

**ASSUMPTION UNIVERSITY**

Vincent Mary School of Science and Technology

**CSX3009 ALGORITHM DESIGN**

**TERM PROJECT REPORT**

COOKIE GALORE

SUBMIT TO

Asst. Prof. Dr. Thitipong Tanprasert

BY

Kaung Khant Si Thu 6228104

Piyaphon Wu 6311020

SEMESTER 1/2022

Table of Contents

[Problem statement 3](#_Toc115288330)

[The Approach to the problem 4](#_Toc115288331)

[Remarks 6](#_Toc115288332)

[Time complexity 6](#_Toc115288333)

[Submission Status 7](#_Toc115288334)

[References 7](#_Toc115288335)

[Code Snippet 1: The function checking the valid move 4](#_Toc115279972)

[Code Snippet 2: Initializing the arrays 5](#_Toc115279973)

[Code Snippet 3: Searching the path to the goal that Santa needs to eat the minimum number of cookies 6](#_Toc115279974)

# Problem statement

Scenario

Santa wants to find the minimum number of cookies he must eat to travel from top-left to bottom-right corner and he can only travel to adjacent cells around him without diagonal moves. The cells have either an open space, denoted by “.”, or cookie denoted by C. Whenever Santa enters the cell with cookie, he always eats it. Since he wants to keep his body slim, we must find the minimum number of cookies he must eat to travel from (1, 1) to (N, M).

Input

The first line will contain N (1 ≤ N ≤ 1000) and M (1 ≤ M ≤ 1000), the number of rows and columns in the grid. The N lines will each contain M cells.

Output

The minimum number of cookies collected while travelling from (1, 1) to (N, M).

Sample

|  |  |
| --- | --- |
| Input | Output |
| 6 4  C..C  .CCC  CC.C  ..CC  CC..  C.C. | 3 |

# The Approach to the problem

There are 2 main constraints that we need to consider before jumping into the approach to problem.

Adjacent moves

Since Santa can only move to adjacent cells with no diagonal, his move can be [up, right, down, and left] which can be written in matrix form as [[-1, 0], [0, 1], [1, 0], [0, -1]].

Valid moves

Because Santa must move within the N x M grid, we will not consider the path where he moves outside the grid. If he moves outside the grid, then we just skip it. Also, if the path has been visited, we do not want him to revisit again.

def isSafe(r, c):  
 global visited  
 return 0 <= r < N and 0 <= c < M and not visited[r][c]

Code Snippet 1: The function checking the valid move

Firstly, we create 3 matrices, grid matrix to contain the input, visited matrix to avoid repeated paths, and dynamic table matrix to save the minimum cookies taken at the paths.

Grid matrix – The cells in the grid “.” and C are replaced with 0 and 1 respectively to easily calculate the number of cookies and put them in the dynamic table matrix.

Visited matrix – A matrix of size containing False values when initiated. Then set the first row and column of the matrix to be True because Santa must start from that cell.

Dynamic table matrix – A matrix of consisting of infinity values when initiated. Then update the first cell’s value with the first cell of the grid matrix since it can be either cookie (1) or blank (0).

import collections

N, M = map(int, input().split())  
  
grid = []  
  
moves = [[-1, 0], [0, 1], [1, 0], [0, -1]]  
  
for \_ in range(N):  
 grid.append([1 if x == "C" else 0 for x in input()])  
  
visited = [[False] \* M for \_ in range(N)]  
visited[0][0] = True

dp = [[float('inf')] \* M for \_ in range(N)]  
dp[0][0] = grid[0][0]  
  
queue = collections.deque()  
queue.append(State(0, 0, grid[0][0]))

Code Snippet 2: Initializing the arrays

After that, create a deque (Doubly ended queue) and append the first state which is the first cell along with its row-column indexes and value (cookie countuu8u). This collection module is chosen for creating a queue with an optimized list to make insertion and popping on both ends easily on the time complexity of O(1) while regular list would be O(n).

while queue:  
 s = queue.popleft()  
 row, column, value = s.row, s.column, s.value  
 if value == dp[row][column]:  
 for nRow, nColumn in moves:  
 nRow += row  
 nColumn += column  
 if isSafe(nRow, nColumn) and value + grid[nRow][nColumn] < dp[nRow][nColumn]:  
 visited[nRow][nColumn] = True  
 dp[nRow][nColumn] = value + grid[nRow][nColumn]  
 if grid[nRow][nColumn] == 1:  
 queue.append(State(nRow, nColumn, dp[nRow][nColumn]))  
 else:  
 queue.appendleft(State(nRow, nColumn, dp[nRow][nColumn]))  
  
print(dp[-1][-1])

Code Snippet 3: Searching the path to the goal that Santa needs to eat the minimum number of cookies

We then use while loop for doing BFS technique to check the paths by picking the first state in the queue. Then check if the number of cookies of the current state is equal to the value in dynamic table to continue from the path that costs the same number of cookies. If the condition is true, then use for loop to get the adjacent cell and check if it is a valid move or not. If it is a valid move, then mark the cell as visited, and set the state’s current value plus the total number of cookies eaten to get at the current index as the cell of dynamic table matrix. After that, if the current cell inside the grid does not have a cookie, we will put it at the beginning of the queue as it is necessary to keep the number of cookies minimum and if there is a cookie, we will put the state of adjacent cell and its number of cookies to the last of the queue.

# Remarks

The original algorithm used heapq data structure to append and pop the State object which took 75.2 s (63.6 MB) to finish the biggest testcase in the DMOJ website. But after reviewing the code with faster runtime that the user named [Spitfire720](https://dmoj.ca/user/Spitfire720) submitted in which they used simpler data structure called deque to manage the current state of Santa, we refactored our code to use the same data structure and the runtime has reduced to 49.19 s (63.5 MB) which is 36.024 % speedup.

Time complexity

# Submission Status

Table

Description automatically generated

# References

Jakob, K. et al. (2022, June 8). *0-1 BFS*. Retrieved from Algorithms for Competitive Programming: https://cp-algorithms.com/graph/01\_bfs.html

Maxwell, C. (2022). *SAC '22 Code Challenge 2 P4 - Cookie Galore*. Retrieved from DM::OJ: https://dmoj.ca/problem/sac22cc2p4/editorial

Waseem, M. (2021, July 15). *Collections In Python : Everything You Need To Know About Python Collections*. Retrieved from Edureka: https://www.edureka.co/blog/collections-in-python/

Reducible. (2020, September 26). *Breadth First Search (BFS): Visualized and Explained*. Retrieved from Youtube: https://www.youtube.com/watch?v=xlVX7dXLS64