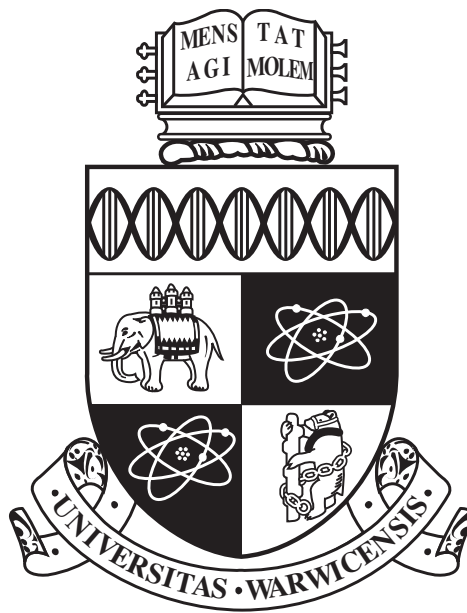

CS132: Computer Organisation & Architecture

Coursework 2



Contents

| | | |
|----------|--------------------------------------|----------|
| 1 | Introduction | 2 |
| 1.1 | Problem 1 | 2 |
| 1.2 | Problem 2 | 2 |
| 2 | Problem 1 | 2 |
| 2.1 | Power set | 2 |
| 2.2 | Pseudocode | 3 |
| 2.3 | Explanation of code design | 4 |
| 2.4 | Flowchart | 5 |
| 3 | Problem 2 | 6 |
| | References | 6 |

1 Introduction

1.1 Problem 1

1.2 Problem 2

2 Problem 1

2.1 Power set

The first problem of coursework 2 introduces the implementation of a "Power set" in code. However, in order for this code to be written, it is first important to break down as to what requires to actually be defined.

Definition 2.1. Power Set

The power set of a finite set S , denoted as 2^S , is the set that contains all subsets of S as its elements. Formally,

$$2^S = \{X : X \subseteq S\}$$

The cardinality of the set (number of elements denoted as $|S|$), is then, as a corollary:

$$\begin{aligned} |2^S| &= 2^{|S|} \\ &= \sum_{i=0}^{|S|} \binom{|S|}{i} \end{aligned}$$

Note that for the sake of this paper, we will not be discussing if S is infinite, as the code will be implemented with the assumption that the input is also finite.

And the intuition behind this corollary is important for our implementation, as it in fact gives us a big hint as to how Problem 1 could be implemented as code. Consider the sets $S = \{x, y\}$, $S' = \{x, y, z\}$, 2^S and $2^{S'}$. In terms of decision trees for their power sets, it would be visualised as the following:

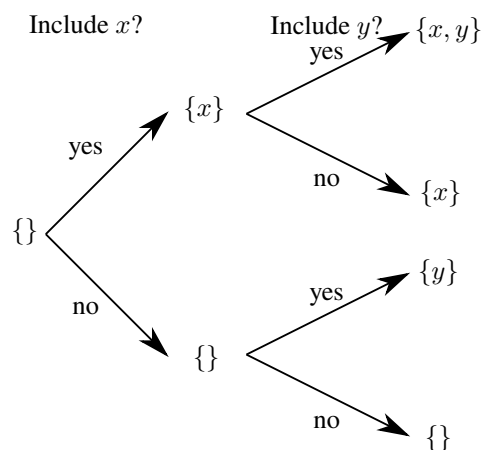
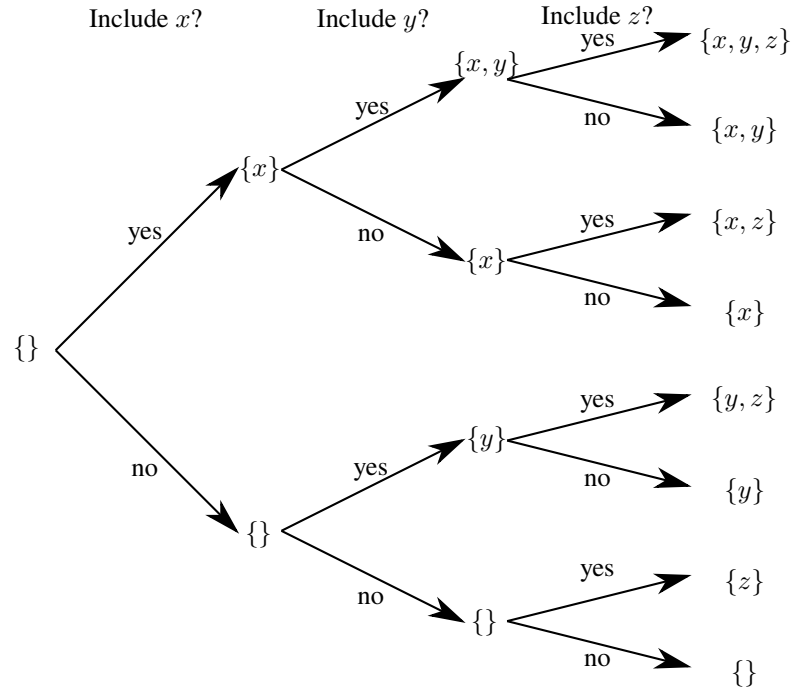


Figure 1: Visualisation of the power set 2^S

Figure 2: Visualisation of the power set $2^{S'}$

These figures make it apparent as to how the cardinality $|2^S| = 2^{|S|}$ precisely works. That is, for each new element to the set S , we must add a new branch of yes or no decisions for all previously existing power set elements. When the answer is no to the new element for all branches, we get 2^S . With all answers yes, we obtain $x \in 2^S : x \cup z$.

Formally, let us define $S' = S \cup \{\zeta\}$ where ζ is the new arbitrary element of a finite set S . Then, we can obtain the following formula for their power set:

$$2^{S^*} = \{x \in 2^S : x \cup \{\zeta\}\}$$

$$2^{S'} = 2^S \cup 2^{S^*}$$

2.2 Pseudocode

Now that the intuition and the mathematical formula has been well defined, we could write the pseudo code that we will implement into C. This will help us plan the structure of what is to come when writing our code.

Data: User input of elements into our set S

Result: A printed 2^S

Begin

Take input from user;

Initialise output variable array;

while Our counter is not at the size of $|S| + 1$ elements **do**

 Initialise counter = 1;

if first element **then**

 Add first singleton set to output variable;

else

 Add new element to each existing set in output variable and store them in output variable;

end

 counter = counter+1;

end

Add empty set to output variable;

Print output variable;

End

Algorithm 1: Power set pseudocode

Now that the pseudocode has been defined, we could move onto the written code and explaining each line's design.

2.3 Explanation of code design

This section will require that the file *powerset.c* is opened as the lines of code that are expressed in this section correspond to the lines of code of that file.

- (Lines 8 – 16) The definition of a two power was defined to keep the numbers in unsigned integers. This helps reduce code as type casting is not required as the original package assumes the output not to be an integer
- (Lines 36 – 40) `arraySize` is converted into integer during the actual method to ensure that the user does not go over and including the number 20.
- (Lines 52 – 55) The sizes of the variables are as following:
 - `S` as input `arraySize` as this is the amount of elements it holds.
 - `powerSetS` as $2^{\text{arraySize}}$ as this is the theoretical amount of elements.
 - `powerSetSStar` as `arraySize – 1` because it will only require at maximum have the same amount of elements that exist in S at that time. However, we subtract another -1 because we add the empty set separate for our algorithm.
 - Memory size as 100 as this is a good amount to assume that the string will not be longer than 99 characters.
- (Lines 104 – 107) It was decided that all subsets will be listed in new lines because otherwise it would be a very long singular line and even difficult to read.

Assumptions include that the set is finite and is not larger or including cardinality size 20. The reason for this is because $2^{20} > 10^6$. Furthermore, unsigned int was used as the general type for numbers as we cannot have negative numbers and int is sufficient for the size of arrays that we are working with. Error catching was implemented for cardinality size to ensure a smooth run. The number error catching was re-used from coursework 1. Pointers for strings were used as it is one of the best and compact methods available in C (“C - Pointers and Strings”, n.d.) (“Array of Pointers to Strings in C”, 2020).

2.4 Flowchart

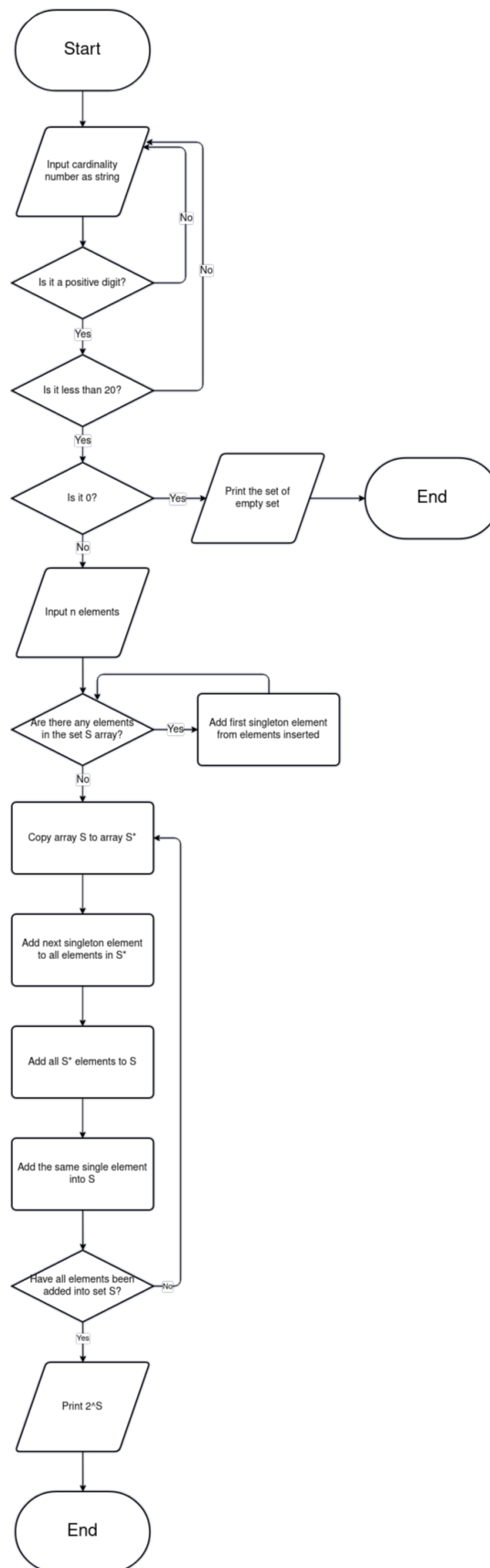


Figure 3: Flowchart of the implemented algorithm

3 Problem 2

References

- Array of Pointers to Strings in C [Article]. (2020). Retrieved 2022-01-14, from <https://overiq.com/c-programming-101/array-of-pointers-to-strings-in-c/>
- C - Pointers and Strings [Article]. (n.d.). Retrieved 2022-01-14, from <https://dyclassroom.com/c/c-pointers-and-strings>