An-Najah National University

Faculty of Engineering

Department of Computer Engineering

Algorithms and Computational Complexity (10636314)

Dynamic Programming Project report

Gandalf the Wizard

Prepared by:

Wala’ Essam Ashqar, 12027854

Submitted by:

Dr. Samer Arandi

Second Smester

June 2024

- ## Introduction

In the mystical realm of Middle-earth, Gandalf must determine the minimum possible value of a chest filled with treasures without opening it. Each treasure has a weight and value which are Known. The target is calculating the minimum value for a given chest weight using two approaches: divide and conquer and dynamic programming.

Like the traditional knapsack problem, this one aims to optimize the overall worth of the objects packed into a backpack while staying under the weight restriction. However, in this case, the objective is to decrease the chest's total worth while keeping the weight restriction in mind.

- ## Divide & Conquer approach (D&C)

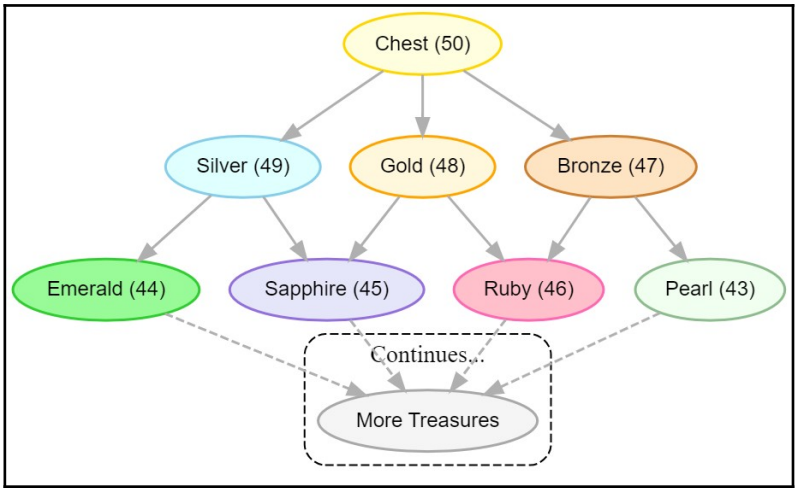
1. Define a value returned by the function that we will optimize.

The function (result) returns the minimum possible value of a chest with weight a using the given types of gems and metals.

2. Define the Parameters Which the function Depends On

- (a) is the weight of the chest.
- (ms) is list of pairs, where each pair contains the weight and value of a type of treasure.
- (arry) is an array to store computed results.

3. Draw the Recursion Tree for f Using the Values from the Example Above



4. The Recursive Code for Divide and Conquer approach.

```cpp
#include <iostream>
#include <vector>
#include <unordered_map>
#include <climits>
using namespace std;

int result(int a, vector<pair<int, int>>& ms, unordered_map<int, int>& arry) {
    if (a == 0) return 0;
    if (arry.find(a) != arry.end()) return arry[a];
    int minVal = INT_MAX;
    for (const auto& m : ms) {
        int matW = m.first;
        int matV = m.second;
        if (a >= matW) {
            int res = result(a - matW, ms, arry);
            if (res != INT_MAX) {
                minVal = min(minVal, res + matV);}}}
    arry[a] = minVal;
    return minVal;}

int main() {
    int allA, numM;
    cin >> allA >> numM;
    vector<pair<int, int>> ms(numM);
    for (int i = 0; i < numM; ++i) {
        int w, val;
        cin >> w >> val;
        ms[i] = {w, val};}
    unordered_map<int, int> arry;
    int finalres = result(allA, ms, arry);
    if (finalres == INT_MAX) {
        finalres = 0;}
    cout << finalres << endl;
    return 0;}
```

- **Dynamic Programming approach (DP)**

1. Create the Table and Determine the values on Cells

The table minT is used to store the minimum values for different weights. Each cell (minT[i]) depends on the cells (minT[i - weight of item]).

2. Determine the Direction of Movement Within the Table

The table is filled from left to right and from top to bottom. For each weight w from 1 to (totalW), and for each item, update (minT[w]) based on the value (minT[w - weight of item]).

3. The Code for the Dynamic Programming approach Which Fills the Table and print the result.

```cpp
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
#include <climits>
using namespace std;

int result(int totalW, vector<pair<int, int>>& ts) {
    vector<int> minT(totalW + 1, INT_MAX);
    minT[0] = 0;
    for (int currW = 1; currW <= totalW; ++currW) {
        for (const auto& t : ts) {
            int tW = t.first;
            int tV = t.second;
            if (currW >= tW && minT[currW - tW] != INT_MAX) {
                minT[currW] = min(minT[currW], minT[currW - tW] + tV);}}}
    return minT[totalW] == INT_MAX ? 0 : minT[totalW];}

int main() {
    int chestW, numT;
    cin >> chestW >> numT;
    vector<pair<int, int>> ts(numT);
    for (int i = 0; i < numT; ++i) {
        int w, val;
        cin >> w >> val;
        ts[i] = {w, val};}
    int res = result(chestW, ts);
    cout << res << endl;
    return 0;}
```

4. Table Example

- Assume the chest has a maximum weight capacity (totalW) of 6 grams. Consider the following:

A. Gold piece: 3 grams, 25 coins
B. Diamond piece: 4 grams, 30 coins
C. Silver piece: 1 gram, 10 coins

- Dynamic Programming Table Structure:

A. The (mint) table is an array where (minT[w]) represents the minimum possible value of the chest for a weight w.
B. Size of the table: totalW + 1 (to accommodate weights from 0 to totalW).

- Table Filling Process:

A. Initialize minT[0] = 0 (base case: no weight means no value).
B. For each weight w from 1 to totalW:
✓ Iterate through each treasure t.
✓ Update minT[w] if adding t provides a lower value than the current value in minT[w].

- Example Calculation:

Weight (w)	minT[0]	minT[1]	minT[2]	minT[3]	minT[4]	minT[5]	minT[6]
Value	0	10	10	10	10	10	25

Result:
The minimum possible value of the chest for totalW = 6 grams is minT[6] = 25.

• Conclusion

The dynamic programming approach is more efficient and suitable for this problem compared to the divide and conquer approach. It ensures that we can handle large input sizes within a reasonable time frame.