3D KNAPSACK PROBLEM SOLVER DOCUMENTATION

OVERVIEW

A C++ program that solves the multi-dimensional knapsack problem with CPU, RAM, and Disk constraints using dynamic programming.

WHAT IT DOES

Given a set of tasks (each with CPU, RAM, disk requirements and a value) and resource limits, this program finds the optimal combination of tasks that:

- Maximizes total value
- Stays within all resource constraints

INPUT FORMAT

The program reads from 'input.txt':

Line 1: n (number of tasks)
Line 2: maxCPU maxRAM maxDisk (resource limits)
Next n lines: cpu ram disk value (for each task)

Example:
3
10 8 5
3 2 1 50
5 4 2 80
2 3 3 40

OUTPUT FORMAT

The program writes to 'output.txt':

Maximum Total Value: [number]
Selected Tasks (0-based indices): [task indices]
Used Resources: CPU=[used] RAM=[used] DISK=[used]
Remaining Resources: CPU=[remaining] RAM=[remaining] DISK=[remaining]

HOW IT WORKS

1. Dynamic Programming: Uses a 4D DP table dp[task][cpu][ram][disk] to store optimal values
2. Optimization: For each task, decides whether to include it or not based on maximum value
3. Backtracking: Traces back through the DP table to find which tasks were selected

KEY COMPONENTS

Task Structure:

- cpu: CPU requirement
- ram: RAM requirement
- disk: Disk requirement
- value: Task value

Main Function:

- knapsack3D(): Core algorithm that fills DP table and finds optimal solution
- Returns maximum achievable value and populates selected tasks list

COMPLEXITY

Time Complexity: O(n × maxCPU × maxRAM × maxDisk)
Space Complexity: O(n × maxCPU × maxRAM × maxDisk)

USAGE

1. Create 'input.txt' with your problem data
2. Compile: g++ -o knapsack knapsack.cpp
3. Run: ./knapsack
4. Check results in 'output.txt'