there is a tie in edge weights, pick the node that comes first alphabetically (e.g., choose B over C).

*Note: Please represent the edge you choose at each step as (X,Y), where X and Y are the two nodes connected by the edge, and represent the current tree you generate as the set of added tree nodes, e.g., {A,C,B}.*

(b) [3 marks] What is the total cost (sum of weights) of the MST?

(c) [9 marks] State whether each of the following statements is True or False. For each answer, give a one-sentence justification.

1. If a graph has unique edge weights, the MST is unique.

2. Kruskal's algorithm adds edges in increasing order of weight, provided they do not form a cycle.

3. An edge with the heaviest weight in a cycle is never part of any MST.

**Question 5. Disjoint Sets (Union-Find)****[12 marks]**

We are managing a collection of sets using the **Union-Find** data structure with the following pseudo-code for the Union operation. Initially, we have 6 isolated items (single-element trees): {1}, {2}, {3}, {4}, {5}, {6}. Each item is a root of its own tree with depth 0.

The pseudo-code for Union is:

```

Union(x, y):
  rootX = Find(x)
  rootY = Find(y)
  if rootX == rootY: return
  if rootX.depth < rootY.depth:
    rootX.parent = rootY
  else:
    rootY.parent = rootX
    if rootX.depth == rootY.depth:
      rootX.depth++

```

(a) **[8 marks]** Draw the forest (collection of trees) that results after the following sequence of operations:

1. Union(1, 2)
2. Union(3, 4)
3. Union(5, 6)
4. Union(2, 4)

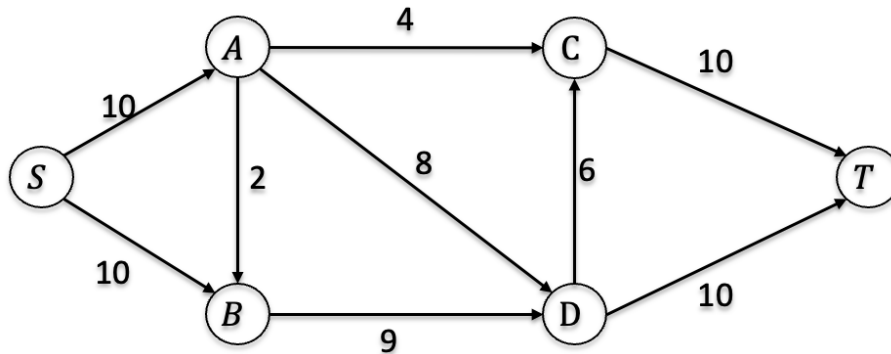
(Note: When unions occur between roots of equal depth, the first argument becomes the parent of the second argument based on the code above).

(b) **[4 marks]** In the resulting structure, what is the value returned by Find(4) (i.e., who is the representative root of 4)? How many "hops" (edges) are traversed to find it?

**Question 6. Network Flow Analysis**

**[28 marks]**

Consider the following directed flow network with Source **S** and Sink **T**. The numbers on the edges represent their **Capacities**.



(a) [6 marks] A student proposes a flow assignment where the flow on edge  $A \rightarrow D$  is **9 units**. Is this a valid flow? Explain why or why not by referring to the specific constraint that applies.

(b) [6 marks] Suppose we measure the flows around Node **D** and find the following values:

- Flow entering D:  $f(A \rightarrow D) = 5, f(B \rightarrow D) = 6$
- Flow leaving D:  $f(D \rightarrow C) = 4, f(D \rightarrow T) = 8$

Does this assignment satisfy the **Flow Conservation** (Balance) property for Node **D**? If not, explain why.

(c) [8 marks] We now run the **Edmonds-Karp algorithm** (which uses BFS to find augmenting paths) on the network (starting with 0 flow everywhere). Suppose the first path found by BFS is:

$$S \rightarrow A \rightarrow D \rightarrow T$$

- What is the **bottleneck capacity** of this path?
- List the **residual capacity** of edge  $A \rightarrow D$  (forward) and the capacity of the new backward edge  $D \rightarrow A$  after this augmentation.

(d) [8 marks] After the first augmentation in part (c), the algorithm continues.

1. Suppose the **second** shortest augmenting path found by BFS in the residual graph is:

$$S \rightarrow A \rightarrow C \rightarrow T$$

please calculate the amount of flow that can be pushed using this path.

2. Identify the **third** shortest augmenting path found by BFS (after the second one is processed) and the amount of flow pushed.

\*\*\*\*\*END OF TEST 1\*\*\*\*\*

\*\*\*\*\*

**SPARE PAGE FOR EXTRA ANSWERS**

Cross out rough working that you do not want marked.  
Specify the question number for work that you do want marked.