RMIT UNIVERSITY

Algorithms and Analysis
COSC2123/3119
Dynamic Programming in Action: The Knapsack-Maze Challenge

Assessment Type	Individual assignment. Submit online via GitHub.
Due Date	Week 11, Friday May 23, 8:00 pm. A late penalty will apply to assessments submitted after 11.59 pm.
Marks	30

1 Learning Outcomes

This assessment relates to four learning outcomes of the course which are:

- CLO 1: Compare, contrast, and apply the key algorithmic design paradigms: brute force, divide and conquer, decrease and conquer, transform and conquer, greedy, dynamic programming and iterative improvement;
- CLO 3: Define, compare, analyse, and solve general algorithmic problem types: sorting, searching, graphs and geometric;
- CLO 4: Theoretically compare and analyse the time complexities of algorithms and data structures; and
- CLO 5: Implement, empirically compare, and apply fundamental algorithms and data structures to real-world problems.

2 Overview

Across multiple tasks in this assignment, you will design and implement algorithms that navigate a maze to collect treasures. You will address both fully observable settings (where treasure locations are known) and partially observable ones (where treasure locations are unknown), which requires strategic exploration and value estimation when solving the maze. Some of the components ask you to critically assess your solutions through both theoretical analysis and controlled empirical experiments to encourage reflection on the relationship between algorithm design and real-world performance. The assignment emphasizes on strategic thinking and the ability to communicate solutions clearly and effectively.

Important Notes

Please read all the following information before attempting your assignment.

- This is an *individual* assignment. You may not collude with any other person (or people) and plagiarize their work. Everyone is expected to present the results of their own thinking and writing. Never copy another student's work (even if they "explain it to you first") and never give your written work to others. Keep any conversation high-level and never show your solution to others. Never copy from the Web or any other resource or use Generative AI like ChatGPT to *generate solutions or the report*. Suspected cases of collusion or plagiarism will be dealt with according to RMIT Academic integrity policy.

- This assignment requires submitting both a written report and your implemented code via **GitHub** Classroom. You must include your report file in the repository and tag your submission appropriately (with the tag `submission`) before the deadline. Detailed instructions for submission via GitHub Classroom are provided in the assignment repository.

- Before implementing any code, please carefully read the `README` section provided in the skeleton code repository and add or modify code **only** within the files explicitly marked with the comment `/* PLEASE UPDATE THIS FILE. */`. Any changes outside these marked sections may negatively affect the reliability of your solution and could cause your submission to fail our automated tests.

- You are expected to properly use git version control and **commit regularly, with clear and meaningful messages** to reflect the progress of your development. Submitting your entire solution in a single commit, or in just a few large commits, is considered poor practice. Please note that marks from automatic tests will be adjusted based on how well you follow these practices. A submission that works perfectly but shows no evidence of a thoughtful development process will receive zero marks. Full marks will only be awarded when correct implementation is paired with clear evidence of good version control and development discipline.

- Respect the word and page limits specified for the report. Content beyond the specified limits will not be read and **will not be considered in marking**.

- Strictly adhere to the assignment submission deadline. The deadline will not be extended under any circumstances, except in the event of natural disasters or similar emergencies.

- Please check the Ed forum discussion for further clarifications about specifications. In addition, Week 7 workshops are dedicated to walk you through the key elements of the assignment. Elham will also go through different aspects of the assignment each week, so even if you cannot make it to the lectorials, make sure to check the recordings.

- In the submission (your PDF file for the report) you will be required to certify that the submitted solution *represents your own work only* by including the following statement:

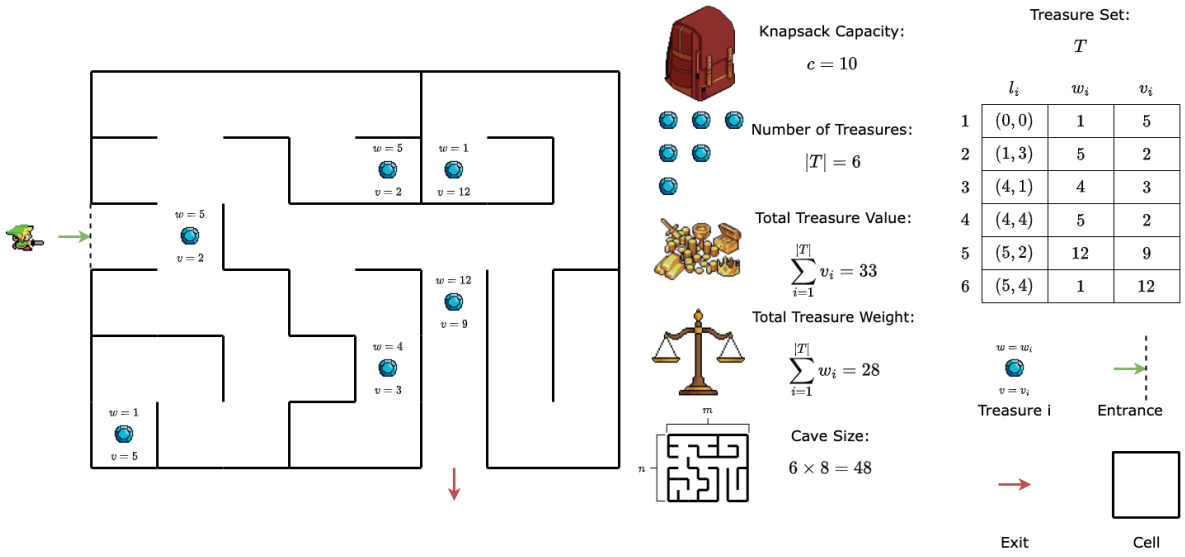
> *I certify that this is all my own original work. If I took any parts from elsewhere, then they were non-essential parts of the assignment, and they are clearly attributed in my submission. I will show I agree to this honor code by typing "Yes":*

3 Assessment details

The assignment is broken up into a number of tasks, to help you progressively complete the project. Please note that although completing one task can assist with subsequent tasks, each task can be attempted independently. Moreover, the tasks are designed in a way that even in cases where your implementation does not fully achieve the expected output, you can still discuss and present the theoretical aspects of your algorithmic design through your report.

Motivation

You are to assist an adventurer searching for treasure. They have heard tales about a maze filled with valuable treasures. The Adventurer's goal is to enter the maze, find and collect as many treasures as they can, and then leave through a different exit. Once they enter, the entrance will close behind them, so they must find another way out!



	l_i	w_i	v_i
1	$(0,0)$	1	5
2	$(1,3)$	5	2
3	$(4,1)$	4	3
4	$(4,4)$	5	2
5	$(5,2)$	12	9
6	$(5,4)$	1	12

Figure 1: A breakdown of all the information available to The Adventurer. The layout of the maze (made up of $n \times m$ cells), as well as entrance/exit locations and treasure locations (including the value and weight of each treasure). The book tells The Adventurer the information in the centre column - number of treasures, total value and total weight of all treasures.

As shown in Figure 1, in their possession The Adventurer has:

- an enchanted bag with carrying capacity c (the maximum weight it can hold);
- a mystical map, which tells them:
  - the layout of the maze (which is fully connected but may have cycles and is always rectangular of size $n \times m$ with square cells) including the entrances, exits, and walls;
  - the location, l_i, weight w_i and value v_i of each treasure i; The location of a treasure is in the form of (row, col), where $0 \leq col \leq m-1$ and $0 \leq row \leq n-1$. As can be seen in the Treasure Set T in Figure 1, the row numbers start from bottom to top and the column numbers start from left to right. The treasures are also sorted using a bottom-to-top, left-to-right sweep.

- a magical book, which contains:
  - the number of treasures in the maze $|T|$;
  - the total value of all the treasures in the maze $v = \sum_{i=1}^{|T|} v_i$; and
  - the total weight of all the treasures in the maze $w = \sum_{i=1}^{|T|} w_i$.

Objective

Your objective is to assist The Adventurer in gathering as many valuable treasures as their knapsack can carry. Mathematically, you wish to:

$$\text{maximise} \sum_{i=1}^{|T|} v_i s_i \qquad \text{subject to} \sum_{i=1}^{|T|} w_i s_i \leq c \text{ and } s_i \in \{0, 1\}$$

$$\text{minimise } |P| \qquad \text{subject to } l_i \in P \text{ if } s_i = 1$$

where:

- $s_i = 1$ means the i^{th} treasure is picked up while $s_i = 0$ means the i^{th} treasure is left behind;
- $P = \langle (row, col), ... \rangle$ is the ordered multiset of cells The Adventurer plans to visit (where the first element is the entrance, the last element is the exit, and each element is adjacent to the previous element).

In other words, we wish to find the shortest path through the maze that maximises the value of our knapsack, subject to the condition that the total weight of the knapsack is less than or equal to its maximum carrying capacity.

Completing Tasks A, B, C, and D will take you through different methods for completing this objective. **Please read each Task carefully, and implement only what is asked in each Task.**

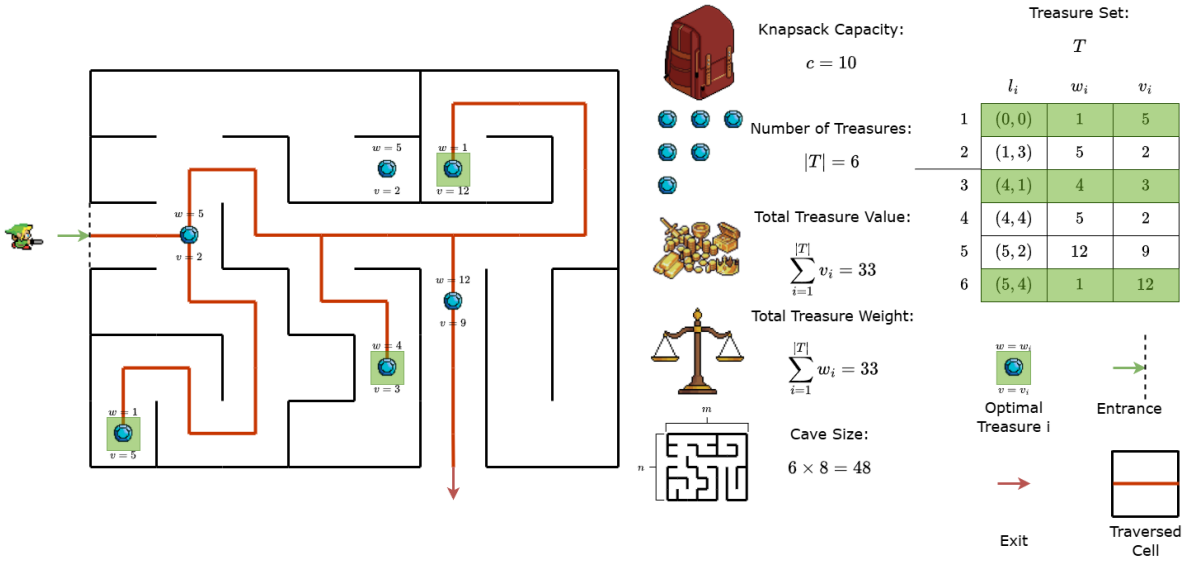


Figure 2: A solution for The Adventurer given the specific problem in Figure 1. We can see that the optimal treasures have been selected in the treasure set T, their positions highlighted on the map, and a route has been planned that takes The Adventurer through each cell containing an optimal treasure to collect before going through the exit.

Task A: Maximising Knapsack Capacity using Recursion (5 marks)

In order to accomplish the objective, it would probably be prudent to start by planning which treasures are the best to collect. To begin, put together a *recursive* strategy to find which combination of treasures would be best to collect: For the knapsack problem, the algorithm recursively calculates the optimal solution to sub-problems by considering each treasure's inclusion and exclusion.

Implementation Task - 5 marks

Implement the below algorithm for recursive knapsack using the skeleton code provided. Ensure that the input and outputs of the function are as described.

Algorithm 1 RecursiveKnapsack(T, c, k)

Input: T, a map of treasures where treasure IDs are mapped to location, weight, value tuples, i.e., $t_i : (l_i, w_i, v_i)$; a positive integer c representing the knapsack capacity; and a positive integer k, the number of treasures.
Output: A list of locations L_{opt} for the optimal selection of treasures, total weight w_{opt}, and total value v_{opt} of the selected treasures.

$L_{opt} \leftarrow \emptyset$; $w_{opt} \leftarrow 0$; $v_{opt} \leftarrow 0$
if $c = 0$ **or** $k = 0$ **then**
    return $L_{opt}, w_{opt}, v_{opt}$

$t \leftarrow T[k-1]$ ▷ Get the k^{th} treasure
$location \leftarrow t$.location
$weight \leftarrow t$.weight
$value \leftarrow t$.value

if $weight > c$ **then**
    return RecursiveKnapsack(T, c, $k-1$)

$(L_{inc}, w_{inc}, v_{inc}) \leftarrow$ RecursiveKnapsack(T, $c - weight$, $k-1$)
$(L_{exc}, w_{exc}, v_{exc}) \leftarrow$ RecursiveKnapsack(T, c, $k-1$)

if $v_{inc} + value > v_{exc}$ **then**
    $L_{opt} \leftarrow L_{inc} \cup \{location\}$
    $w_{opt} \leftarrow w_{inc} + weight$
    $v_{opt} \leftarrow v_{inc} + value$
else
    $L_{opt} \leftarrow L_{exc}$
    $w_{opt} \leftarrow w_{exc}$
    $v_{opt} \leftarrow v_{exc}$

return $L_{opt}, w_{opt}, v_{opt}$

Report Task - 0 mark

You are NOT required to write anything in your report for Task A.

Task B: Maximising Knapsack using Dynamic Programming (7 marks)

For Task A, you implemented a recursive algorithm to solve the knapsack problem in order to find the optimal treasures to collect when you enter the maze. As a fail-safe to ensure you have definitely found the best solution, you must implement another method using dynamic programming. If you are correct, both solutions should come to the same conclusion!
Whilst the approach in Task A required recursion to find solutions to sub-problems, dynamic programming allows you to store previous solutions to sub-problems and use them to solve future problems. We do this by filling in a dynamic-programming table such that:

$$F(i,\ j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ F(i-1,\ j) & \text{if } w_i > j \\ \max\{F(i-1, j-w_i) + v_i,\ F(i-1, j)\} & \text{else} \end{cases}$$

for $i \in \{0, 1, ..., |T|\}$ and $j \in \{0, 1, ..., c\}$. $F(i, j)$ is the most valuable subset of the first i treasures that fit into a knapsack with capacity j and $F(0,\ j) = F(i,\ 0) = 0$ makes up our initial conditions - the optimal solution if we have no treasures or a knapsack with a carrying capacity of 0.

Intuition

For each $F(i,\ j)$, we consider picking up treasure i for a knapsack with carrying capacity j. We therefore check the last optimal solution where the new treasure would be able to fit in our bag ($F(i-1,\ j-w_i)$) and see if this value (plus the value of the current treasure v_i) exceeds the optimal solution if the treasure is not picket up which is captured by $F(i-1, j)$. The optimal solution for the subproblem is the maximum of these two values. The index $F(|T|,\ c)$ will hold the maximum value the knapsack can take for all the treasures.

Implementation Task - 5 marks

In the provided skeleton code, implement the dynamic programming solution to the knapsack problem using **memory functions** (or lazy/sparse programming). Your function should take in exactly the same input arguments as Task A. It should return a tuple containing the location for the optimal treasures to take, the total weight of the items in the knapsack and the value (again, just as Task A does). Your function also needs to generate a csv file to save the dynamic programming table where each row correspond to the i_{th} treasure and every column (separated by a comma) corresponds to a capacity between 0 to c (a function to convert a dynamic-programming table to CSV file is provided for you). The snapshot in Figure 3 shows the **dynamic programming table using memory functions** for the example provided in Figure 1. You can find the relevant information regarding memory functions in the lecture notes and lecture recordings of Week 9.

Report Task - 2 marks

In your report, write the pseudo-code for this problem (use the pseudo-code in Task A as a guide for how to do this). Briefly explain the potential benefits and downsides of this approach against the approach in task A. Total page limit for Task B is **one page**.

Items \ Capacity	0	1	2	3	4	5	6	7	8	9	10
None	0	0	0	0	0	0	0	0	0	0	0
$(1,5)$	0	5	#	#	5	5	5	#	#	5	5
$(5,2)$	0	5	#	#	5	5	7	#	#	7	7
$(4,3)$	#	#	#	#	5	8	#	#	#	8	10
$(5,2)$	#	#	#	#	#	#	#	#	#	8	10
$(12,9)$	#	#	#	#	#	#	#	#	#	8	10
$(1,12)$	#	#	#	#	#	#	#	#	#	#	20

Figure 3: Example of a dynamic programming table constructed using memory functions. Cells marked with # indicate subproblems that were never computed during the execution. Non # entries represent the values computed and stored for the function $F(i, j)$. Backtracking will give the optimal items to pick up.

Task C: Analysis of the Complete Problem (8 marks)

Now you are sure which treasures to collect, you can begin finding an optimal path P to enter the maze through the entrance, pick up the *optimal* treasures, and leave the maze through the exit. Luckily, the mystical map calculates this for you! But you'd like to know how it works...

In this task, you will empirically compare the two algorithms you implemented in Tasks A and B. You are provided with a function *findItemsAndCalculatePath* in the skeleton code. This function: (1) Takes as input the maze structure, entrances, exits, treasure layout and knapsack strategy (recursive or DP); and (2) executes a path-finding solution to find the shortest possible path from an entrance to an exit while collecting all the treasures that result in the optimal output. Your task is to (i) understand this function and to (ii) conduct a meaningful empirical analysis of your two strategies in Task A and B.

Implementation Task - 0 mark

While you are required to implement and run tests as part of this task, no marks are awarded for the implementation itself. Marks will be given based solely on the quality of your analysis, experimental design, and discussion in the report (see bellow).

Report Task - 8 marks

Your report should include:

1. **Algorithm Complexity Analysis:** Strongly justify (perhaps with reference to the equations and/or pseudo-code):
   (a) the theoretical time complexities of both knapsack solutions from Tasks A and B; and
   (b) the theoretical time complexity for the function *findItemsAndCalculatePath*.
2. **Empirical Design**: Highlight which variables you believe are important when calculating the algorithmic complexity of the function *findItemsAndCalculatePath* with the recursive/dynamic-programming knapsack solution. Explain why this is the case, and why you chose to ignore other variables (if any).

3. **Empirical Analysis:** Provide a thorough analysis using one or more plots to compare the running time of *findItemsAndCalculatePath* with the two different knapsack strategies from Task A and B. How would you interpret your observation?

4. **Reflection:** Discuss whether your empirical results align with your theoretical complexity predictions. If there are discrepancies, explain possible causes — e.g., implementation details, constant factors, etc.

Total page limit for Task C is **two pages**.

Task D: Maximizing Knapsack Value without Having the Map (10 marks)

In this task, we consider a scenario where The Adventurer explores a maze to collect treasures *without knowing their exact locations*. The full maze structure (maze layout) is known in advance, but the locations of treasures are not. The adventurer only knows:

- The total number of treasures, $|T|$, present *somewhere* in the maze; and
- the total combined value of all treasures, v; and
- treasures combined weight w and minimum weight 1.

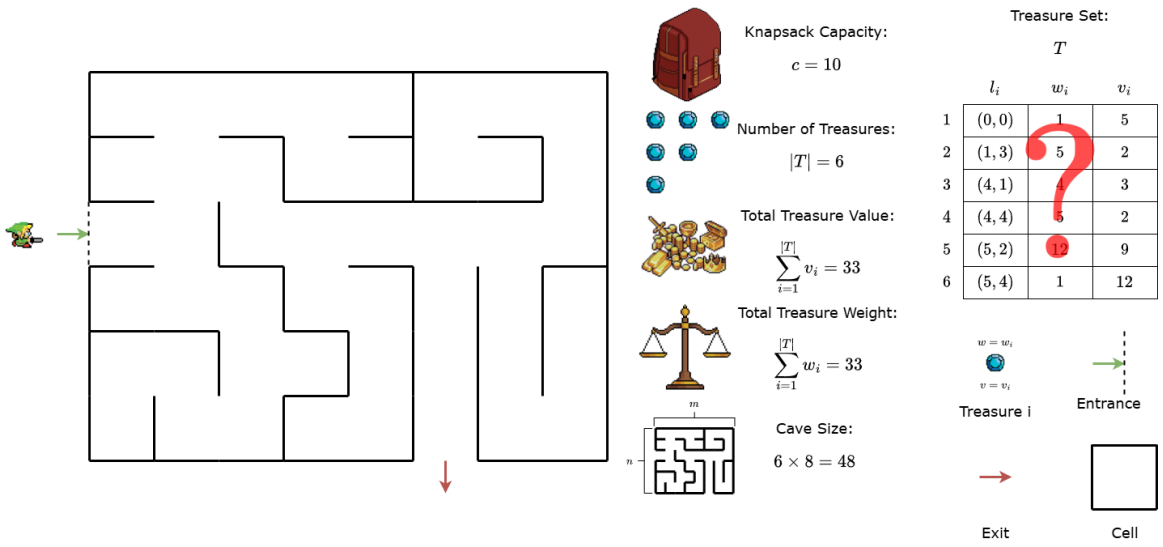


Figure 4: The new problem presented to The Adventurer. The maze layout, total number of treasures, and their total value and weight are known in advance. However, the locations of each these treasures, as well as their specific attributes, is a mystery.

The Adventurer assumes that (i) there is only one entrance and one exit in the maze, and (ii) all cells have equal chance to have a treasure. In other words, the initial probability of a cell containing a treasure is equal to $\frac{|T|}{n \times m}$. On top of all this, the maze is full of terrors - The Adventurer wishes to minimise exploration (where possible). They must navigate the maze and search the cells strategically to collect treasures and maximize the **net reward** objective:

$$Reward = \text{knapsack value} - \text{cells explored}$$

Your task is to design and implement an algorithm to employ a treasure search after entering the maze which aims to maximize collected treasure value while minimizing the number of cells

explored. Your solution must strategically decide which cells to explore, taking into account the trade-off between the value of potential treasures and the exploration cost associated with entering additional cells. The implementation should use the prior knowledge of the maze's layout, total treasure count, cumulative treasure value, and weight distribution, to effectively balance exploration and exploitation.

Implementation Task - 5 marks

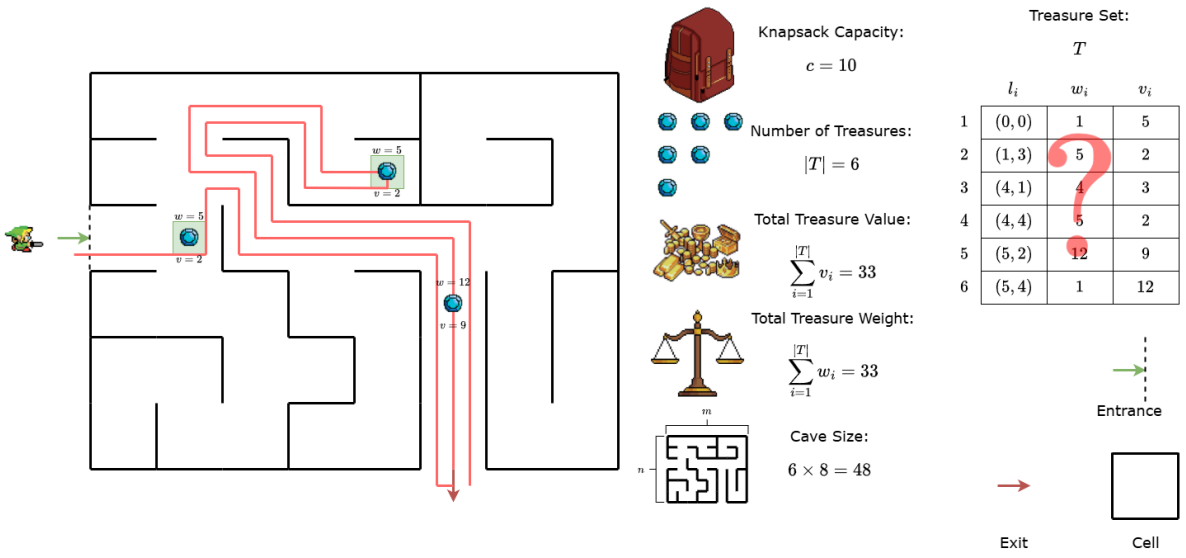
Implement your solution in the provided code skeleton for Task D. Your solution receives as input, the maze, T, v and w, and outputs the sequence of cells explored from the entrance to the exit (repetitions are allowed, i.e. re-entering a cell you have already visited does not count as a new explored cell) as well as the selected treasures. Your solutions will be tested against our own solution, and marks will be awarded based on a weighting system in terms of the total $Reward$ gained.

For a solution to be valid, it must:

1. have a path P that begins at the entrance;
2. have a path P that ends at the exit;
3. have a path P such that each sequential element is adjacent, so if $P_k = (i, j)$ then:

   $$P_{k+1} = (i+1, j) \text{ or } (i-1, j) \text{ or } (i, j+1) \text{ or } (i, j-1)$$

   and P_{k+1} and P_k do not have a wall between them.
4. the weight of the items in the knapsack must not exceed its capacity; and
5. the solution cannot use prior knowledge of item locations - **a cell can only be checked for an item after it has been visited** (and so must be added to the path P).



	l_i	w_i	v_i
1	(0, 0)	1	5
2	(1, 3)	5	2
3	(4, 1)	4	3
4	(4, 4)	5	2
5	(5, 2)	12	9
6	(5, 4)	1	12

Figure 5: A valid solution to the hidden treasure problem. During exploration, The Adventurer uncovers 3 treasures. 2 can fit in the knapsack (combined weight of 10 and combined value of 4), and they are taken before The Adventurer leaves the maze. Thus, whilst the path length is 41, the number of unique visits is only 18, giving the final reward to be $4 - 18 = -14$.

Report Task - 5 marks

The written report must include:

1. **Algorithms Design**: Regardless of whether or not you have implemented your solution, in your report describe your exploration and treasure-selection strategy. Include pseudo-code and explain **how** your approach balances between expected treasure value and cell cost. Clearly discuss the complexity of your solution with regard to the input;

2. **Assumptions** Explain how your algorithm handles different probability models for treasure placement. Clearly discuss if and how your solution can handle cases where there is a spatial bias in the distribution of the treasures, e.g., the probability of having a treasure is higher for cells that are more distant from the entrance and exits. See Figure 6 for some potential examples. Please note that your answer does not need to focus on finding a solution for a specific probability distribution. You are only required to discuss **if and how your solution could be impacted by the change** in the initial assumption.

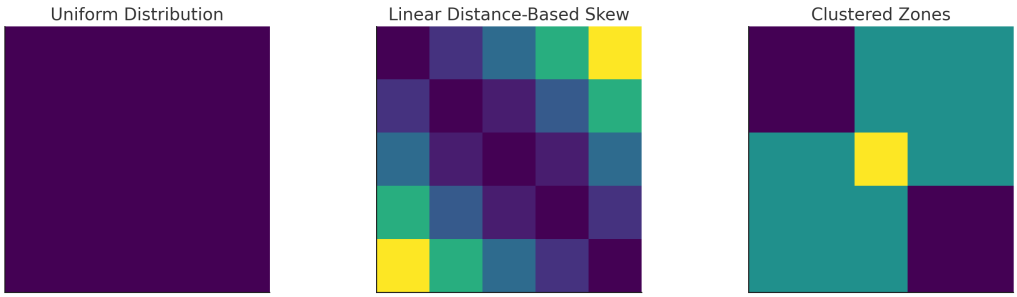


Figure 6: Example of different probability distributions for treasure placement. The uniform distribution (left) implies that all cells have an equal probability to have a treasure (4% in a 5 by 5 maze). The other two examples show different scenarios where the yellow cells have the highest probability to have a treasure followed by green and blue cells.

Total page limit for Task D is **two pages**.

4 Assessment

The project will be marked out of 30. The assessment in this project will be broken down into several parts for each Task. The criteria discussed in Table 1 will be considered when allocating marks.

5 Late Submission Penalty

Late submissions will incur a 10% penalty on the total marks of the corresponding assessment task per day or part of day late, i.e, 3 marks per day. Submissions that are late by 5 days or more are not accepted and will be awarded zero, unless special consideration has been granted. Please ensure your submission is correct (more details in the FAQ Resources available via the Ed Forum) as re-submissions after the due date and time will be considered as late submissions. We strongly advice you submit **at least one hour before the deadline**. Late submissions due to slow Internet will not be looked upon favorably, even if it is a few minutes late. Any claim of late submission due to slow Internet will require documentation and evidence that submission attempts were made at least **one hour before the deadline**.

6 Academic integrity and plagiarism (standard warning)

Academic integrity is about honest presentation of your academic work. It means acknowledging the work of others while developing your own insights, knowledge and ideas. You should take extreme care that you have:

- Acknowledged words, data, diagrams, models, frameworks and/or ideas of others you have quoted (i.e. directly copied), summarised, paraphrased, discussed or mentioned in your assessment through the appropriate referencing methods.
- Provided a reference list of the publication details so your reader can locate the source if necessary. This includes material taken from Internet sites. If you do not acknowledge the sources of your material, you may be accused of plagiarism because you have passed off the work and ideas of another source without appropriate referencing, as if they were your own.

RMIT University treats plagiarism as a very serious offense constituting misconduct. Plagiarism covers a variety of inappropriate behaviours, including:

- Failure to properly document a source
- Copyright material from the internet or databases
- Collusion between students

For further information on our policies and procedures, please refer to this link.

7 Getting Help

There are multiple venues to get help. There are two weekly consultation hours (see Canvas or the Ed Forum for time and location details). In addition, you are encouraged to discuss any

issues you have with your Tutor. We will also be posting common questions on the Ed Forum and we encourage you to check and participate in the discussion. However, please **refrain from posting solutions**, particularly as this assignment is focused on algorithmic design. Please carefully read the FAQ page on the Forum Etiquette or download it here to make sure you do not mistakenly post something that would be considered inappropriate.

Table 1 - Assessment Criteria

Criteria	**Ratings**	**Pts**
Task A Implementation Automatic Testing Results	**4 to 0 pts** Number of passed tests normalized by appropriate software engineering practices (subject to valid implementation)	4 pts
Task A Implementation Quality of Implementation	**1 to 0 pts** Valid, correct and clean implementation	1 pt
Task B Implementation Automatic Testing Results	**4 to 0 pts** Number of passed tests normalized by appropriate software engineering practices (subject to valid implementation)	4 pts
Task B Implementation Quality of Implementation	**1 to 0 pt** Valid, correct and clean implementation	1 pt
Task B Report	**Complete - 2 to 1.5 pts** The report is well-organized and clearly describes the algorithm. The pseudo-code is included, and the steps are clear and sound. The discussion about the advantages and disadvantages of the two knapsack algorithms are clearly discussed and the justifications are sound. **Satisfactory - 1.5 to 1 pts** The report is generally understandable but may have some unclear sections. The Pseudo-code is provided but it includes ambiguous steps. Comparison is provided between the two approaches, but it is not clear or sound at times. **Incomplete - 1 to 0 pts** The report does not include a proper pseudo-code and/or any clear discussion regarding the algorithm. The comparison is not provided or is not accurate.	2 pts

Task C Report	**Complete - 8 to 6 pts** The report is well-organized and all four aspects are comprehensively addressed: (i) complexity analysis is correct and the justification is clear and sound, (ii) the design of the test is sound and clearly communicated. The inclusion and exclusion of all parameters are well-justified, (iii) the results of the empirical analysis are presented and communicated clearly, and (iv) the comparison between theoretical and empirical analysis is clear and sound. **Satisfactory - 6 to 4 pts** The report is generally understandable but may have some unclear sections in regards to one or two of the listed aspects above. **Incomplete - 4 to 0 pts** The report does not include a proper discussion on the empirical design and/or the empirical analysis lacks depth and proper justification. The theoretical analysis is incorrect and/or not well-explained and the report lacks a sound comparison between the theoretical and empirical analyses.	8 pts
Task D Implementation Automatic Testing Results	The marking will be awarded following a comparison with our solution. All bellow bands are determined based on the number of passed tests normalized by appropriate software engineering practices (subject to valid implementation): **Efficient & Optimal solution - 4 pts** the solution achieves comparable net rewards compared to our solution **Efficient but sub-optimal solution - 3 pts** the solution explores comparable number of cells but with lower values of the treasure collection **Inefficient but Optimal solution - 2 pts** the solution explores more number of cells but achieves comparable and/or higher values for the treasure collection **Inefficient & sub-optimal solution - 1 pts** the solution works but explores more number of cells while achieving worse value in the knapsack	4 pts
Task D Implementation Quality of Implementation	**1 to 0 pts** Valid, correct and clean implementation. Finding a path from an entrance to an exit is NOT a valid implementation.	1 pt

Task D Report	**Complete - 5 to 3.5 pts** A well-articulated exploration and selection strategy is provided which includes detailed and correct pseudo-code. Clear explanation of how the algorithm balances expected treasure value against cell cost as well as a thorough and correct complexity analysis with respect to input size is provided. The report clearly discusses how the changes in probability distribution would impact the strategy and performance of the algorithm and the provided discussion is sound and relevant to the proposed design. **Satisfactory - 3.5 to 2 pts** Strategy is explained but lacks depth or precision. Pseudo-code and complexity analysis are present but may have minor errors or be incomplete. Some discussion regarding the impact of probability distributions on the algorithm is provided, but it lacks depth and/or is not clearly relevant to the proposed design. **Incomplete - 2 to 0 pts** The report provides no clear strategy presented, or lacks logical structure. Pseudo-code is missing or significantly flawed. Complexity analysis has incorrect reasoning and no meaningful discussion of how assumptions affect the algorithm is provided.	5 pts