ALGOSUP Project 5 2023

Technical Specification (Draft May 16th)

# Audience

This document is primarily intended for:

* Software developer – to understand the user and technical requirements, and to guide decision-making and planning. Help them understand risks and challenges, customer requirements, and additional technical requirements and choices made.

Secondary audiences

* Program manager – to validate against the functional specification, and the client expectations.
* QA – to aid in preparing the test-plan and to use for validating issues.
* Project manager – to help identify risks and dependencies

# Deliverable

A console application that determines the “best” blending process to achieve a desired proportion of wines using a fixed number of tanks, and in the fewest steps.

## Details

The possible set of wines to be processed is up to 400, and the number of tanks available for blending can be up to 300 (700 including source tanks).

The act of blending tanks consists of taking a series set of inputs tanks (1 or more) combining them and outputting them into 1 or more output tanks.

Evaluating the best blending process is assumed to be an intractable problem due to the sheer number of possible combinations, so the goal is to achieve the best possible result in a reasonable amount of time.

# Requirements

* Able to run on Windows and Mac.
* Target blend recipe is user defined.
* Sizes of tanks is user defined.
* Sizes of tanks containing initial wines.
* Output steps
* Output reached blend.
* Reproducible – given the same input, produces the same output.
* Steps are never impossible.
* Steps never leave a tank partially full or empty

## Nice to have:

* Usage instructions if arguments are missing or invalid.
* Error messages if input files are missing or output directory is not writable.
* Configurable options affect program behavior can be provided via JSON file.
* Output progress as it happens.
* Change format of output.
* Can redirect output to a file via command-line or options
* Can load and validate a process.
* Ability to associate names with tanks and/or wines.
* Changing the heuristics
* Change how evaluation happens: optimize to minimize wasted wine.
* Interactive mode, so user can step through a process.
* Logging

# Priorities

1. Correctness of algorithm – no half-full tanks, no illegal transfers.
2. Reasonable performance at scale (doesn’t take hours to run).

# Technical Architecture

## Technology Choice

* Console application
* Programming language C# 9
* Runtime .NET 6.0
* No additional libraries beyond the System library are required.

C# was chosen because it works on multiple platforms, performs well, has great unit testing tools, is robust, and makes development quick. Turning a C# program into a library is quick and easy.

## Input Data

* Three files on the command line:
  + Arg[0] - a text file containing list of whole numbers, each on a separate line representing relative tank sizes.
  + Arg[1] – a text file list of numbers, representing relative wine amount in target blend.
  + Arg[2] – a JSON options file (optional)

## Output Data

* Three text files
  + Result.txt – a list of wine percentages at the end of the process
  + Steps.txt – a list of steps describing the process
  + Output<timetamp>.json – a time stamped file containing the input data and result data for

# Non-Requirements / Out of scope

* Graphical user interface
* History of running the application
* Security and passwords
* “Previous use history” of tanks
* Blending data “notes from the tasting team”
* Data stored in a separate database – JSON files will be output that can be used as records
* Security concerns

# Operating Requirements:

* Should be able to compute a recipe in under 60 minutes with
  + 400 wine blends
  + 300 tanks

# Assumptions

* It doesn’t matter what the specified wine amounts add up to (1, 100, other)
* Wine cannot be transferred from a tank back into itself during the same step/operation..
* Wine cannot be added or removed from the system.
* Whether we count the number of transfers, or group them together won’t make a difference.
* When considering how close two wines are, we can use Euclidean distance.
* The total wine in the system is a constant
* Identifying an optimal result may not be possible, we are satisfied with good enough.

# Configurable Options

The following are ideas for what could be considered configurable options. They are not obligatory.

* Whether to optimize for minimum waste or not
* To run the program in parallel or not
* Verbosity of output
* Interactive mode or automated
* Input and output file names.
* The maximum number of input and output tanks can be considered together in a blend.

# Key Functions, Operations, and Algorithms

Some of the key operations that the software will perform and that need to be represented via functions are:

* Evaluating the list of possible transfers for a given tank
* Evaluating the “closeness” of two blends of wines – this is performed by normalizing the blend in each mix (using the vector normal function for N dimension) and performing a Euclidean distance operation.
* Compute a score for two system states to choose which one is better.
* Quickly compute which tanks add up to a given amount: a problem known as the *combination sum problem*.

# Code Design Principles

The following are the coding design principles:

* Rely minimally on third party libraries.
* Minimize use of mutable data types – when systems can change during execution it makes reasoning about them very hard, and can hinder parallelism.
* Enable system to evaluate possible states in parallel.
* Minimize use of static data – flexibility in the system is very important and static data tends to create hidden coupling and limit refactoring.
* Emphasize improved algorithmic complexity over micro-optimization
* Use unit tests to assure correctness during development.
* Use assertions to validate key assumptions
  + Assure that the amount of wine in a system is always constant
  + Assure that tanks are always full
* Assure that all types are exposed publicly so that the program can be used as a library in other application.

# Ideas and Hypotheses

The following are some hypotheses about characteristics of the problem domain that could benefit from further research and could help guide the implementation.

## Unproven Hypotheses

* Given a state where the best wine blend is X, it would never be desirable to backtrack to a system state that is worse.
* A purely “greedy” approach won’t produce an optimal result.
* Precomputing the list of possible transitions based purely on tank volumes can be desirable.
* The Manhattan distance metric might yield the same results
* Computing stuck states could be useful and might want to be penalized.
* It could be useful to score the ability to create blends that can be combined with the target blend effectively, rather than just the combined.
* Knowing what tank combinations add up to a particular value might optimize the process of figuring out transition possibilities.
* Sometimes random searching a tree of possibilities might be good to prevent the system from getting stuck in local minima.
* A deeper lookahead, even if all cases can’t be considered might still be desirable.

## Proven Hypotheses

* A transfer from one tank to a single other tank does not change worth considering.
* Some tanks are not reachable and can be removed from consideration.
* The set of tank lists that can be used in any transfer can be pre-filtered.

# Main Classes

* Configuration – Data set up on program initialization based on inputs from user that don’t change. Contains: list of tank sizes, the target mix, and configuration options.
* Configuration Options – easily serialized set of values used to control the parameters of operation, such as heuristics.
* Mix – a list of numbers corresponding to the amount of each wine in the tank
* State – a list of mixes, one for each tank.
* Tank list – a set of indices representing a set of tanks. Indices should be non-repeated and ascending.
* Transfer – contains an input tank list and output tank list. Used for considering possible steps, and storing the list of chosen steps to arrive at the final state.

# Challenge

* There are extremely large numbers of operations that are possible from any point in the system.
* The number of combinations of tanks possible is 2^N. The number of combinations of tanks that add up to a particular volume is less than that, but figuring them out in a reasonable amount of time is still long.
* It can take a long time to evaluate the score of a position (look at the mix in each tank and compute the distance from the target).
* Many states can be considered equivalent, and transitioning between them is pointless.
* The system can get caught in a loop and repeat previous states.
* It is not clear how we can quickly identify whether a state has already been visited or not. It may require some alternative.
* There does not seem to be any overlapping sub-problem and optimal substructure that would allow us to find a solution via dynamic programming.
* Some algorithms are fine for small inputs but take far too long for long inputs. An iterative approach that takes into account a maximum time length could be desirable.

# Possible Bugs

* The system may accumulate error causing the amount of wine to fluctuate
* A transfer may specify a source tank that is empty (or that does not exist)
* A transfer may specify a target tank that is not empty (or that does not exist)
* A transfer may specify a tank more than once.
* A transfer may specify more inputs or outputs than allowed.
* The system takes too long to execute.
* The mix specified in a tank does not add up to the correct volume.
* The system may get stuck and not improve the current wine.

# Development Process

* Develop a prototype console application.
* Write some unit tests.
* Create some sample input files.
* Develop a brute force or random algorithm.
* Assure that the program does not generate invalid intermediate steps using asserts and explicit error checking.
* Develop a more complex solution based on creating a lookahead tree:
  + evaluate a state based on the best reachable state N moves in the future
* Experiment and Optimize to Improve Scalability
  + Explore the limitations of the algorithm with different sizes of inputs.
  + Try different heuristics to simplify and reduce the problem space
  + Experiment with pre-computat
  + Experiment with greedy algorithms

# Glossary

* Transfer – a transfer of wines from one or more tanks to one or more tanks.
* Step – a step is one or more transfers that can be executed simultaneously
  + Could be one transfer per step, doesn’t change much
* State – the amount of each type of wine in each tank
* Blend – aka mix – a set of proportions of wines
* Blending process – aka “the process” and “the algorithm” – a series of steps that must be carried out to blend the mix
* Distance – closeness - given two wine blends, this a measure of how similar they are.