shuspaceAssessment Brief

|  |  |  |
| --- | --- | --- |
| Module Leader: | | Level 7 |
| Module Name Data Structures & Algorithm | | Module Code 55-709698 |
| Assignment Title Task A: An application of Searching Techniques | | |
| Assignment type  Individual assignment | Weighting 25% | Magnitude  Completing this task should take around 5 hours. |
| Submission date/time  Tuesday 25th February 2025 at 3:00pm UK time. | Blackboard submission: Yes  Turnitin submission: No | Format  Source code and text document. |
| Planned feedback date  Friday 7th March 2025 | Mode of feedback  You will receive written feedback on your submission through Blackboard. | In-module retrieval available?  No |
|  | | |
| Module Learning Outcomes  * Design algorithms in an appropriate abstract notation and characterise them and describe their attributes using appropriate metrics. * Compare and contrast similar algorithms and describe their relative merits and demerits. * Optimise implementations of algorithms. * Construct and manipulate the complex data structures required of many algorithms using appropriate programming languages. | | |

# **Introduction**

In this task, you will use Python to develop a program that solves the "aggressive cow problem." Your solution will involve implementing a suitable searching algorithm to optimize the placement of cows, demonstrating your understanding of algorithmic problem-solving in real-world scenarios. Additionally, you will need to analyse the complexity of your solution to evaluate its efficiency. This analysis should be well-documented in a detailed report, explaining the reasoning behind the algorithm usage and the performance of your solution under different conditions. Your tutors will assess and grade your submission based on the correctness of the solution, the efficiency of the algorithm, and the quality of your report. This task accounts for 25% of the total grade for the assessment. Your submission will be evaluated using the rubrics given at the end of this document. You must complete all exercises by the submission date given in the table on page one of this assignment specification.

## **AVOIDING PLAGIARISM**

If you use algorithms, code samples, or inspiration from sources such as discussions on Stack Overflow or Reddit, or blogposts on sites such as Medium, you must include a citation in your code as a comment. You are expected to use a range of source material of your own when developing your solution. Doing so without citation is plagiarism. Doing so with citation is good practice.

# **Artificial Intelligence and Academic Integrity – AI&AI**

It is important you do not use AI tools to generate an assignment and submit it as if it were your own work. Our regulations state:

Contract cheating/concerns over authorship: This form of misconduct involves another person (or artificial intelligence) creating the assignment which you then submit as your own. Examples of this sort of misconduct include: buying an assignment from an ‘essay mill’/professional writer; submitting an assignment which you have downloaded from a file-sharing site; acquiring an essay from another student or family member and submitting it as your own; attempting to pass off work created by artificial intelligence as your own. These activities show a clear intention to deceive the marker and are treated as misconduct.

Further guidance is available here: [https://blogs.shu.ac.uk/assessment4students/preparing-to-submit-work/#AI](https://eur02.safelinks.protection.outlook.com/?url=https%3A%2F%2Fblogs.shu.ac.uk%2Fassessment4students%2Fpreparing-to-submit-work%2F%23AI&data=05%7C02%7Cm.jacobi%40shu.ac.uk%7Ce43edfef59d143757cc808dc9077c828%7C8968f6a1ac13472fb899f7316e439f43%7C0%7C0%7C638544091847747773%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C0%7C%7C%7C&sdata=Mu35s2bWJFR7ESDUdLI4GvdJiCwl6CyBS42osK5Z468%3D&reserved=0)

# **TASK A: AN APPLICATION OF SEARCHING TECHNIQUES.**

## **PROBLEM STATEMENT: AGGRESSIVE COW PROBLEM**

Robert, a farmer in Sheffield, has a new barn where he’s arranged stalls in a straight line for his cows. However, after moving the cows into these stalls, he noticed they were not comfortable and became aggressive, potentially due to the close proximity. In an effort to prevent the cows from fighting or harming each other, Robert needs to reorganize their stall assignments. His goal is to ensure there is a minimum distance between any two cows, but he wants this minimum distance to be as large as possible under the current layout of stalls. This means that he is aiming to place the cows in such a way that the smallest distance between any two cows is maximized, ensuring the cows are spaced far enough apart to minimize aggression, but also making the best possible use of the available stalls. Each stall can have only one cow at any point of time, i.e., two cows cannot be placed in the same stall.

Few important methods and data items to be incorporated in your design is described below, read them carefully to build your solution.

1. Stall\_position[x1, x2, x3,… xn] : Each element in this array denotes the position of a stall available for reassignment, located along a straight line at positions x1, x2, x3,… xn. The length of Stall\_position[ ] represents the number of stalls available for reassignment. Use Python’s random() function to generate the items for this array, then sort them in ascending order.
2. Cow\_count : This represents the total number of aggressive cows in the barn that need stall assignment.
3. aggressiveCows() : This method selects a distance value, “d”, to be verified by distance\_OK() method. It accepts Stall\_position[ ], total stall count and total cows as its argument. The method finds the largest possible minimum distance between two cows and returns it.
4. distance\_OK() : The method to check if it is possible to position the cows ‘d’ distance apart. It accepts Stall\_position[ ], ‘d’ and total cows as its argument. Returns true if it is possible to have chosen distance ‘d’, false otherwise.
5. cowsPlaced : Variable to keep count of cows placed in the stall using the ‘d’ distance.
6. stall\_count : Variable to store total number of stalls available for assignment.

The aggressiveCows() method can be implemented using two approaches. First approach (basic method) - choosing distance ‘d’ using minimum and maximum stalls. Second approach: use binary search to choose the distance ‘d’.

## **FIRST APPROACH (BASIC METHOD)**

Step1: Iterate for each possible distance from 1 to the maximum distance, i.e., max(stall position) - min(stall position) and call the function distance\_OK() to check if the arrangement is possible.

Step 2: For each distance, begin by placing cows at the first stall and check if

If (current position – previous position >= distance)

True: update previous position = current position, check next distance.

Step 3: Update cowsPlaced whenever a cow is placed i.e cowsPlaced+=1;

Step 4: Verify

if ( cowsPlaced >= Cow\_count )

True: the arrangement is possible.

Otherwise: arrangement not possible.

Step 5: Repeat Step 2 to 4 for the next distance, stop when arrangement not possible return the answer.

### **Example**

If there are three aggressive cows, with stall positions 1, 4, 6, 7, and 10 available then :

Cow\_count = 3

Stall\_position = [1, 4, 6, 7,10]

### **Method**

Here is the explanation of the method for finding the arrangement of cows for each possible distance.

Now, the range of the distance between any two cows is:

Range: [ 1, max(Stall\_position) - min(Stall\_position) ]

= [1, (10-1)]

= [1, 9]

So, we need to  verify the arrangement of cows for each possible distance from 1 to 9.

**Minimum distance- 1**

Place first cow at position 1 and the second cow at position 4 as:

current position – previous position >= distance

i.e., 4-1 >=1 is true.

The third cow can be placed at position 6.

Therefore, distance can be 1.

**Minimum distance- 2**

Place the first cow at position 1, second cow at position 4, third cow at position 6.

Therefore, distance can be 2.

**Minimum distance- 3**

Place the first cow at position 1, the second cow at position 4, and the third cow at position 7. Therefore, distance can be 3.

**Minimum distance- 4**

Place the first cow at position 1, the second cow at position 6, and the third cow at position 10.

Therefore, distance can be 4.

**Minimum distance- 5**

Place the first cow at position 1, the second cow at position 6, but we can’t place the third cow as the distance will be less than the minimum distance.

i.e., current position – previous position >= distance

i.e., 10 - 6 < 5 if we choose 10.

Therefore, this arrangement is not possible i.e., when distance = 5. So we update distance as 4. Now we check further for minimum distance from 6 to 9. The arrangement is demonstrated in the table below.

### **Arrangement Table**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Minimum Distance** | **Stall Positions** | | | | | **Arrangement Valid?**  **(Yes/No)** |
| 1 | 4 | 6 | 7 | 10 |
| 1 | Cow1 | Cow 2 | Cow 3 |  |  | Yes |
| 2 | Cow1 | Cow 2 | Cow 3 |  |  | Yes |
| 3 | Cow1 | Cow 2 |  | Cow 3 |  | Yes |
| 4 | Cow1 |  | Cow 2 |  | Cow 3 | Yes |
| 5 | Cow1 |  | Cow 2 |  |  | No |
| 6 | Cow1 |  |  | Cow 2 |  | No |
| 7 | Cow1 |  |  |  | Cow 2 | No |
| 8 | Cow1 |  |  |  | Cow 2 | No |
| 9 | Cow1 |  |  |  | Cow 2 | No |

### **Conclusion**

Therefore, from the table it is clear that to ensure the cows do not hurt each other in the new barn, they must be placed in the stall at distance of 4 from one another.

## **SECOND APPROACH(BINARY SEARCH)**

Step1: Iterate for each possible distance from 1 to the maximum distance, i.e., max(stall position) - min(stall position), apply binary search to take mid-point as the distance ‘d’.

Step 2: Call the function distance\_OK() to check if the arrangement is possible with distance ‘d’.

Step 3: For each distance, begin by placing cows at the first stall and check if

If (current position – previous position >= distance)

True: update previous position = current position, check next distance.

Step 4: Update cowsPlaced whenever a cow is placed i.e cowsPlaced+=1;

Step 5: Verify

if ( cowsPlaced >= Cow\_count )

True:the arrangement is possible, update low = mid + 1

Otherwise: arrangement not possible, high = mid-1

Step 6: Repeat Step 2 to 5 for the next distance, stop when further arrangement not possible return the answer.

### **Example**

If there are three aggressive cows, with stall positions 1, 4, 6, 7, 10 available then:

Cow\_count = 3

Stall\_position = [1, 4, 6, 7,10]

### **Method**

Here is the explanation of the method for finding the arrangement of cows for each possible distance.

Now, the range for the distance between any two cows is:

 Range: [ 1, max(Stall\_position) - min(Stall\_position) ]

= [1, (10-1)]

= [1,9]

So, we need to  verify the arrangement of cows for each possible distance from 1 to 9.

**First midpoint**

Low = 1, High = 9, Mid = (1 + 9) / 2 = 5.

Minimum distance = 5,

Place the first cow at position 1, the second cow at position 6. Now, we cannot place the third cow; hence the arrangement is not possible.

Update High = Mid – 1, therefore, High = 4.

**Arrangement Table**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Minimum Distance** | **Stall Positions** | | | | | |
| 1 | 4 | 6 | 7 | 10 | **Arrangement Valid?**  **(Yes/No)** |
| 5 | Cow1 |  | Cow2 |  |  | No |

**Second midpoint**

Low = 1, High = 4,Mid = (1 + 4) / 2 = 2.

Minimum distance = 2.

Place the first cow at position 1, and the second cow at position 4, the third cow at position 6. Hence this arrangement is possible.

Update Low = Mid + 1, now Low = 3.

**Arrangement Table**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Minimum Distance** | **Stall Positions** | | | | | |
| 1 | 4 | 6 | 7 | 10 | **Arrangement Valid?**  **(Yes/No)** |
| 2 | Cow1 | Cow2 | Cow3 |  |  | Yes |

**Third midpoint**

Low = 3, High = 4, Mid = (3 + 4) / 2 = 3.

Minimum distance = 3.

Place the first cow at position 1, the second cow at position 4, the third cow at position 7. Hence arrangement is possible.

Update Low = Mid + 1, now, Low = 4.

**Arrangement Table**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Minimum Distance** | **Stall Positions** | | | | | |
| 1 | 4 | 6 | 7 | 10 | **Arrangement Valid?**  **(Yes/No)** |
| 3 | Cow1 | Cow2 |  | Cow3 |  | Yes |

**Fourth midpoint**

Low = 4, High = 4, Mid = (4 + 4) / 2 = 4.

Minimum distance = 4.

Place the first cow at position 1, the second cow at position 6, The third cow at position 10. Hence arrangement is possible.

Update Low = Mid + 1, now, Low = 5.

**Arrangement Table**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Minimum Distance** | **Stall Positions** | | | | | |
| 1 | 4 | 6 | 7 | 10 | **Arrangement Valid?**  **(Yes/No)** |
| 4 | Cow1 |  | Cow2 |  | Cow3 | Yes |

Now, Low > High, therefore we can stop the binary search and return the answer i.e., largest minimum distance as 4.

### **Conclusion**

Therefore, from the table it is clear that to ensure the cows do not hurt each other in the new barn, they must be placed in the stall at distance of 4 from one another.

## **Questions**

### **Question 1**

Write a Python program to solve the “AGGRESSIVE COW PROBLEM” using two different approaches mentioned in the previous section and execute them. First, implement the solution with the “First Approach”, where you attempt to place the cows, verifying the distance at each stall. Then, implement the “Second Approach”, which refines the solution through binary search to find the optimal minimum distance between cows.

For each approach, you must incorporate the appropriate functions, such as for placing cows and validating distances, along with data items mentioned in the problem statement. Assume we always start by placing the first cow at the first stall available in both the approaches.

### **Question 2**

Once your solution is implemented, perform an evaluation of the time complexity for both approaches. Compare their performance to understand which is more efficient and suitable for larger input sizes, providing insights into their practical usage. You must perform a thorough performance analysis of both the approaches. You may use tables, diagrams etc. to supplement your analysis.

## **What to Submit?**

As part of your submission for “Task A” you are required to upload the following to the Blackboard VLE by the submission deadline.

1. An executable Python source code.
2. A written document (100-150 words): Microsoft Word **or** PDF format for the task.

# **ASSESSMENT CRITERIA (TASK-A)**

## **Criteria & Weighting**

|  |  |
| --- | --- |
| Criteria List | Weighting% |
| Criterion 1: An assessment of understanding of the problem. | 30% |
| Criterion 4: Implementation | 40% |
| Criterion 5: Result analysis and discussion | 30% |

## **Rubrics**

| **Criteria and Weighting** | **Levels of achievement** | | | | |
| --- | --- | --- | --- | --- | --- |
| **Expert**  **(Max: 100%)** | **Practitioner**  **(Max: 79%)** | **Apprentice**  **(Max: 69%)** | **Beginner**  **(Max: 59%)** | **Fail**  **(Max: 49%)** |
| Criterion 1 – An assessment of understanding of the problem. | Exceptional work of the highest quality, demonstrating excellent knowledge and understanding.  Problem-solving strategy is accurately and clearly described. | Very high-quality work, demonstrating excellent knowledge and understanding.  However, there are some minor gaps in obvious areas where it could be extended/ improved. | High-quality work, demonstrating good knowledge and understanding.  However, the work presented struggles to identify key factors, has several minor errors, and there are some obvious areas where it could be improved/ extended. | Demonstrates limited knowledge and understanding, missing major aspects of the problem.  Critical areas of the task are neglected, causing gaps in the overall problem understanding. | Some attempt has been made to tackle the task.  Fails to understand the problem or misinterprets key elements of the problem. |
| Criterion 4 – Implementation | Outstanding implementation has been demonstrated for the work. A comprehensive program developed using object-oriented programming, auxiliary methods for outputs, and readable code.  The program developed produces accurate results for the problem/solution. The outputs of the program are well documented in the program artefact.  The program is easy for the assessor to run.  Wherever projects are applicable, use GitHub for the source code repository (available as public), which is well-documented in the report. | Excellent implementation demonstrated for the work. The program developed for the work produces accurate results for the problem/solutions. Some auxiliary methods are written to support the main objective of the work.  Program outputs and results are saved in the working environment and well documented in the report. Readable code.  The program is runnable by the assessor with minimal environment settings. | Accurate results and outputs produced by the program developed for the work. The source code for the work is evident in the report and included in the submission. | The program developed for the work can produce reliable results for the problem/solutions. Some components required for the work are not evident in the implementation artefact. | The program developed for the work is incomplete, and invalid or arguable outputs are produced for result analysis and discussion. Limited evidence was demonstrated in the implementation artefact. |
| Criterion 5 – Result analysis and discussion | Exceptional work for the results demonstrated in the report. The presentation of the results in the report looks very neat and tidy (whichever projects are applicable). Result tables or diagrams are named accordingly. All result tables and diagrams are exceptionally analysed and discussed comprehensively. Wherever projects are applicable, the discussion covers the algorithm complexity and data structure efficiency based on the empirical experiment results. | An excellent result analysis was conducted for the work and was well documented in the report. Appropriate tables and diagrams are used to demonstrate the results (if applicable). A comprehensive discussion of the results is well-written with a convincing tone and evidence. | The result analysis and discussion of the work are fairly evident in the report. Some diagrams (if applicable) that support the results and discussion should be included for marks. | The result analysis and discussion presented in the report need more evidence. Very basic content and points are discussed in the work. | Result analysis for the work is almost not evident in the report. No evidence base is written for the discussion section. |

# University Grading Descriptor

A screenshot of a computer

Description automatically generated