DATA STRUCTURE AND ALGORITHM (ET0702) MINI-PROJECT REPORT

Name: Stanley Lin Sheng Yang (P2120795) and Chia Weng Chin (P2120836) Class: DCPE/FT/2A/24

Topic chosen: Topic 1 - Storage Rack Management System

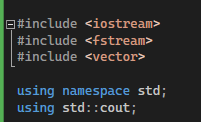
**Group members and contributions**

Weng Chin: Implementation of algorithm for the program, output of the results

Stanley: File I/O, accessing all 2D array inputs. Preparing, summing, and concatenating inputs to necessary variables such as double and vectors in the main function before algorithm. Display the user inputs on CLI. Research and strategy of algorithm.

A more detailed distribution of work is in our cpp file.

**Description of design and features of the program**



- **#include <iostream>** is a pre-processor directive that tells the compiler to include the header file "iostream", which provides input/output operations for C++ programs. This header file contains definitions for input/output streams (such as ’**cin**‘ and ‘**cout**‘) that allow you to perform input/output operations in your program.

- **#include <fstream>** is a pre-processor directive that tells the compiler to include the header file "fstream". This header file provides definitions for the **fstream** class and its derived classes **ifstream** and **ofstream**, which allow you to perform file input/output operations in your C++ program. The **fstream** class is used for input and output operations on files, the **ifstream** class is used for input operations, and the **ofstream** class is used for output operations. The functions declared in this header file can be used to open and manipulate files, read and write data to files, and close files when you're done with them.

-**#include <vector>** is a pre-processor directive that tells the compiler to include the header file "vector". This header file provides definitions for the **vector** class, a container class in the Standard Template Library (STL) of the C++ Standard Library. The **vector** class is a dynamic array that stores and manages a collection of elements. It provides various member functions for dynamically adding and removing elements, accessing elements, and resizing the array. The **vector** class is a useful and efficient data structure for many common tasks in C++ programming, such as storing sequences of values and performing operations on sequences of values.

-**using namespace std;** is a C++ statement that tells the compiler to use the **std** namespace in the current scope. The **std** namespace is the standard namespace defined by the C++ Standard Library, which contains definitions for many common classes, functions, and objects used in C++ programming, such as **cout**, **cin**, **vector**, and others. Using the directive, you can access the names defined in the **std** namespace without having to prefix them with **std::** every time you use them in your code.

We encountered some error during our coding process when it comes to **cout**, therefore instead of writing **std::cout** to output text to the standard output at every line , you can simply write **using** **std::cout** to solve the problem. This can make your code more readable and easier to write. However, it's generally recommended to use **using** directives sparingly, as overuse can lead to naming conflicts and other issues.

**Solving the requirements of the project**

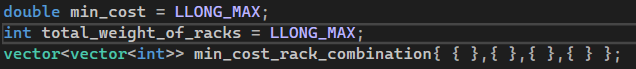
We saw these key requirements based the question given:

* Each rack has a different of weight limits and cost (to be read in from a text file)
* Develop a program to read in, from file, the number of racks, weight limits and costs of the racks, the weights of the 2 different types of packages and the number of packs for each type of packages.
* ***The program then*** ***prints out the most cost-effective arrangement of the packages on the racks and other useful information.***
* Assuming that a rack will be charged as long as the rack is being used
* flexible enough to take different combination of input data.

All functions that were coded:

1. void fileIO(int& number\_of\_racks, double package\_weight\_amount\_rack\_limit\_cost[][2])
2. void permutations(vector<int>& elements, vector<vector<int>>& current, int index, int number\_of\_racks, vector<double> vec\_for\_weight\_of\_individual\_rack, vector<double> vec\_for\_cost\_of\_individual\_rack)
3. void compute\_racks(int total\_weight\_of\_all\_racks, int total\_weight\_of\_input, int number\_of\_racks, vector<int> concatenated\_elements, vector<vector<int>>&racks\_vec, vector<double>&vec\_for\_weight\_of\_individual\_rack, vector<double>&vec\_for\_cost\_of\_individual\_rack)
4. main()

**Global Variable declared in the software**

****

* double min\_cost = LLONG\_MAX; à//global variable to initialize min\_cost and to compare with every other possible cost\_amount
* int total\_weight\_of\_racks = LLONG\_MAX; à global variable to initialize total weight of all racks for ease of comparing throughout different functions
* vector<vector<int>> min\_cost\_rack\_combination{ { },{ },{ },{ } };à Initialize the minimum cost rack combination on a 2D vector of vector of the lowest cost combination and being global variable will also be easier to call this 2D vector throughout the functions

**Function to find the most cost-effective arrangement of the packages:**



1. Firstly, fileIO function takes in several parameters:
2. number\_of\_racks: an integer representing number of racks specified by the user.
3. package\_weight\_amount\_rack\_limit\_cost: an array of double representing the 2 packages weight and the amount of it, and the limit and cost of individual rack specified by the user.

Text

Description automatically generated

ifstream text(“Input.txt”) reads data from a text file named "Input.txt".

textfile >> number\_of\_racks;

will take the first element of Input.txt and store it in number\_of\_racks.

Graphical user interface, application

Description automatically generated

Then it reads the data using nested for-loops. The outer for-loop iterates six times, the first 2 rows highlighted in blue is the details of packages and the next 4 is details of racks.

The inner for-loop iterates twice, which corresponds to the two columns in the array. The textfile input stream is used to read each value from the file and store it in the appropriate position in the array.

The while loop condition uses the eof() function to determine if the end of the file has been reached.

This loop will exit when there is no more data to read from the file.



1. Secondly, permutations function takes in several parameters:
2. elements: a vector of integers representing the input elements to be permutated.
3. current: a 2D vector of integers representing the current permutation of the constructed elements.
4. index: an integer representing the current position in the input elements vector
5. number\_of\_racks: an integer representing the total number of racks available to hold the elements.
6. vec\_for\_weight\_of\_individual\_rack: a vector of doubles representing the maximum weight capacity of each rack.
7. vec\_for\_cost\_of\_individual\_rack: a vector of doubles that represents the cost of each rack.

Text

Description automatically generated

if (index == elements.size()) à The function first checks if it has processed all input elements.

double total\_current\_cost = 0;

for (int rack\_number = 0; rack\_number < number\_of\_racks; rack\_number++) {

if (current[rack\_number].size() > 0) {

total\_current\_cost += vec\_for\_cost\_of\_individual\_rack[rack\_number];

}

}

If so, the function would store the result in a locally initialize variable “double total\_current\_cost” after it calculates the total cost of the current permutation by summing up the cost of all **non-empty racks**  (the question previously stated that we should add the cost whenever a package is added to the rack).

if (total\_current\_cost < min\_cost) {

min\_cost = min(total\_current\_cost, min\_cost);

min\_cost\_rack\_combination = current;

}

Thereafter, the function would compare the current cost with the current minimum cost. If the total cost is lower than the minimum cost, it updates the minimum cost and the minimum cost rack combination (min\_cost\_rack\_combination) to the current permutation.

Text

Description automatically generated

for (int rack\_number = 0; rack\_number < number\_of\_racks; rack\_number++) {

int current\_weight = 0;

for (int current\_package\_no = 0; current\_package\_no < current[rack\_number].size(); current\_package\_no++) {

current\_weight += current[rack\_number][current\_package\_no];

}

If the function has not processed all input elements, it loops through each rack and adds up the total weight that is currently on the rack.

if (current\_weight + elements[index] <= vec\_for\_weight\_of\_individual\_rack[rack\_number] )

checks if the current element can be added to that rack (i.e. the current rack's weight capacity has not been exceeded).

current[rack\_number].push\_back(elements[index]);

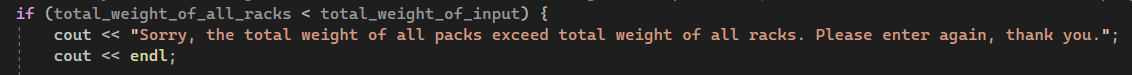
permutations(elements, current, index + 1, number\_of\_racks, vec\_for\_weight\_of\_individual\_rack, vec\_for\_cost\_of\_individual\_rack);

current[rack\_number].pop\_back();

If so, it adds the element to the current rack, recursively calls the function with the updated current vector and the next index, and then removes the element from the current rack before proceeding to the next rack.



1. Function to print out the most cost-effective arrangement of the packages on the racks and other useful information.



If the total weight of the packages input is more than the total weight of the racks can handle, a error message would be printed out

Text

Description automatically generated

This section of the code is responsible for printing out the results of the algorithm, i.e., the package allocation to each rack.

permutations(concatenated\_elements, racks\_vec, 0, number\_of\_racks, vec\_for\_weight\_of\_individual\_rack, vec\_for\_cost\_of\_individual\_rack);

First, the permutations function is called with the necessary arguments, including the input concatenated\_elements vector and empty racks\_vec vector to be filled with the package allocation to each rack. The function recursively generates all possible permutations of the packages, checking for the minimum cost of each possible combination, and updating the min\_cost variable and min\_cost\_rack\_combination vector accordingly.

for (int i = 0; i < min\_cost\_rack\_combination.size(); i++) {

cout << "Rack" << i + 1;

cout << endl;

The outer loop is iterating through the number of racks we've used in the solution. min\_cost\_rack\_combination.size() returns the number of racks that were used. For each rack, we print out the rack number using cout << "Rack" << i + 1 and add a new line using cout << endl.

for (int j = 0; j < min\_cost\_rack\_combination[i].size(); j++) {

cout << min\_cost\_rack\_combination[i][j] << " ";

Sum += min\_cost\_rack\_combination[i][j];}

The inner loop is iterating through each element of the rack. We print out the weight of each element in the rack using cout << min\_cost\_rack\_combination[i][j] << " ". We also keep track of the total weight of all the elements in this rack by adding the weight of each element to a variable called Sum.

if (Sum < vec\_for\_weight\_of\_individual\_rack[i] && Sum > 0) {

int wastage = vec\_for\_weight\_of\_individual\_rack[i] - Sum;

wastage\_total += wastage;

cout << "\t Wastage: " << wastage << "kg";

}

We then check to see if there is any wastage in the current rack. If the total weight of the elements in the rack is less than the maximum weight capacity of the rack, we know that there is some unused space in the rack. We calculate the amount of unused space (or wastage) by subtracting the total weight of the elements in the rack from the maximum weight capacity of the individual rack. We add this amount to a variable called wastage\_total which keeps track of the total amount of wastage across all racks. We then print out the amount of wastage using cout << "\t Wastage: " << wastage << "kg".

if (Sum > 0) {number\_of\_racks\_used++;}

Finally, we check if any elements were placed in the current rack by seeing if Sum > 0. If the rack has some elements, we increment the number\_of\_racks\_used variable, which keeps track of the total number of racks used in the solution.

1. Last but not least the main() function,

Text

Description automatically generated

int number\_of\_racks = 0;

//This array variable stores the array index of package weight ammount. The individual weight limit of racks and the cost of that specific rack

double package\_weight\_amount\_rack\_limit\_cost[6][2];

the 2 variables above is needed for fileIO() function, it and that function uses passing by reference to give those 2 variables new values in the main(). package\_weight\_amount\_rack\_limit\_cost[6][2]; will be crucial in the preparation of data later.

//package\_weight\_amount\_rack\_limit\_cost[0][0] stores the value of one of the package weight.

//package\_weight\_amount\_rack\_limit\_cost[0][1] stores the ammount of packages with the weight of package\_weight\_amount\_rack\_limit\_cost[0][0]

cout << "Amount of " << package\_weight\_amount\_rack\_limit\_cost[0][0] << " KG packs is " << package\_weight\_amount\_rack\_limit\_cost[0][1] << endl;

//similarily for this, package\_weight\_amount\_rack\_limit\_cost[1][0] stores the value of another package weight.

//package\_weight\_amount\_rack\_limit\_cost[1][1] stores the ammount of packages with the weight of package\_weight\_amount\_rack\_limit\_cost[1][0]

cout << "Amount of " << package\_weight\_amount\_rack\_limit\_cost[1][0] << " KG packs is " << package\_weight\_amount\_rack\_limit\_cost[1][1] << endl;

//this vector<int> stores both weighted packs (eg. 2 and 3) with their amount of it (eg. 3 and 4) together, for example 2 2 2 3 3 3 3.

vector<int> concatenated\_elements;

//with the above example package\_weight\_amount\_rack\_limit\_cost[0][1] will be 4

for (int i = 0; i < package\_weight\_amount\_rack\_limit\_cost[0][1]; ++i) {

//this will push in 2 each time it loops here.

concatenated\_elements.push\_back(package\_weight\_amount\_rack\_limit\_cost[0][0]);

}

Text

Description automatically generated

for (int i = 0; i < package\_weight\_amount\_rack\_limit\_cost[1][1]; ++i) {

concatenated\_elements.push\_back(package\_weight\_amount\_rack\_limit\_cost[1][0]);

}

/\*vector<int> ::iterator it;

for (it = concatenated\_elements.begin(); it != concatenated\_elements.end(); it++) {

cout << \*it << " ";

}\*/

//this vector<double> stores the cost of individual rack starting from lightest rack with least cost, for example 5.6 6.9 9.9 11.5

vector<double> vec\_for\_cost\_of\_individual\_rack;

for (int i = 2; i < 6; i++) {

vec\_for\_cost\_of\_individual\_rack.push\_back(package\_weight\_amount\_rack\_limit\_cost[i][1]);

}

//this vector<double> stores the weight limit of individual rack starting from lightest rack with least limit, for example 5 7 9 11

vector<double> vec\_for\_weight\_of\_individual\_rack;

for (int i = 2; i < 6; i++) {

vec\_for\_weight\_of\_individual\_rack.push\_back(package\_weight\_amount\_rack\_limit\_cost[i][0]);

}

vector<vector<int> > racks\_vec{ { },{ },{ },{ } };

//This for loop helps to sum the total weight of all packs, with the earlier example 2 2 2 3 3 3 3. Total\_weight\_of\_input will store 18

int total\_weight\_of\_input = 0;

for (int no\_of\_input\_element = 0; no\_of\_input\_element < concatenated\_elements.size(); no\_of\_input\_element++) {

total\_weight\_of\_input += concatenated\_elements[no\_of\_input\_element];

}

Text

Description automatically generated

//This for loop helps to sum the limit of all racks, with the earlier example 5 7 9 11. total\_weight\_of\_all\_racks will be 32

double total\_weight\_of\_all\_racks = 0;

for (int i = 0; i < vec\_for\_weight\_of\_individual\_rack.size(); i++) {

total\_weight\_of\_all\_racks += vec\_for\_weight\_of\_individual\_rack[i];

}

//instant enter racks if total weight of packs less than total limit of all racks

for (int i = 0; i < number\_of\_racks; i++) {

if (total\_weight\_of\_input <= vec\_for\_weight\_of\_individual\_rack[i]) {

min\_cost\_rack\_combination[i] = concatenated\_elements;

}

}

//this will call compute\_racks() function

compute\_racks(total\_weight\_of\_all\_racks, total\_weight\_of\_input, number\_of\_racks, concatenated\_elements, racks\_vec, vec\_for\_weight\_of\_individual\_rack, vec\_for\_cost\_of\_individual\_rack);

return 0;

}

**Operating environment:** Microsoft windows, Intel core i7, Visual Studio 2022, x86.

**Program User Guide**

The purpose is to aid the users on how to use the software. This guide can be purely instructional, with screenshots directing the users, and should be simple and intuitive. For the developer, the user guide is a reminder of the need to develop simple and intuitive operations.

Steps:

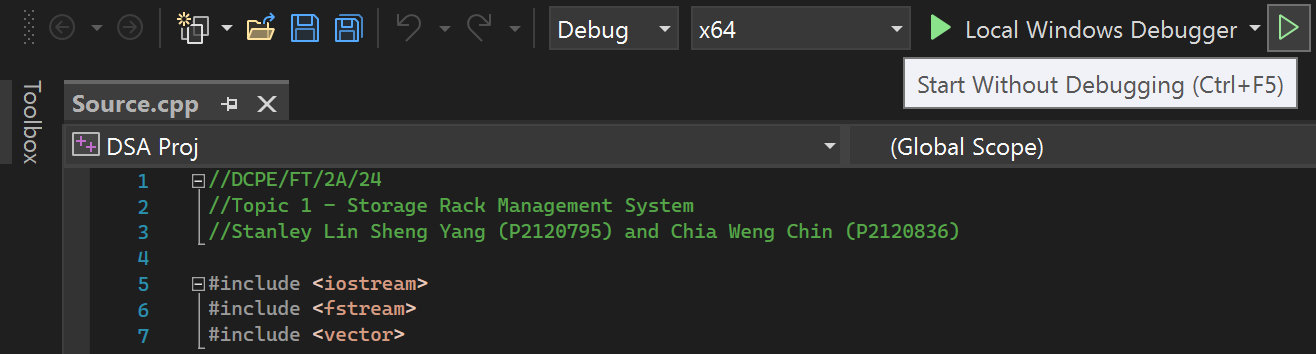
1. The user will first open “Input.txt” file. As shown in Fig. 1.
2. In the first row, enter the amount of racks. (In our screenshot, 4)
3. In the second row, first column, enter the weight of the pack. (In our screenshot, 3)
4. In the second row, second column, enter the quantity of that weight of the pack. (In our screenshot, 5)
5. In the third row, first column, enter another weight of the pack. (In our screenshot, 4)
6. In the third row, second column, enter the quantity of the other weight of the pack. (In our screenshot, 3)
7. Note for step 7 to 15: Start with the lightest weight limit and the lowest cost rack first.
8. In the fourth row, first column, enter the first rack’s weight limit. (In our screenshot, 5)
9. In the fourth row, second column, enter the first rack’s cost. (In our screenshot, 5.6)
10. In the fifth row, first column, enter the second rack’s weight limit. (In our screenshot, 7)
11. In the fifth row, second column, enter the second rack’s cost. (In our screenshot, 6.9)
12. In the sixth row, first column, enter the third rack’s weight limit. (In our screenshot, 9)
13. In the sixth row, second column, enter the third rack’s cost. (In our screenshot, 9.9)
14. In the seventh row, first column, enter the fourth rack’s weight limit. (In our screenshot, 9)
15. In the seventh row, second column, enter the fourth rack’s cost. (In our screenshot, 11.5)
16. Please remember to control + S after entering.

A screenshot of a computer

Description automatically generated with medium confidence

(Fig. 1)

1. Next, open “Source.cpp”.
2. Click on the green triangle, “Start without Debugging”. As shown in Fig. 2.



(Fig. 2)

1. As shown in Fig. 3, The program then prints out the most cost-effective arrangement of the packages on the racks and other useful information.

Text

Description automatically generated

(Fig. 3)

1. If the user entered the weight packs, till sum of weight of all packs exceeds the sum of weight limit of all racks. We will leave a friendly prompt to the user, and not display the racks. As shown in Fig. 4.

Graphical user interface, text

Description automatically generated

(Fig. 4)

**Research and reference**

Dynamic Programming - Learn to Solve Algorithmic Problems & Coding Challenges by freeCodeCamp.org (<https://www.youtube.com/watch?v=oBt53YbR9Kk>)

Greedy Algorithms Explained by Tech with Tim (https://www.youtube.com/watch?v=lfQvPHGtu6Q)

Creating all possible k combinations of n items in C++ from stack Overflow

(https://stackoverflow.com/questions/12991758/creating-all-possible-k-combinations-of-n-items-in-c)

**Individual reflection on this project journey**

Stanley’s reflection:

Through the past 17 weeks, I constantly learned data structure and algorithm knowledge such as calling functions using passing-by references, pointers, list, vector, queue, stack, and many types of searching methods. These knowledge allowed me to start the project with a strong understanding of what was required. However, what was taught in class may not be enough and I began searching for algorithms beyond classroom. Since this project requires me to find ***the most cost-effective arrangement of the packages on the racks,*** many keywords start to appear in my head, “optimization”, “greedy”, “randomization”. I began searching on Google and YouTube, along those lines, and I found Knapsack algorithm from a YouTube channel called “Tech with Tim” and did try to understand. It does check out the “greedy” and “optimization”, but it doesn’t check out the “randomization” aspect and it works well on a small scale but may not a big one.

I stepped back from Knapsack algorithm and targeted “randomization” way of doing, and found a mass randomization algorithm on stack Overflow, with the title called “Creating all possible k combinations of n items in C++ from stack Overflow” and came up with a way to brute force test every possibility but it appears to have O(2^n), and this had to be scratched as well. I continued researching and send resources to my teammate.

Then I consulted the teacher, and she mentioned it is possible to do this project well just by using what was taught. With her advice, I read the MiniProject.pdf again, and realized I just have to start clean coding as many requirements as possible. I started to do file I/O requirement, accessing them from a 2D array. Then prepare, summing, and concatenating inputs to necessary variables such as double and vectors in the main function before algorithm. It turned out to be great, and the project finally has a clear direction.

This project felt different from other programming projects of DCPE, I was able to better appreciate data structures and algorithms and can finally call C++ my main programming language. This project has encouraged me to adopt a more structured and logical approach, which will be beneficial in the future. The skill set is vital in a programming job interview. I had gone for a Singtel Shine Cadet coding interview in Year 1 and start of Year 2. And of course, I haven’t gone through this module, and I failed it miserably. But now, after this project and this module, I feel more prepared in a data structure and algorithm interview.

Weng Chin’s reflection:

Through working on this problem of packing items into racks, I learned about various algorithms and techniques that can be used to find the optimal solution. Specifically, I gained experience using recursion to generate all possible combinations of items and brute force search to find the combination with the lowest cost. Additionally, I learned about the importance of considering constraints, such as the weight limit of the racks, to ensure that the solution is feasible. This made me come to the conclusion of choosing the Branch and Bound Algorithm through many rounds of trial and error.

One major regret that I have after the end of this project is not being able to resolve the problem of solving the edge case of if the total weight of the package exceeds the total weight the racks can handle. I feel that with a bit more time, I would like to try finishing the code with not particularly any algorithm, but by filling up from the bottom rack to the top by maximising the package weight I can fit in each rack at every stage since the eventually all the racks will need to be filled. This might overcome the problem that branch and bound had, which was that whenever a permutation had a rack that exceeds the total weight of the rack can handle, the permutation would be cancelled out.

Most importantly, the project also has taught me how to approach code with more diligence. I encountered multiple roadblocks while arriving at our final code. For example, I faced many problems incorporating the 2D vectors in our code as it was a topic, I had little practice with. However, through a bit of persistence and a lot of practice, I am now more confident in my ability to solve problems that require using vectors. Also, the need to switch between different algorithms can be quite frustrating at the start because I was pretty lost at the start because my team could not really decide on an algorithm to carry out our project on. Eventually, with help from the teacher and some friends, we became more certain about the direction of our code.

Overall, this problem challenged me to think critically and algorithmically, and allowed me to develop my programming skills in a practical way. By working through the solution step by step, I deepened my understanding of programming concepts and gained confidence in my ability to solve complex problems using C++.