

Dungeon Game

Mid Semester Project, IE402 Optimization, Autumn Semester 2021 - DAICT, Gandhinagar, 25th October 2021

Nilay Shah - 201901026
B.Tech. 3rd year
DAICT, Gandhinagar
201901026@daiict.ac.in

Kanishk Patel - 201901035
B.Tech. 3rd year
DAICT, Gandhinagar
201901035@daiict.ac.in

Harsh Patel - 201901036
B.Tech. 3rd year
DAICT, Gandhinagar
201901036@daiict.ac.in

Ayush Prajapati - 201901099
B.Tech. 3rd year
DAICT, Gandhinagar
201901099@daiict.ac.in

Sahil Chaudhari - 201901148
B.Tech. 3rd year
DAICT, Gandhinagar
201901148@daiict.ac.in

Abstract

In this article, we will discuss the very interesting dungeon game problem and its popularities among passionate gamers and a few algorithms listed in the document to solve our dungeon game challenge. For the initial parts, we will introduced the dungeon game, the history behind the dungeon game, the problem statement of the dungeon game and the later parts include possible ways of solving the problem using popular algorithms, pseudo-code and supporting flowchart of the solution. In the end result comparison of all the algorithms and complexity analysis will be done so that efficiency of each algorithm can be proved. At the end of the document, all the codes for algorithms are provided as a part of appendix.

Index Terms—Dungeon; Game Master; Recursion; Flowchart; Dynamic Programming; Optimal Path

I. INTRODUCTION

A. What is a Dungeon Game?

In Dungeon Game, the meaning of "Dungeon" is Closed-Off area in the gaming perspective. Dun-

geons are generally found in enclosed areas like caves, rooms, castles, or fortresses. In the Role-Playing Game, the Player generally encounters enemies in the dungeon. [11] [12]

B. Different Types of Dungeon Game:

The dungeon has been very historic and later on the implementations as role-playing games were made. One of the games that came in 1974 is The Dungeons & Dragons, which is a role playing game that is based on the worlds of swords and sorcery. It's like picturing a falling castle in the dark forest and then a fantasy adventure with limitless possibilities. This game allows a way of determining the consequences of the player's actions. Actions are determined by dice where anything is possible but some things become more probable than others. Dungeons were designed by people who are called Dungeon Master. It takes a lot of time and energy to create and run these amazing games that players love and remember. Dungeon Master facilitates the game, directs the monsters and non-player characters, and also narrates the story. [14] [15]

The type of problem having dungeons and storytelling became very interesting. The conditions, rules, and tricky dungeons make the game interesting.

One of the dungeon games - Dungeon! (Published in 1975) is very similar to Dragons & Dungeons and Dungeon Game Puzzles/Problems. Which features a map of a simple six-level dungeon with hallways, rooms, and chambers. Players need to move on the board seeking to defeat the monsters and claim the treasure. Greater treasures are located in deeper levels of the dungeon, along with tougher monsters. Players need to choose different character classes with various abilities. This game aims for sub-parts to be the first to return to the beginning chamber with a set value of treasure. [17]

The Classic Dungeon is also a dungeon game. Which is a board game with a simple introduction: travel into a dungeon, kill the monsters, take their treasure, and escape. This game is also very similar to the D&D style with the fantasy theme, and the players take the role of adventurers. But unlike in D&D, where the adventurers work together, in this game, the adventurers work against each other to kill as many monsters and gather as much treasure as fast as they can, and escape before the other players. It has a feel of a dungeon hack with no complex rules and also dice-chucking to push the luck. [18]

Currently, the dungeon and dragon game which is a popular example of a dungeon game has a tremendous result which is "A new infographic released by Wizards of the Coast shows just how popular the game was in 2020. The infographic breaks down stats about the Dungeons and Dragons player base, revealing that the game has achieved more than 50 million players to date." [19]

C. Aim

To get familiarized with dungeon games and understand different algorithms to solve the problem. We will look upon different approaches like Recursion, dynamic Programming (Top –Down and Bottom Up) and Binary Search to solve the problems. We will also look upon the complexity analysis of various algorithms and result after the compilation and running all the codes.

D. Objective

- To understand the dungeon and the dungeon game and a variety of algorithms to solve the problem.
- To get familiar with the dungeon game problems.
- To implement all the optimal and efficient solution with their respective complexity analysis.
- To get the optimal path after performing optimality test.

E. Possible algorithms to solve dungeon game Problem

1) *BFS*: The breadth-first search algorithm for the graph is the same as the breadth-first traversal algorithm for trees. Here, we visit the nodes that are on the same path rather than going to depth for any particular path. In the graph, there may be cycles so we may encounter the same node again. To avoid this we use a boolean array to keep track of visited nodes. [3] [4]

2) *Recursion*: In the recursion process, a function calls itself directly or indirectly, and here the corresponding function is called a recursive function. In this approach of the solution, the problem is divided into sub-parts and then each subpart is solved and the value is returned to the calling function. Here, function calls itself that's why for termination we have to put a base condition. [7]

3) *Dynamic Programming*: Dynamic Programming is the main methodology to solve our problem because it solves it very efficiently. Dynamic programming is mainly an optimization over a simple recursion. The advantage of dynamic programming is that we divide problems into smaller subproblems and solve and store the result of each subproblem, so later when it is needed the result need not be calculated again and this reduces

complexity by a significant amount. [8] [9]

II. LITERATURE REVIEW

A. From where is the Name Dungeon derived?

The word dungeon comes from the French word donjon, (Also called dungeon) which means "keep" and refers to a castle's main tower. The word initially appeared in English at the beginning of the 14th century, when it had the same meaning as donjon. The proper original definition of "keep" is still in use, however, it has mostly been overused in popular culture and has come to imply a cell or "oubliette." An oubliette, also known as a bottle dungeon, is a basement area that can only be accessed through a hatch or hole in a high ceiling. [2]

An oubliette is frequently referred to as a "bottle dungeon." It has a narrow entrance at the top, and the area below is often so small that it's impossible to lie down, but other versions have a larger cell. Although many real dungeons are nothing more than a single plain room with a heavy door or access only through a hatchway or trapdoor in the floor of the room above, the use of dungeons for torture, as well as their association with common human fears of being trapped underground, have made dungeons a powerful metaphor in a variety of contexts. [17]

Dungeons, as a whole, have become associated with underground complexes of cells and torture chambers. As a result, the number of true dungeons in castles is often exaggerated to interest tourists. Many chambers described as dungeons or oubliettes were water-cisterns or even latrines.

B. History

Dungeons are common elements in modern fantasy literature, related tabletop, and video games. The most famous examples are the various Dungeons & Dragons media. In this context, the word "dungeon" is often used broadly to describe

any labyrinthine complex (castle, cave system, etc) rather than a prison cell or torture chamber specifically. A role-playing game involving dungeon exploration is called a dungeon crawl.

The dungeon was written in 1975 or 1976 by Don Daglow, then a student at Claremont University Center. The game was an unlicensed implementation of the new tabletop role-playing game Dungeons & Dragons (D&D) and described the movements of a multi-player party through a monster-inhabited dungeon. Players chose what actions to take in combat and where to move each character in the party.

Although the game was nominally played entirely in text, it was also the first game to employ a line of sight graphics display. Its use of computer graphics consisted of top-down dungeon maps that showed the portions of the playfield the party had seen, allowing for light or darkness, the different "infravision" abilities of elves, dwarves, etc.

This advancement was possible because many university computer terminals had switched by the mid-1970s to CRT screens, which could be refreshed with text in a few seconds instead of a minute or more. Earlier games printed game status for the player on Teletype machines or a line printer, at speeds ranging from 10 to 30 characters per second.

While Dungeon was widely available via DECUS, it was picked up by fewer universities and systems in the mid-1970s than Daglow's earlier Star Trek video game had been in 1971. Years later DECUS distributed another game named Dungeon, that was in fact a version of Zork, a text adventure game that would later become the model for early MUDs. A third game called Dungeon was released on PLATO in 1975, by John Daleske, Gary Fritz, Jan Good, Bill Gammel, and Mark Nakada. [1]

III. PROBLEM STATEMENT

Basically what happens in most dungeon games is that players accept challenges from the game master (GM) or dungeon master who is the creator of games and complete the objective of the challenge facing several obstacles and up-downs. Players are allowed to attempt any action valid inside the world of the dungeon, however, because of glitches and bugs unexpected can happen. GM can keep track of players' actions and based on that GM can predict the next actions to be undertaken by players. Based on this GM can set obstacles throughout the dungeon and control the flow. However, for this a lot of work is needed so generally, GM uses predefined dungeons for the objective quest. [10]

We will look at the small problem of the dungeon subsection, which represents the dungeon game very effectively. There are some Dungeon Game puzzles. It's generally on a 2D grid. The story goes like the knight has to save the princess from demons by traveling through the dungeons. And the knight has to travel from the top left corner of the room to the bottom right corner of the room because the princess is trapped there by demons. His health is affected by demons who have been situated by GM. During the movement from one cell to another cell, the knight has to reach the princess quickly only by moving rightwards or downwards from a cell set in the room. A knight needs some health to rescue the princess but he must need to take minimum health at the starting point and therefore he needs to choose an optimal path through dungeons. The conditions, rules, and tricky dungeons make the game interesting. As we are implementing a dungeon game, the Game master will be implemented in our program itself which creates all the obstacles or traps to make the Knight's work hard.

Let's simplify our dungeon game problem into a much simpler problem statement and try to solve it using the appropriate algorithm.

Given a grid with each cell consisting of positive, negative or no points i.e., zero points. We can move across a cell only if we have positive points (> 0). Whenever we pass through a cell, points in that cell are added to our overall points.

Our main goal is to find minimum initial health to reach the cell $(m-1, n-1)$ from $(0, 0)$. [6]

1) Constraint:

- 1) From a cell (i, j) we can move to $(i+1, j)$ or $(i, j+1)$.
- 2) We cannot move from (i, j) if your overall points at (i, j) is ≤ 0 .
- 3) We have to reach at $(n-1, m-1)$ with minimum positive points i.e., > 0 .

Note: The traps are denoted as a negative integers, which reduces Knight's health. The traps can be range from -1000 to 0 and the power ups range from 0 to 1000. The range can be changed as per Dungeon Master's requirements .

The matrix dimensions are taken by input form and the dungeon master generates random dungeon matrix, which generates all the traps and power-ups in our program. You can think of the solution to this problem like finding the min-cost to reach the destination cell but that will not guarantee you the minimum initial health. Constraint (2) can be violated in that case.

As you can see, the minimum health required at a given cell can be solved by the minimum health required to reach from rightward cell to destination or the minimum health required from downwards cell to destination by adding the original cell cost with it. If this value is greater than or equal to zero then we need a minimum of 1 health and if it's less than zero then we will require absolute(ans) +1 health to reach from cell to destination. In this way, we will approach the problem.

A. Example



-2	-3	3
		
-3	-10	1
-5	30	-4
		

Fig. 1. Dungeon Game with optimal path

B. Flow chart

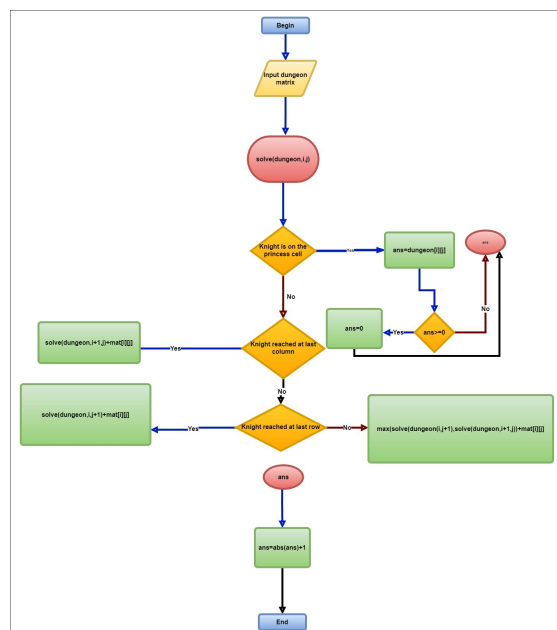


Fig. 2. Flowchart

The above flowchart denoted as below gives all the information to solve the dungeon problem.

This flowchart supports recursive approach, which is the most basic approach to solve this kind of problems.

In Fig.1 example of a dungeon game, you will need a minimum of health which is 6 to reach the destination. The optimal path has been shown below for the optimal(minimum) health. Note that, negative values are denoted as traps and positive values are denoted as power ups.

IV. Different algorithmic approaches to solve dungeon game Problem

We will implement the dungeon game problem with different algorithms paradigms. The algorithms provides the same result but as per the time complexity and space complexity are concerned, they give different results which is our main interest. We will solve the dungeon game problem with the following approaches:

A. Recursive Approach

Recursive approach is the most general paradigm to solve the dungeon game problem. The very first approach to solve any optimal path problem is basically this Recursive approach. The algorithm states that as we know knight has two paths to go from one cell to another. One is rightwards and second is downwards. If Knight has a choice then he will definitely choose the path where he doesn't need to waste his health. Thus, Knight will choose the cell with minimum health required.

In this paradigm, Knight will first find the most negative health at a particular cell, so knight only need to take $\text{absolute}(\text{most negative Health}) + 1$ as per the given constraints. Note that at any point of time, the most negative health becomes positive (Knight has enough health, so no need to take extra health), then he will consider that health as zero and continues the process of the algorithm.

Pseudo-code for recursive Approach

```

1) Procedure solve(matrix,i,j)
2) if i=last-row and j=last-col then
   ans=matrix[i][j];
3) else if i=last-row then
   Update ans as
   solve(matrix,i+1,j)+matrix[i][j];
4) else if j=last-col then
   Update ans as
   solve(matrix,i,j+1)+matrix[i][j];
5) else
   Update ans as
   max(solve(matrix,i+1,j),
   solve(matrix,i,j+1))+ matrix[i][j];
6) END if
7) If ans greater than or equal to 0
   then
   Update ans as 0
8) ENDIF
9) return ans;
10) END procedure
11) return abs(solve (matrix, 0,0))+1;

```

B. Dynamic Programming Bottom-Up Approach

The optimized version of recursive approach is this bottom up approach. In this approach we only need to put a memory part over recursive approach in order to avoid repetitive calculation. That's how the algorithm optimize the problem in a great extent. we will discuss the complexity analysis in the discussion section.

Pseudo-code for DP Bottom Up Approach

```

1) Procedure solve(matrix,i,j)
2) if dp[i][j]!=-1 then
   return dp[i][j];
3) END if
4) if i=last-row and j=last-col then
   ans=matrix[i][j];
5) else if i=last-row then
   Update ans as
   solve(matrix,i+1,j)+matrix[i][j];
6) else if j=last-col then
   Update ans as
   solve(matrix,i,j+1)+matrix[i][j];
7) else
   Update ans as
   max(solve(matrix,i+1,j),
   solve(matrix,i,j+1))+ matrix[i][j];

```

```

8) END if
9) If ans greater than or equal to 0
   then
   Update ans as 0
10)END if
11)return ans;
12) END procedure
13) return abs(solve (matrix, 0,0))+1;

```

C. Dynamic Programming Top-Down Approach

Top-Down approach is more space optimized in comparison of bottom up approach as here algorithm doesn't use extra space besides the number of rows x number of columns. We will analyze these things in the discussion section furthermore. The additional thing in this algorithm is that we have implemented dungeon master's code, which will randomly generate the dungeon matrix so that every time different traps and energy boost up has been added. The main part of this problem is that we have also made trace back of the dungeon path which will trace the optimal path along with the optimal health. The psuedocode for all three DP bottom up approach, Dungeon master and Optimal Path Trace-back has been added below.

Pseudo-code for DP Top Down Approach

```

1. Procedure dungeon_game(matrix)
   //matrix is in 2D vector form
2. Set m as row and n as column of
   matrix
3. Set vector dungeon as dungeon
   (m+1) row and (m+1) column and
   initiaize all the cell with minimum
   value
4. IF matrix[n-1][m-1] (last cell has
   no traps)
   then Update dungeon[n-1][m-1] as 0
   ELSE update dungeon[n-1][m-1] as
   matrix[n-1][m-1]
   ENDIF
5. FOR i=n-1 to 0
   FOR j=m-1 to 0
   IF last cell
   then continue;
   ENDIF
   Set Rightwards_Cost as
   matrix[i][j]+ dungeon[i][j+1]

```

```

Set Downwards_Cost as
    matrix[i][j]+dungeon[i+1][j]
IF Rightwards_Cost is positive or
    zero then
Update Rightwards_Cost as 0.
ENDIF
IF Downwards_Cost is positive or
    zero then
Update Downwards_Cost as 0.
ENDIF
Update dungeon[i][j]=
    max(rightwards,downwards)
ENDFOR
ENDFOR
6. Return abs(dungron[0][0] +1)

```

PseudoCode For TraceBack of the Optimal Path

```

1. Procedure optimal_path()
2. Set vector of pair min_path
3. Insert first cell to the min_path
   // As first cell is a starting cell
4. Set x as 0 and y as 0
5. WHILE x not reach to the end row
    or y not reach to the end column
Set right as dungeon[x][y+1] and left
as dungeon[x+1][y]
IF right is greater than equal to left
Set y as y+1
ELSE Set x as x+1
ENDIF
IF x is not in last row or y is not
in last column
THEN insert {x,y} to min_path
ENDIF
ENDWHILE
6. Return min_path

```

PseudoCode for Dungeon Master (Random Trap Generator)

```

1. Procedure dungeon_master()
2. Read Number Of Rows and Number Of
Columns and set as n and m
respectively
3. FOR i=0 to n
FOR j=0 to m
Set matrix[i][j] as
    rand()%(maximum_number +1
    -minimum_number))+minimum_number
// to set traps or energy
boost between the minimum and
maximum range
ENDFOR

```

```

ENDFOR
4. Return Matrix[i][j]

```

D. Binary Search Approach

In this approach, minimum Health points will be in range $[1, 1000 \cdot (rn, cn) + 1]$. Where, rn indicates row number and cn indicates column number. Using the given range we will find middle element of each interval and using middle element will decide whether to look for solution in right interval or left interval. Middle element might be an optimum solution if knight traverse dungeon with initial Health(which is middle element) successfully. Here, traverse successfully means with an positive value of health. Binary Search algorithm will be applied again and again and that will minimize initial health and return minimum health.

Pseudo Code for Binary search algorithm

```

1. Procedure calculateminHP(dungeon)
2. rn=numbers of row; cn=numbers of
column;
3. minHP=1, maxHP=1000*(rn+cn)+1;
4. while(minHP<=maxHP)
mid = (minHP+maxHP)/2
IF ( survival_check(dungeon,mid) )
then
ans=mid;
maxHP=mid-1;
ELSE
minHP=mid+1;
5.ENDIF;
ENDWHILE;
6.return ans;

```

Pseudo Code for survival check

```

1. Procedure survival_check(dungeon,
inithealth)
2. test[rn][cn]=zeros;
3. test[0][0]=inithealth+
dungeon[0][0];
4. for(int i=0; i<rn; ++i)
for(int j=0; j<cn; ++j)
IF(i>0 && test[i-1][j]>0) then
Update test[i][j] as
(test[i][j]
, test[i-1][j]+

```

```

        dungeon[i][j]);
    ENDIF
    IF(j>0 && test[i][j-1]>0)

        Update test[i][j] as
            (test[i][j]
            ,test[i][j-1]+
            dungeon[i][j]);
    ENDIF
ENDFOR
ENDFOR
5. return 1 if test[rn-1][cn-1]>0;

```

V. PLAN OF WORK

1) Goal 1 (Completed)

- **Objective:** Finding good quantity and quality of Research Paper/Reference Materials. Formulating the Problem Statement.
- **Time period:** 12 Days
- **Strategy:** We decided that each member will find good resources and then divide those resources amongst ourselves. Everyone shared their views, understanding of the Dungeon problem in the Google Meet. Then we have formulated the overall structure of the dungeon problem. The references we have used for this are mentioned in the Reference section.

1) Goal 2 (Completed)

- **Objective:** Understanding the real dungeon games, which are played by the Passionate gamer and relate it with our problem statement, which gives us a good insight in our problem domain.
- **Time period:** 5 Days
- **Strategy:** For understanding the concept of our problem, We have watched some YouTube videos and also referred some online resources.

1) Goal 3 (Completed)

- **Objective** Introduce about various dungeon games and the history about dungeon game and its popularity among young generation.
- **Time period:** 3 Days
- **Strategy:** We have divide some work among our group members and gather all the required information.

1) Goal 4 (Completed)

- **Objective:** Formulate pseudo-code and flow-chart of our problem.
- **Time period:** 1 Days
- **Strategy:** We have made flow-chart with the help of online editor.

1) Goal 5 (Completed)

- **Objective:** Implement the Brute force approach using recursion method and BFS. Optimize the solution of dungeon game with binary search and Dynamic programming approach.
- **Time period:** 2 Days
- **Strategy:** We will finalize our solution with above designated approaches. We can also define the time complexity equations for both recursive approach and dynamic programming approach of solution and compare both equations for Optimization analysis.

VI. RESULTS AND DISCUSSION

As we discussed our solution using various types of approaches which are recursive approach, DP bottom-up approach, DP top-down approach and binary search approach. From this above approaches' implementations and by observing pseudo code and flow chart, We observe that some approach takes more time and some approaches take less time to execute. In this section, we will know more about the time and space complexity analysis of all approaches very briefly and put all

A. Discussion on Complexity analysis of various algorithms

Time Complexity: Let's assume the number of row of the dungeon matrix is "m" and the column is "n". So we have a matrix of size m x n. In each cell, Knight has two paths to go from: One is rightwards cell and second is downwards cell. You can observe the situation with following recursive tree.



Space Complexity: Space Complexity of recursive approach depends upon the maximum depth of the recursion tree, which is approximately $O(m * n)$, where m is the number of row and n is the number of column.

Time Complexity: In the recursive approach, some function calls repeated again and again (Like Solve(mat,1,1)). This can make the

Space Complexity: In the algorithm, the storing dungeon matrix is size of $m_{max} \times n_{max}$. Where, m_{max} and n_{max} is given in the constraints which is maximum possible number of rows and columns. By analysing the algorithm, space complexity is $O(m_{max} \times n_{max})$.

Time Complexity: In the Dynamic Programming Top-down approach, we have only used two iterations of for loop, which runs $m*n$ times. That's why, we get time complexity of $O(m*n)$, where m is number of row and n is the number of column.

Space Complexity: The dungeon Matrix is of size $m \times n$. Thus, in this approach space complexity will be $O(m \times n)$ which is reduced in comparison to DP Bottom up approach where we have taken space of $O(m_{max} \times n_{max})$.

Time Complexity: We will search for optimum solution in the range $[1, 1000 \times (rn \times cn) + 1]$. Time complexity for binary search algorithm will be $O(\log_2(1000 \times (rn \times cn)))$ and for each search we will traverse $rn \times cn$ positions of matrix so total time complexity

of this algorithm will be $O(rn \times cn \times \log_2 Z)$ where $Z=1000 \times (rn \times cn)$. For obtaining optimal solution we traverse dungeon matrix of 'rn' rows and 'cn' columns.

Space Complexity: This algorithm has $O(rn \times cn)$ as space complexity. Here rn is row number and cn is column number.

B. Results

Now, we will discuss about the results that we got after running the above algorithms.

1) Initial Result for 3 x 3 dungeon matrix

```
Dungeon Matrix is shown below:
-2 -3 3
-5 -10 1
10 30 -5

Minimum Health Required to rescue the princess = 7
PS C:\Users\91997\Desktop\C++Debug\.vscode>
```

Fig. 4. Minimum Health

The above result has the value of minimum required health to reach at the destination to rescue princess from demons. In this case, we have taken 3 x 3 matrix for only the understanding purpose. We have implemented the dungeon master for random dungeon matrix generator in DP Top down approach in addition to minimum required health and optimal path.

2) User entered rows=3 and column = 3

```
PS C:\Users\91997\Desktop\C++Debug\.vscode> cd "C:\Users\91997\Desktop\C++Debug\.vscode" & if ($?) { g++ DungeonGame.cpp -o DungeonGame 2>&| (
) & .\DungeonGame }
Enter Number of rows and columns:
3 3
Dungeon matrix set by the Dungeon Master is as follows:
999 542 669
513 858 717
473 344 -51
min health required to rescue Princess= 1473
King should go for this path to save princess:
(0,0) -> (1,0) -> (1,1) -> (1,2) -> (2,2)
PS C:\Users\91997\Desktop\C++Debug\.vscode>
```

Fig. 5. Minimum health required is 1473 with optimal Path for number of rows=3 and column=3

3) User entered rows=9 and column = 9

```
Enter Number of rows and columns:
9
9
Dungeon matrix set by the Dungeon Master is as follows:
999 542 669 513 858 717 473 344 51
548 703 869 278 181 957 509 4 957
175 434 425 483 981 847 708 624 413
253 789 886 445 718 276 533 869 983
655 714 27 890 311 808 307 482 665
138 134 768 761 115 535 431 154 299
973 147 281 717 516 222 690 34 421
842 712 989 36 42 255 363 433 794
883 278 642 343 1 86 459 547 620
min health required to rescue Princess= 1503
King should go for this path to save princess:
(0,0) -> (0,1) -> (1,1) -> (2,1) -> (3,1) -> (3,2) -> (3,3) -> (3,4) -> (3,5) -> (3,6) -> (3,7) -> (3,8) -> (4,8) -> (5,8) -> (6,8) -> (7,8) -> (8,8)
PS C:\Users\91997\Desktop\C++Debug\.vscode>
```

Fig. 6. Minimum health required is 1503 with optimal Path for number of rows=9 and column=9

The above results describes the minimum health require to reach at the destination cell along with the optimal path that should be chosen by the knight to reach from the starting cell to destination cell. As you can observe that the dungeon matrix is generated with respect to given constraints. In this case dungeon master acts as dungeon matrix generator. It allocates all the traps and health boosts. After the result of optimal health, we got the result of optimal path for Knight to follow to reach at the destination with minimum initial health. The decision of path itself is taken by the DP Top - Down algorithm.

VII. CONCLUSION

This document briefly discussed all the aspects of dungeon games. The history of the dungeon, the popularity of the dungeon, defined the problem

statement, pseudo-code and its flowchart for the recursive solution, Dynamic Programming paradigm (Top Down and Bottom Up Approach) and Binary Search approach. We have solved the dungeon game problem with brute force approach to the most optimal approach in terms of both time and space complexity.

We can categorize the Dungeon Game problem with an optimal path searching problem, similar to the min Cost path problem. The first observation any reader can make is to relate the given problem's solution with Dijkstra's Algorithm, which is used to find the shortest path among all the possible routes. That is used in geographical maps. To find locations of a map which refers to vertices of a graph. The variation of the algorithm can also be used to find the shortest Path in IP routing. The optimal path algorithms are also used in telephone networks.

VIII. ACKNOWLEDGMENT

We are thankful to Prof. Manish Kumar for providing us the opportunity to research on such topics and have the experience of applying what we study to real-life scenarios. Also thankful to Teaching Assistants for guiding us to carry out our research in the correct direction.

REFERENCES

- [1] [https://en.m.wikipedia.org/wiki/Dungeon_\(video_game\)](https://en.m.wikipedia.org/wiki/Dungeon_(video_game))
- [2] <https://en.m.wikipedia.org/wiki/Dungeon>
- [3] <https://www.google.com/url?sa=t&source=web&rct=j&url=https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/amp/&ved=2ahUKEwlpO79pNvzAhXDzDgGH8wBSgQFnoECAUQA&usg=AOvVaw1eN3uZ7xFUAW7QvyJ1c4Ur&cf=1>
- [4] https://www.google.com/url?sa=t&source=web&rct=j&url=https://en.m.wikipedia.org/wiki/Breadth-first_search&ved=2ahUKEwi21OqOpdvzAhWjrJUCHe2BCxsQFnoECAUQA&sqi=2&usg=AOvVaw2cDGQoqRZRKkG9CeJ-AE1E
- [5] <https://leetcode.com/problems/dungeon-game/>
- [6] <https://www.geeksforgeeks.org/minimum-positive-points-to-reach-destination/>
- [7] https://www.google.com/url?sa=t&source=web&rct=j&url=https://www.geeksforgeeks.org/recursion/amp/&ved=2ahUKEwi8hqGlpdvzAhWRzDgGHUnSDokQFnoECAUQA&usg=AOvVaw0Plx-BULspaDpg_UGZ9MUc&cf=1
- [8] <https://www.geeksforgeeks.org/dynamic-programming/>
- [9] https://www.google.com/url?sa=t&source=web&rct=j&url=https://en.m.wikipedia.org/wiki/Dynamic_programming&ved=2ahUKEwjM_favpdvzAhX5yzgGHUbsDPAQFnoECC8QAQ&usg=AOvVaw3FIPockElaGgY7kCWgwCt9
- [10] <https://medium.com/@vpandeliev/stage-1-problem-statement-competitor-analysis-549107869a35>
- [11] <https://www.cyberdefinitions.com/definitions/DUNGEON.html#:~:text=DUNGEON%20means%20Closed%2DOff%20Area,castles%2C%20fortresses%2C%20or%20caves.>
- [12] <https://www.merriam-webster.com/dictionary/dungeon#h1>
- [13] <https://www.merriam-webster.com/dictionary/donjon>
- [14] <https://mykindofmeeple.com/what-is-dungeons-and-dragons/>
- [15] <https://dnd.wizards.com/rules-introduction>
- [16] <https://leetcode.com/problems/dungeon-game/>
- [17] <https://en.wikipedia.org/wiki/Dungeon>
- [18] <https://critical-hits.com/blog/2008/05/09/review-the-classic-dungeon/>
- [19] <https://gamerant.com/dungeons-and-dragons-infographic-2021/>
- [20] <https://app.diagrams.net/>
- [21] <https://youtu.be/4uUGxZXoR5o>

IX. APPENDIX

Codes for all approaches

A. Recursive Approach

```
/*
    Author :- Group 26
    Project :- Dungeon Game
*/
#include <iostream>
#include <bits/stdc++.h>
#include<unordered_set>
#include<unordered_map>
#include<vector>
#define fast
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
#define ll long long int
#define maxn 1000
const int maximum_number=1000,
minimum_number=-1000;
using namespace std;
int dungeon(vector<vector<int>>
    &mat,int i,int j)
{
    int ans;
    if(i==mat.size()-1 &&
        j==mat[0].size()-1) //
        boundary levels
    {
        ans=mat[i][j];
    }
    else if(j==mat[0].size()-1)
    {
        ans=dungeon(mat,i+1,j)
        +mat[i][j];
    }
    else if(i==mat.size()-1)
    {
        ans=dungeon(mat,i,j+1)
        +mat[i][j];
    }
    else{
        ans=max(dungeon(mat,i,j+1),
            dungeon(mat,i+1,j))+mat[i][j]; //
            either move rightwards or
            downwards
    }
    ans=ans>=0 ? 0:ans; //if
        ans(needed health at some
        instance) is already
```

```
        positive then no extra
        health needed
    return ans;
}

int main()
{
    int n=3,m=3;
    vector<vector<int>> matrix={
        {-2,-3,3 },
        {-5,-10,1},
        {10,30,-5}
    };
    cout<<"Entered Matrix is shown
        below:\n\n";
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<m;j++)
        {
            cout<<matrix[i][j]<<" ";
        }
        cout<<"\n";
    }

    // We need to give absolute of
    (returned answer) + 1 health to
    rescue princess
    cout<<"\nMinimum Health Required
        to rescue the princess =
        "<<abs(dungeon(matrix,0,0))+1;

    return 0;
}
```

B. DP Bottom-Up Approach

```
/*
    Author :- Group 26
    Project :- Dungeon Game
*/
#include <iostream>
#include <bits/stdc++.h>
#include<unordered_set>
#include<unordered_map>
#include<vector>
#define fast
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
#define ll long long int
#define maxn 1000
// Constraint on energy boost level
    and Traps are defined here.
```

```

const int maximum_number=1000,
minimum_number=-1000;

using namespace std;

int dp[201][201]; //dp table to
memorize the repetitive recursive
calls
int dungeon(vector<vector<int>>
&mat,int i,int j)
{
    if(dp[i][j]!=-1) // if we
        already encounter the
        subproblem then return it
        directly
    {
        return dp[i][j];
    }
    int ans;
    if(i==mat.size()-1 &&
j==mat[0].size()-1) //
        boundary levels
    {
        ans=mat[i][j];
    }
    else if(j==mat[0].size()-1)
    {
        ans=dungeon(mat,i+1,j)
        +mat[i][j];
    }
    else if(i==mat.size()-1)
    {
        ans=dungeon(mat,i,j+1)
        +mat[i][j];
    }
    else{
        ans=max(dungeon(mat,i,j+1),
dungeon(mat,i+1,j))+mat[i][j];
        // either move rightwards
        or downwards
    }
    ans=ans>=0 ? 0:ans; //if
        ans(needed health at some
        instance) is already
        positive then no extra
        health needed
    dp[i][j]=ans;
    return ans;
}

int main()

```

```

{
    memset(dp,-1,sizeof(dp));
    int n=3,m=3;
    vector<vector<int>> matrix={
        {-2,-3,3 },
        {-5,-10,1},
        {10,30,-5}
    };
    cout<<"Entered Matrix is shown
        below:\n\n";
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<m;j++)
        {
            cout<<matrix[i][j]<<" ";
        }
        cout<<"\n";
    }

    // We need to give absolute of
    (returned answer) + 1 health to
    rescue princess
    cout<<"\nMinimum Health Required
        to rescue the princess =
        "<<abs(dungeon(matrix,0,0))+1;

    return 0;
}

```

C. DP Top-Down Approach

```

/*
    Author :- Group 26
    Project :- Dungeon Game
    Program: Minimum required
        Health and optimal Path
        to rescue Princess
    Approach:Dynamic
        Programming Approach
        [Top - Down DP]
    Time Complexity: O(M*N)
    Space Complexity: O(M*N)
*/

#include <iostream>
#include <bits/stdc++.h>
#include<unordered_set>
#include<unordered_map>
#include<vector>
#define fast
    ios_base::sync_with_stdio(false);
cin.tie(NULL);
#define ll long long int
#define maxn 1000

```

```

using namespace std;
const int maximum_number=1000,
minimum_number=-1000;
void dungeon_game(vector<vector<int>>
&matrix)
{
int n=matrix.size();
int m=matrix[0].size();
vector<vector<int>>
dungeon(n+1,vector<int>
(m+1,INT_MIN+maxn)); // 2D vector
of size (n+1)*(m+1) initialize
with minimum value

```

```

dungeon[n-1][m-1]=matrix[n-1][m-1]>=0?
0:matrix[n-1][m-1]; //if princess
cell (last cell) has no traps then

```

```

for(int i=n-1;i>=0;i--)
{
for(int j=m-1;j>=0;j--)
{
if(i==n-1 && j==m-1) continue;
int rightwards_cost=matrix[i][j]
+dungeon[i][j+1];
int downwards_cost=matrix[i][j]
+dungeon[i+1][j];
//if cost (rightwards or downwards)
are already positive then
//no extra health needed at this
particular instance
rightwards_cost=rightwards_cost>=0
? 0:rightwards_cost;
downwards_cost=downwards_cost>=0
? 0: downwards_cost;
dungeon[i][j]=max(rightwards_cost,
downwards_cost);
}
}

```

```

cout<<"min health required to rescue
Princess=
"<<abs(dungeon[0][0])+1<<"\n\n";
// Trace Back the optimal path
vector <pair<int,int>> min_path;

```

```

min_path.push_back({0,0}); //
starting index (cell)

```

```

int x=0,y=0;
while(!(x==n-1 && y==m-1))
{
int right=dungeon[x][y+1];
int left=dungeon[x+1][y];
if(right>=left)
{
x=x;
y=y+1;
}
else
{
x=x+1;
y=y;
}
if(!(x==n-1 && y==m-1))
//min_path.push_back({x,y}); //
extra insertion of optimal cells
healthoptimal path
needed
for
particular
instance
cout<<"King should go for this path
to save princess: \n\n";

```

```

for(auto path_to_min: min_path)
{
cout<<"("<<path_to_min.first<<","<
<<path_to_min.second<<")"<<" -> ";
}
cout<<"("<<n-1<<","<<m-1<<")";
}

```

```

int main()
{
int n,m;
cout<<"Enter Number of rows and
columns: \n";
cin>>n>>m;

```

```

vector <vector<int>>
matrix(n,vector<int> (m));
for(int i=0;i<n;i++)
{
for(int j=0;j<m;j++)
{
matrix[i][j]=(rand()% (maximum_number
+ 1 - minimum_number)) +

```

```

        minimum_number;
    }
}
cout<<"\nDungeon matrix set by the
Dungeon Master is as follows:\n\n";
for(int i=0;i<n;i++)
{
    for(int j=0;j<m;j++)
    {
        cout<<matrix[i][j]<<" ";
    }
    cout<<"\n";
}
cout<<"\n";
dungeon_game(matrix);

return 0;
}

```

D. Binary search Approach

```

#include <bits/stdc++.h>
#include<vector>
#include<iostream>
#include<cmath>
using namespace std;

/*
Author :- Group 26
Project :- Dungeon Game
Objective :- Minimize initial health
to enter the dungeon
--> This code will calculate
minimum health required for
knight to enter a dungeon.
--> calculateMinimumHP will accept
dungeon as parameter and
calculate minimum health using
binary search algorithm.
--> minimum HP will be in range
[1,1000*(rn,cn)+1].
--> rn indicates row number and cn
indicates column number.
--> acceptable is implemented
using dynamic programming which
will return 1 if knight
traverse dungeon with
initHealth successfully.
--> initHealth indicates initial
health.
--> calculateMinimumHP will
minimize initial health and
return minimum health.

```

```

--> time complexity for this
algorithm is of order
O(rn*cn*Z), where
Z=1000*(rn,cn)+1

int rn,cn;
int
calculateMinimumHP(vector<vector<int>>&
dungeon);
bool acceptable(vector<vector<int>>&
dungeon, int initHealth);

int main()
{
vector<vector<int>> matrix={
    {-2,-3,3 },
    {-5,-10,1},
    {10,30,-5}
};
cout<<"Entered Matrix is shown
below:\n\n";
for(int i=0;i<3;i++)
{
    for(int j=0;j<3;j++)
    {
        cout<<matrix[i][j]<<" ";
    }
    cout<<"\n";
}
// function call below will use
inary search algorithm to
calculate minimum HP to enter the
dungeon
cout<<"\nMinimum Health Required
to rescue the princess =
"<<calculateMinimumHP(matrix);

return 0;
}

int
calculateMinimumHP(vector<vector<int>>&
dungeon)
{
rn=dungeon.size();
//find row number
cn=dungeon[0].size();
//find column number
int minh=1, maxh=1000*(rn+cn)+1,
ans=maxh, mid; //if all
elements are positive then,
ans=1 else worst case
scenario(all elements are

```

```

-1000) ans=1000*(rn+cn)+1
while(minh<=maxh)
    //Binary Search Algorithm
{
    mid=(minh+maxh)/2;
    if(acceptable(dungeon,mid))
        // is true if knight is able
        // to survive with *mid health
        // and will look for min health
        // in left interval of
        [1-mid-1000*(rn,cn)+1]
    {
        ans=mid;
        maxh=mid-1;
    }
    else
        // look for min health in
        // right interval
        [1-mid-1000*(rn,cn)+1]
    {
        minh=mid+1;
    }
}
return ans;
}

```

```

bool acceptable(vector<vector<int>>&
dungeon, int initHealth)
{
    vector<vector<int>> test(rn,
        vector<int>(cn, 0)); // vector
        with all initial value 0
    test[0][0]= initHealth +
        dungeon[0][0]; // enters in
        the first cell of dungeon
    for(int i=0; i<rn; ++i)
    {
        for(int j=0; j<cn; ++j)
        {
            if(i>0 && test[i-1][j]>0)
                // if knight
                advances in
                row(horizontal)
            {
                test[i][j]=max(test[i][j],
                    test[i-1][j]+dungeon[i][j]);
            }
            if(j>0 && test[i][j-1]>0)
                // if knight
                advances in
                column(vertical)
            {
                test[i][j]=max(test[i][j],
                    test[i][j-1]+dungeon[i][j]);
            }
        }
    }
}

```

```

    }
}
return test[rn-1][cn-1]>0;
        // if knight
        survives with initial health
        then BINGO(1)
}

```

Link to Access the code files

<https://drive.google.com/drive/folders/1f7SzM7CMNBklM416wZu33tgdCVtR0sB9?usp=sharing>