**Knapsack Problem**

Shane Ferrell

COP4533 Fall 2023 Programming Project

<https://github.com/shaneferrellwv/knapsack>

**Project Description**

This C++ program implements a solution to the Knapsack problem using dynamic programming. The Knapsack problem is a classic problem in combinatorial optimization. The program is designed to solve the problem of selecting a number of items with given weights and values to maximize the total value in a knapsack of a given capacity.

**Files in this Project**

1. knapsack.cpp: The main program file that contains the knapsack class and the main function.

2. item.h: A header file that defines the item class, representing the items to be put in the knapsack.

**How to Run**

To compile and run this program, you need a C++ compiler. Use the following commands:

g++ -o knapsack knapsack.cpp

./knapsack <input\_file> <capacity> <number\_of\_items>

• <input\_file>: A file containing the list of items (format: name weight value).

• <capacity>: The maximum weight capacity of the knapsack.

• <number\_of\_items>: The number of items to be included in the knapsack.

**Additional Information & Troubhleshooting**

**Implementation Details**

* **Class knapsack**: Manages the knapsack problem's solution. It includes methods for allocating/deallocating the 3D dynamic programming array, solving the knapsack problem, and tracing back the items included in the solution.
* **Class item**: Represents an item with a name, weight, and value. Includes a utility function to create a list of items from an input file.

**Input File Format**

The input file should be in the input/ directory and formatted as whitespace-separated values: name weight value.

The sum of the values of items in the input file should not be greater than 99999998. The program may produce unexpected behavior if provided an input file that does not meet this specification.

**Dependencies**

• Standard C++ libraries: <iostream>, <vector>, <algorithm>, <limits>, <fstream>, <sstream>

**Error Handling**

The program includes basic error handling for file input and format issues.

**License**

This project is open-source and free to use under the MIT License.

**Contact**

For bugs, suggestions, or further queries, please contact the developer at [shaneferrell@ufl.edu](mailto:shaneferrell@ufl.edu).

**Source Code**

**knapsack.cpp**

#include <iostream>

#include <vector>

#include <algorithm>

#include <limits>

#include "item.h"

using namespace std;

const int negativeInfinity = -99999999;

class knapsack

{

// user-defined member variables

vector<item> items;

int N, C, K;

int \*\*\*dp;

// intermediate member variables

int maxValue;

vector<item> sack;

// utility function to allocate 3D dp array

void allocate3DArray()

{

dp = new int\*\*[N + 1];

for (int i = 0; i <= N; i++)

{

dp[i] = new int\*[C + 1];

for (int w = 0; w <= C; w++)

{

dp[i][w] = new int[K + 1];

}

}

}

// utility function to delete 3D dp array

void deallocate3DArray()

{

for (int i = 0; i <= N; i++)

{

for (int w = 0; w <= C; w++)

{

delete[] dp[i][w];

}

delete[] dp[i];

}

delete[] dp;

}

// solves knapsack problem using bottom-up dynamic programming for recurrence relation

// dp[i][w][k] has max value of first i items of size exactly k whose total weight is exactly w

int solve()

{

for (int i = 0; i <= N; i++)

{

for (int w = 0; w <= C; w++)

{

for (int k = 0; k <= K; k++)

{

if ((i < k) || (w > 0 && k == 0) || (w == 0 && k > 0))

dp[i][w][k] = negativeInfinity;

else if (w == 0 && k == 0)

dp[i][w][k] = 0;

else if (items[i - 1].w > w)

dp[i][w][k] = dp[i - 1][w][k];

else

dp[i][w][k] = max(dp[i - 1][w][k], items[i - 1].v + dp[i - 1][w - items[i - 1].w][k - 1]);

}

}

}

return dp[N][C][K];

}

void traceback()

{

int bagValue = dp[N][C][K];

int c = C; // current bag weight

int k = K; // items removed

int n = N - 1; // current bag size

while (bagValue > 0)

{

for (int i = 0; i < items.size(); i++)

{

// weight of item chosen by bottom-up dynamic programming solution

if (c - items[n - i].w >= 0)

{

int chosenItemValue = bagValue - dp[N][c - items[n - i].w][k - 1];

// if item was chosen

if (chosenItemValue == items[n - i].v)

{

// add item to list of items in solution

sack.push\_back(items[n - i]);

// update bag value and bag weight

bagValue -= items[n - i].v;

c -= items[n - i].w;

// remove item from items list

items.erase(items.begin() + n - i);

k--;

n--;

break;

}

}

}

}

bagValue;

}

public:

// knapsack constructor

knapsack(string fileName, int capacity, int numberOfItems)

{

items = item::createItemsList(fileName);

N = items.size();

C = capacity;

K = numberOfItems;

allocate3DArray();

maxValue = solve();

traceback();

}

// knapsack destructor

~knapsack()

{

deallocate3DArray();

}

void printSolution()

{

if (maxValue >= 0)

{

cout << "Max value in knapsack: " << maxValue << endl;

cout << "Knapsack contains: ";

for (auto it = sack.begin(); it != sack.end(); ++it)

{

cout << it->name;

if (next(it) != sack.end())

cout << ", ";

}

}

else

cout << "There is no subset of exactly " << K << " items exactly totaling weight " << C << endl;

}

};

int main(int argc, char\* argv[])

{

string fileName = argv[1];

int capacity = atoi(argv[2]);

int numberOfItems = atoi(argv[3]);

try

{

knapsack k = knapsack(fileName, capacity, numberOfItems);

k.printSolution();

}

catch(const exception& e)

{

cerr << e.what() << endl;

}

return 0;

}

**item.h**

#include <iostream>

#include <vector>

#include <fstream>

#include <sstream>

using namespace std;

// class to represent item choices

class item

{

public:

string name;

int w; // weight

int v; // value

// item constructor

item(string name, int weight, int value)

{

this->name = name;

this->w = weight;

this->v = value;

}

// utlility function to create list of items from input file

// file must be whitespace-separated values within input/ directory

// in the form: name weight value

static vector<item> createItemsList(const string& filePath)

{

vector<item> items;

ifstream itemsFile("input/" + filePath);

// check if file exists

if (!itemsFile)

throw invalid\_argument("File input/" + filePath + " does not exist");

string line;

while (getline(itemsFile, line))

{

istringstream stream(line);

string name;

int w;

double v;

// check if file is in correct format (string int double)

if (!(stream >> name >> w >> v) || !(stream.eof()))

throw invalid\_argument("File " + filePath + " in invalid format");

items.push\_back(item(name, w, v));

}

return items;

}

};

**Traceback Function**

knapsack.cpp 75: public void traceback()

The traceback function works by first looking at the solution of the knapsack given by the bottom-up dynamic programming solution at dp[N][C][K], which is the max value of first N items of size exactly K whose total weight is exactly C.

Using the max value (the solution), the last item that was added to the knapsack is determined by finding the value of the last item added at the next subproblem. To find the value of the item added, we can look at dp[N][c – wi][k – 1] where c is the current weight of the knapsack, wi is the weight of any item in our items list, and k is the number of items removed from the knapsack, and subtract the value from the value of the knapsack to find the value of an item in the knapsack. We can choose any item that has this value and weight from our list of items.

Once the item has been determined, we can simply update the current knapsack value, weight, items removed, items possibly in the knapsack, and number of items in the knapsack, and add the chosen item to our list of selected items, and repeat the process until the knapsack has no remaining items, weight, or value.