

**Overview**

Grain holding company gets orders to either take in or output grain to trucks. Each order has different types of grain and different amounts, these differences influence the revenue received for that order. The company has a limited number of silos with limited space and limited time in the day to process all the orders.

**Basic Problem**

The grain holding company receives too many grain orders each day and must select and schedule orders to maximise profit. Each order of grain has a time to complete and an expected revenue depending on grain type and amount, the amount of space in the silos is irrevelant and orders can only input grain. Treaing the order's time as weight, the revenue as value and the order as a object to put into the knapsack, the knapsack algorithm can be used to select the optium schedule of orders.

<https://www.geeksforgeeks.org/0-1-knapsack-problem-dp-10/>

<https://www.guru99.com/knapsack-problem-dynamic-programming.html>

As the problem is simple the algorithm can be expected to test all possible selection options and return the best choice.

**Advanced Problem**

The problem now has more dimensions than just time and revenue. Free space and the amount of grain available in the silos now matters with each grain type going to it's own silo and orders either inputing or outputting grain.

Expectation is this problem is too complex to test all possible schedule selections. May require a greedy approach and may return a good enough solution but may not return optimum solution.

<https://www.scirp.org/journal/paperinformation.aspx?paperid=87646>

**Program Structure**

Order requests are files read by the program on starting.

Program outputs to console using letters for animation

User can select random assignment, to show algorithm is more efficient

User can also create their own schedule and get program to run it. To show how difficult it is to do by human and just how more efficient algorithm is.

**Main class**

Main function

-create array of all files in folder

-iterate over files in array to create order instances

-put orders through knapsack algorithm

-print output

Knapsac algorithm function

-takes a array of orders

-returns optimum selection of orders for the day

Get files in folder function

-given a folder address

-gets address for all txt files in folder

-returns array of address for txt files

Read file and create order function

-takes string address of txt file

-reads the data in the file

-creates an instance of order class

-returns the instance of the order class

Randomise Algorithm

-takes a array of orders

-returns random selection of orders for the day

**Order Class**

Variables:

Grain type

Size

Load type: either unloading or outloading

Functions:

Get grain type

Get size

Get load type

**Silo Class**

Variables:

Grain type

Grain limit

Current grain level

Functions

Get grain type

Get size

Update grain level

Update grain type

Overflow function

Underflow function

**Business class**

Represents the data and logic of the business

calculates value of order by multiplying grain value with amount

calculate time grain time multiplied by amount

variables:

Day schedule: array represents hours in the day

Revenue: int for money. Can be positive or negative

Silos: array of instances of silos

**functions**

Update schedule

Get schedule

Update revenue

Get revenue

Get silos

**Display Class**

Used to print display and animations

**Road Map**

Iteration one. -complete

Program can read all order txt files from a folder and turn them into instances of order class

Program has business class

iteration two - complete

order class has more variables

business class has better functions

create more order data

Iteration three.

update business class so schedule and values work out - done

review how amount and order times work - done

review knapsack algorithm - done

review how best to set up C++ code - done

clean up code - done

iteration four

Implement basic knapsac algorithm - done

inloading only - done

Program goes on time and $ value of order - done

No issue with silos overfilling - done

get code to handle more than 3 orders

create more order data

clean up code

iteration five

review genetic algorithms (multivariate knapsack)

review how to update business to make the problem multivarate

update design doc as appropriote

Iteration five

include silos

Knapsac problem can now include outloading as well

Iteration six

Knapsac problem now takes into account grain overflow or underflow