Tecnológico de Monterrey

Tecnológico de Monterrey - Campus Monterrey
School of Engineering and Sciences
Engineering in Computational Technologies
Analysis and Design of Advanced Algorithms

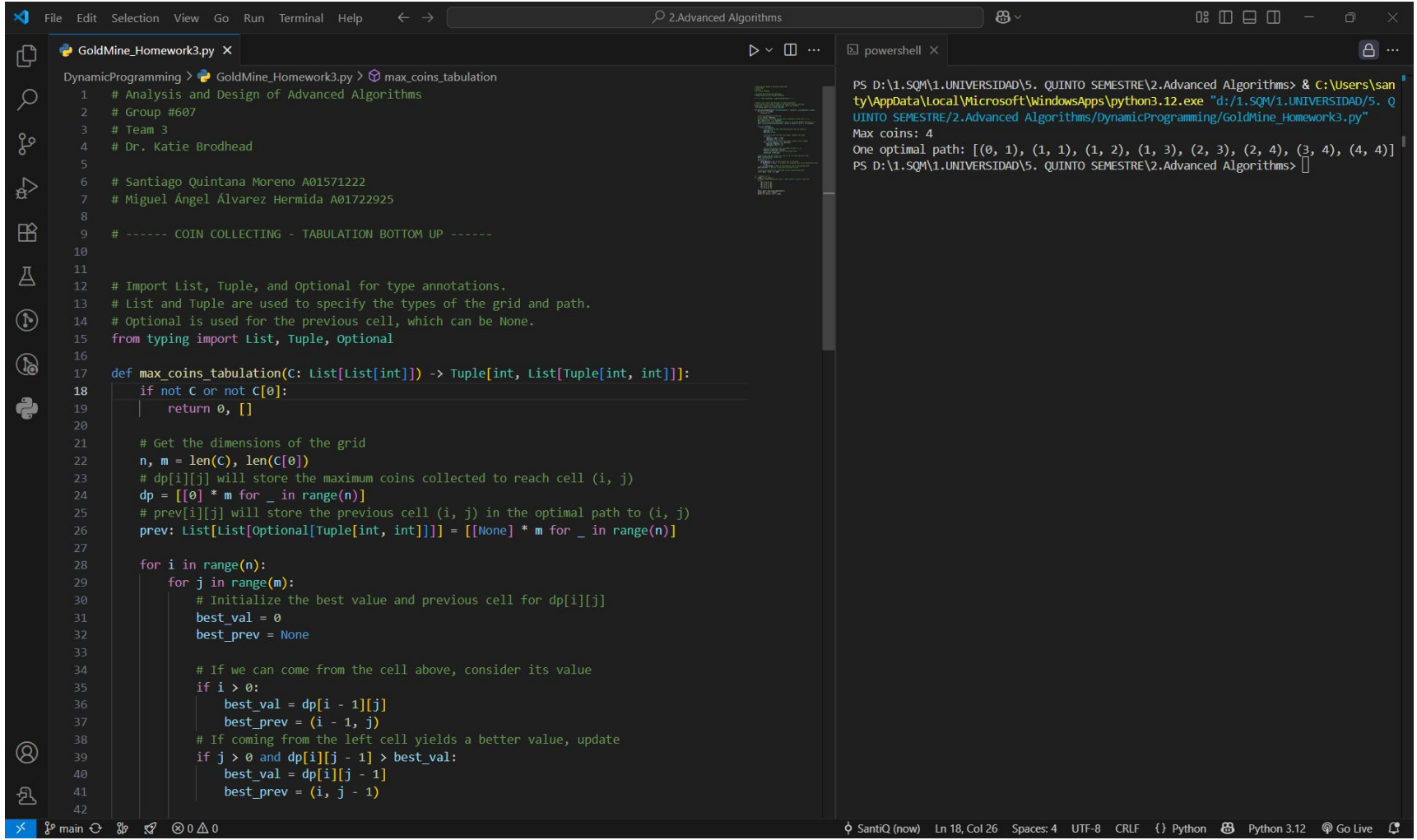
Class Activity 3:
Coin Change through Memoization or Tabulation

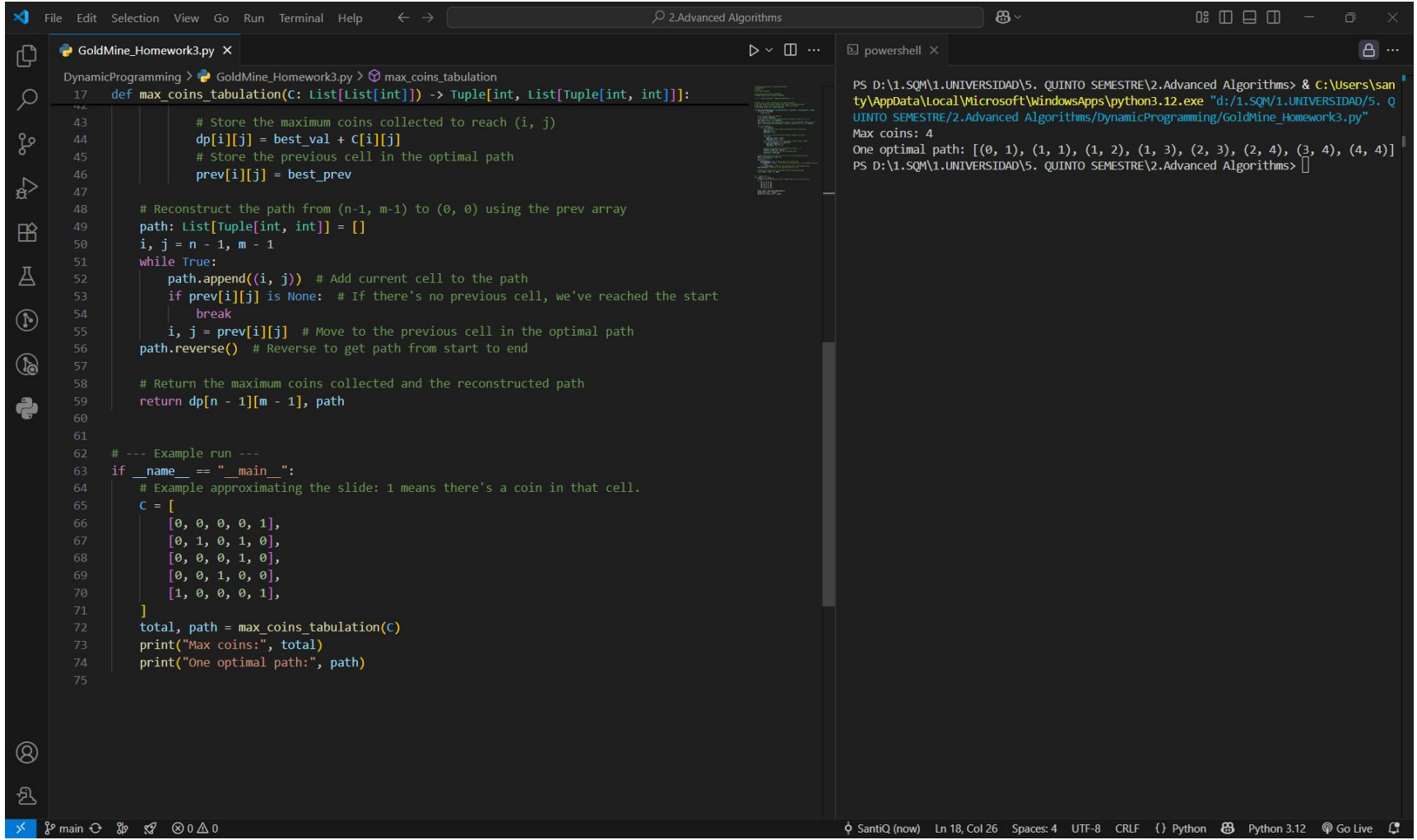
Group: 607
Team #3
Dr. Katie Brodhead

Santiago Quintana Moreno A01571222
Miguel Ángel Álvarez Hermida a01722925

Gold Mine - Tabulation

https://colab.research.google.com/drive/1kRMVjIz5v4-58qJouW5aF5BnfX8_xgFW?usp=sharing


File Edit Selection View Go Run Terminal Help
2.Advanced Algorithms
GoldMine_Homework3.py
DynamicProgramming > GoldMine_Homework3.py > max_coins_tabulation
# Analysis and Design of Advanced Algorithms
# Group #607
# Team 3
# Dr. Katie Brodhead
# Santiago Quintana Moreno A01571222
# Miguel Ángel Álvarez Hermida A01722925
# ------ COIN COLLECTING - TABULATION BOTTOM UP ------
# Import List, Tuple, and Optional for type annotations.
# List and Tuple are used to specify the types of the grid and path.
# Optional is used for the previous cell, which can be None.
from typing import List, Tuple, Optional
def max_coins_tabulation(C: List[List[int]]) -> Tuple[int, List[Tuple[int, int]]]:
if not C or not C[0]:
return 0, []
Get the dimensions of the grid
n, m = len(C), len(C[0])
# dp[i][j] will store the maximum coins collected to reach cell (i, j)
dp = [[0] * m for _ in range(n)]
# prev[i][j] will store the previous cell (i, j) in the optimal path to (i, j)
prev: List[List[Optional[Tuple[int, int]]]] = [[None] * m for _ in range(n)]
for i in range(n):
for j in range(m):
# Initialize the best value and previous cell for dp[i][j]
best_val = 0
best_prev = None
# If we can come from the cell above, consider its value
if i > 0:
best_val = dp[i - 1][j]
best_prev = (i - 1, j)
# If coming from the left cell yields a better value, update
if j > 0 and dp[i][j - 1] > best_val:
best_val = dp[i][j - 1]
best_prev = (i, j - 1)
powershell
PS D:\1.SQM\1.UNIVERSIDAD\5. QUINTO SEMESTRE\2.Advanced Algorithms> & C:\Users\santy\AppData\Local\Microsoft\WindowsApps\python3.12.exe "d:/1.SQM/1.UNIVERSIDAD/5. QUINTO SEMESTRE/2.Advanced Algorithms/DynamicProgramming/GoldMine_Homework3.py"
Max coins: 4
One optimal path: [(0, 1), (1, 1), (1, 2), (1, 3), (2, 3), (2, 4), (3, 4), (4, 4)]
PS D:\1.SQM\1.UNIVERSIDAD\5. QUINTO SEMESTRE\2.Advanced Algorithms>
main
SantiQ (now)
Ln 18, Col 26
Spaces: 4
UTF-8
CRLF
Python
Python 3.12
Go Live


File Edit Selection View Go Run Terminal Help
2.Advanced Algorithms
GoldMine_Homework3.py
DynamicProgramming > GoldMine_Homework3.py > max_coins_tabulation
def max_coins_tabulation(C: List[List[int]]) -> Tuple[int, List[Tuple[int, int]]]:
# Store the maximum coins collected to reach (i, j)
dp[i][j] = best_val + C[i][j]
# Store the previous cell in the optimal path
prev[i][j] = best_prev
# Reconstruct the path from (n-1, m-1) to (0, 0) using the prev array
path: List[Tuple[int, int]] = []
i, j = n - 1, m - 1
while True:
path.append((i, j)) # Add current cell to the path
if prev[i][j] is None: # If there's no previous cell, we've reached the start
break
i, j = prev[i][j] # Move to the previous cell in the optimal path
path.reverse() # Reverse to get path from start to end
Return the maximum coins collected and the reconstructed path
return dp[n - 1][m - 1], path
# --- Example run ---
if __name__ == "__main__":
# Example approximating the slide: 1 means there's a coin in that cell.
C = [
[0, 0, 0, 0, 1],
[0, 1, 0, 1, 0],
[0, 0, 0, 1, 0],
[0, 0, 1, 0, 0],
[1, 0, 0, 0, 1],
]
total, path = max_coins_tabulation(C)
print("Max coins:", total)
print("One optimal path:", path)
powershell
PS D:\1.SQM\1.UNIVERSIDAD\5. QUINTO SEMESTRE\2.Advanced Algorithms> & C:\Users\santy\AppData\Local\Microsoft\WindowsApps\python3.12.exe "d:/1.SQM/1.UNIVERSIDAD/5. QUINTO SEMESTRE/2.Advanced Algorithms/DynamicProgramming/GoldMine_Homework3.py"
Max coins: 4
One optimal path: [(0, 1), (1, 1), (1, 2), (1, 3), (2, 3), (2, 4), (3, 4), (4, 4)]
PS D:\1.SQM\1.UNIVERSIDAD\5. QUINTO SEMESTRE\2.Advanced Algorithms>
main
SantiQ (now) Ln 18, Col 26 Spaces: 4 UTF-8 CRLF Python Python 3.12 Go Live

Why tabulation (Bottom Up) solution?

In Python, this problem is better bottom-up through tabulation for this grid because it avoids recursion overhead and recursion-depth limits on large inputs. Tabulation has the same O(nm) time complexity but uses predictable loops that are more cache-friendly and typically faster in practice. It also makes extras like path reconstruction easy with a predecessor table, and if you only need the value, you can shrink space to O(m) with a sweep. Since memoization offers no asymptotic gain and adds call overhead plus stack risk, tabulation is the safer, simpler choice.

REFERENCES

GeeksforGeeks. (2025, July 23). *Tabulation vs Memoization*. GeeksforGeeks. https://www.geeksforgeeks.org/dsa/tabulation-vs-memoization/

GeeksforGeeks. (2025, July 25). *Dynamic Programming or DP*. GeeksforGeeks. https://www.geeksforgeeks.org/competitive-programming/dynamic-programming/

GeeksforGeeks. (2025, July 8). *Gold Mine Problem*. GeeksforGeeks. https://www.geeksforgeeks.org/dsa/gold-mine-problem/

Nasika, S. (2025, June 30). *Gold Mine Problem (Visualization and code examples)*. Final Round AI. https://www.finalroundai.com/articles/gold-mine-problem