M269 Prsntn25J TMA03

Notes

Phil Molyneux

13 May 2026

M269 Prsntn25J TMA03
Phil Molyneux
M269 25J TMA03 Topics
Q 1
Q 2
Question 3
Question 4
Question 5

M269 25J TMA03

Topics

- Part 1 Qs 1 – 2 (58 Marks)
- Part 2 Qs 3 – 5 (42 Marks)
- Q1 Recursion and trees (36 Marks)
- Q2 Backtracking (22 Marks)
- Q3 Complexity classes (10 Marks)
- Q4 Turing machine (28 Marks)
- Q5 Computability (4 Marks)

M269 25J TMA03 Topics

Q 1

Q 2

Question 3

Question 4

Question 5

M269 25J TMA03

Common Marking Points

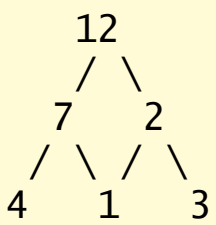
- **Common Marking Points**
- `allowed` must not report any non-allowed Python usage to get a bare paas
- `ruff` must present no infelicities to get a distinction

Question 1

Preamble

- Using binary trees to represent *triangles* of number

(36 marks)



- Note that this is *not* a binary tree
- So using the Binary Tree ADT to construct it is misleading
- Also represented as `12//7 2//4 1 5`
- At each level `r` there are `r` integers
- The *triangle* is similar in layout to Pascal's triangle but below we have multiplication instead of addition operations
- **Task** find the smallest product of integers, going from the top to the bottom of the triangle, moving left or right to the next row.
- For the example triangle, the smallest product is $12 \rightarrow 2 \rightarrow 1 = 24$, whereas the path $12 \rightarrow 7 \rightarrow 4 = 336$

Question 1

Code

```
ONE = leaf(1)
ROOT7 = join(7, FOUR, ONE)       # FOUR and FIVE in m269_tree
ROOT2 = join(2, ONE, FIVE)       # common ONE node with ROOT7
ROOT12 = join(12, ROOT7, ROOT2) #
ROOT3 = join(3, FIVE, FOUR)      # another example: 3 // 5 4

q1_public_tests = [
    # case,           triangle,  product
    ("example",       ROOT12,    24),
    ("subtree at 7",  ROOT7,     7),
    ("subtree at 2",  ROOT2,     2),
    ("no 1 nodes",    ROOT3,     12),
]
```


Q 1
Q1(a)
Q1(b)
Q1(c)
Q1(d)
Q1(e)
Q1(f)
Q 2
Question 3
Question 4
Question 5

Q1(a)

Tests

- **Task**: Write up to five tests for this problem. **(6 marks)**
- **Criteria**
  - All tests have different purpose
  - Tests cover two edge cases
  - At least 4 tests
  - `ruff` reports no errors

```
# Write here your code to construct test trees.

q1_your_tests = [
    # case,     triangle,   product
    # Add your tests here.
]
```

Q1(a)

Tests (2)

- **Suggestions**
- Smallest inout
- Smallest output with all numbers > 0
- Positive and zero numbers
- All zero
- Backtracking helps
- Backtracking = Exhaustive
- 3-row greedy fails
- 4-row greedy fails
- Note: You need to know how the algorithm works — see next section

Q1(b)

Description

- Consider an algorithm that, from the root downwards, keeps choosing the smallest child, multiplying the integers it goes through.
- What kind of algorithm is this? **(4 marks)**
- Does it compute the smallest product correctly?
- The above algorithm makes *local* choices at each stage — what is that called
- Remember *Greedy algorithms hardly ever work*

M269 25J TMA03 Topics

Q 1

Q1(a)

Q1(b)

Q1(c)

Q1(d)

Q1(e)

Q1(f)

Q 2

Question 3

Question 4

Question 5

Q1(d)

Code

- **Task**: Write a recursive Python function to find the smallest product. **(6 marks)**
- This part only assesses that the function is well-defined; the correctness will be assessed in Q1(e).
- Remember: *Greedy algorithms hardly ever work*
- **Criteria** Type annotations present and (syntactically) correct

```
# Don't change the function name, even if you used a different name in Q1(c).
def q1_smallest_product(triangle):
    pass

# Passing your own tests from Q1(a) is not a criterion here nor in Q1(e), but
# if your code fails some of your tests, you should check the code and the tests.
test(q1_smallest_product, q1_your_tests)
```

WQ(e)

Code Correctness

- **Code correctness** **(8 marks)**
- In Q1(b) you have to have a counter example to demonstrate that the greedy algorithm can get the *wrong* answer

Q1(f)

Complexity

- Use magic %timeit **(8 marks)**
- See %timeit documentation
- See also timeit

```
def triangle(height: int) -> Tree:
    """Return a triangle 1, 2 2, 3 3 3, ... with the given height.

    Preconditions: height > 0
    """
    # construct triangle bottom up to do the joins
    below = [leaf(height)] * height
    for row in range(height - 1, 0, -1):
        current = []
        for column in range(row):
            left = below[column]
            right = below[column + 1]
            current.append(join(row, left, right))
        below = current
    return below[0]

for height in range(1, 1): # replace the second 1 with an appropriate number
    numbers = triangle(height)
    # replace this line with code to measure your function's run-time on 'numbers'
```

M269 Prsntn25J TMA03

Phil Molyneux

M269 25J TMA03 Topics

Q 1
Q1(a)
Q1(b)
Q1(c)
Q1(d)
Q1(e)
Q1(f)

Q 2

Question 3

Question 4

Question 5

Q2

Preamble

- Same as Q1 but zeroes not allowed **(22 marks)**

Q2(a)

Algorithmic technique

- Explain why backtracking can solve this problem and why it may be more efficient than exhaustive search for the smallest product. **(6 marks)**

Q2(a)

Algorithmic technique (2)

- **Backtracking** a path sequence is expanded by one item at a time
- When the product reaches or exceeds the the smallest path so far, we can cease extending this path
- Backtracking allows us the prune the search space

Q2(b)

Algorithm efficiency

- Construct two triangles such that backtracking is more efficient than exhaustive search for one triangle but not for the other one.
- Explain why that happens **(6 marks)**

Q2(b)

Algorithm efficiency

- `3 // 2 2 // 1 1 1` Backtracking more efficient than exhaustive search
- `1 // 2 2 // 3 3 3` Backtracking behaves like exhaustive search

Q2(c)

Algorithm

- Explain how to solve the problem with backtracking, by answering the following questions, with brief justifications. **(6 marks)**
  - Which of the four templates from Section 25.2 applies to this problem?
  - What is a candidate and what is an extension?
  - When is an extension compatible with the current candidate?
  - When is a candidate a solution?
  - What are the local and global constraints?

Q2(b)

Algorithm efficiency (2)

- Section 25.2 Backtracking templates
  - S25.2.1 Constraints on sequences
  - S25.2.2 Best sequence
  - S25.2.3 Constraints on sets
  - S25.2.4 Best set
- Candidate is a path — extension is a subtree yet to explore
- An extension is compatible with a candidate if it is the left or right subtree of the last node in the path
- A candidate is a solution if the path reaches a leaf.
- There is no global constraint on a path except you could take the view that a global constraint is for the candidates product to be less than the current smallest product. The local constraint on a node is that it is the left or right child of the previous node.

Q3

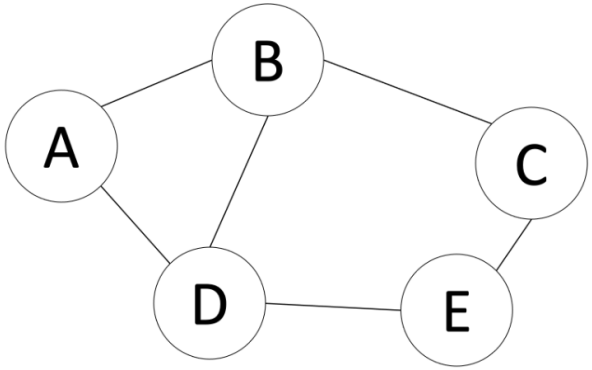
Preamble

- A city council wants to install 360° surveillance cameras at junctions, to film all adjoining streets for the safety of its residents.
- You can assume the camera's range is enough to film all the way along a street.
- The council wants to know if a given number of cameras are enough to film all streets.
- Here is the decision problem, formulated on graphs.
- **Function**: filming
- **Input**: *city*, an undirected graph; *cameras*, an integer
- **Preconditions**: *city* has $n > 0$ nodes; $0 \leqslant$ cameras $\leqslant n$
- **Output**: *enough*, a Boolean
- **Postconditions**: *enough* is true if and only if *city* has *cameras* nodes so that each edge is attached to at least one of those nodes

Q3

Preamble (2)

- Here is an example graph, with nodes representing junctions, and edges representing streets.



- If *cameras* is 1, then *enough* is false, because there is no single node attached to all edges.
- In other words, one camera is not enough to film all streets, no matter at which junction it is installed.
- However, for *cameras* = 3, *enough* is true, because installing cameras at junctions B, D, E will film all streets.
- Prove that this decision problem is in class NP by answering the following parts.

Q3(a)

Certificate

- Describe in general a certificate for a problem instance (a graph *city* and an integer *cameras*) that has a true output.
- Give as example a certificate for the graph above and cameras = 3.

M269 25J TMA03 Topics

Q 1

Q 2

Question 3

Q3 Preamble

Q3(a)

Q3(b)

Question 4

Question 5

Q3(a)

Certificate (2)

- A Certificate for a Decision Problem is a sample solution that can be checked in polynomial time
- An example here might be a sequence with *cameras* nodes [B, D, E] (there are other possibilities, such as a set of nodes)
- That it is a solution can be verified in polynomial time
- The procedure has to show that all edges are covered and do that in polynomial time
- Could be done by copying *city* and removing all nodes in *cameras* and confirming this removes all edges
- Remember to use Big-Theta notation $\Theta(n^2)$ and $\Theta(n+e)$

Q3(b)

Verifier

- Explain why certificates for this problem can be verified in polynomial time, by outlining the verifier's algorithm and analysing its complexity. **(6 marks)**

Q3(b)

Verifier

- For each operation in the verifier procedure provide the complexity
- The procedure has to copy the graph, $\Theta(n+e)$, remove all nodes in the certificate, $\Theta(|\text{cameras}| \times n)$ and check all edges have been removed, $\Theta(e)$
- The above may not be correct — the essence is that evry edge is connected to at least one node in the certificate
- TODO: Q3(b) finish

Q4

Preamble

- **Task** You are asked to write a Turing machine that checks if the input is a well-formed string
- The input should have the following properties:
  - be a sequence of a least two symbols
  - start with a single- or double-quote mark
  - end with the same quote mark
  - not have the start/end quote mark in between.
- The input tape consists of one or more symbols from the set 'a', "'", '"', followed by infinite blanks.
- The input tape can be written as a list literal, like ['"', "a", '"'], or as a Python string converted to a list, like list('"a"').
- If you want the input tape to include both kinds of quotes, use a triple-quoted string (Section 4.2.1).
- The output is True if the symbol sequence is a well-formed string and False if not
- TODO: Q4 missing parts

Q4(a)

Test table

- Write up to 8 tests for this problem.

```
# Don't change the public tests, other than the debug parameter.
q4_public_tests = [
    # case,              input tape,     debug,  output
    ("1-letter string",  list('"a"'),    False,  [True]),
    ("2-letter string",  list("'aa'"),   False,  [True]),
    ("no end quote",     ['"'],          False,  [False]),
    ("' within '",       list("'a'"),    False,  [False]),
]

q4_your_tests = [
    # case,              input tape,     debug,  output
    # Add your tests here.
]

# Don't change these lines: they help you check your tests.
IN_SYMBOLS = {"a", "'", '"'}
OUT_SYMBOLS = {True, False}
check_tm_tests(q4_your_tests, IN_SYMBOLS, OUT_SYMBOLS, max_tests=8)
```

M269 Prsntn25J TMA03
Phil Molyneux
M269 25J TMA03 Topics
Q 1
Q 2
Question 3
Question 4
Q4 Preamble
Q4(a)
Q4(b)
Q4(c)
Question 5

Q4(b)

Algorithm

- Outline an algorithm for a Turing machine that solves the problem
- **Distinction points**
  - The algorithm is broadly correct for all four properties.
  - The outline explains the purpose of all states other than the start state.
  - The outline is not unnecessarily verbose.
  - The algorithm uses at most 4 states other than the start state.
  - The algorithm does a single pass over the input, without modifying it.

Q4(b)

States

- **States**

M269 25J TMA03 Topics

Q 1

Q 2

Question 3

Question 4

Q4 Preamble

Q4(a)

Q4(b)

Q4(c)

Question 5

Q4(c)

Code

- Your code is inserted below as a Python dictionary

```
# Don't change the name of the Turing machine.
your_tm = {
    # write the transitions here in the form
    # (state, symbol): (new_symbol, LEFT or RIGHT or STAY, new_state),
}

# Don't change this line: it helps you check your machine against the criteria.
check_tm(your_tm, IN_SYMBOLS, OUT_SYMBOLS, max_states=5)

# Passing your own tests from Q4(a) is not a criterion here nor in Q4(d), but
# if your machine fails some of your tests, you should check the machine and the tes
test_tm(your_tm, q4_your_tests)
```

Q5

Computability

- **Task** Explain if the following decision problem is computable. **(4 marks)**
  - Given an algorithm in the form of a Python function and given two Boolean expressions Pre and Post, which represent the pre- and post-conditions respectively, we want to know if the algorithm is correct (Section 2.9.3).

Q5

Computability (2)

- This is asking the question: Given any input satisfying the preconditions, will the output satisfy the postconditions ?
- You could appeal to Rice's theorem or show some undecidable problem can be reduced to it