# 資料結構報告範例

王語晨

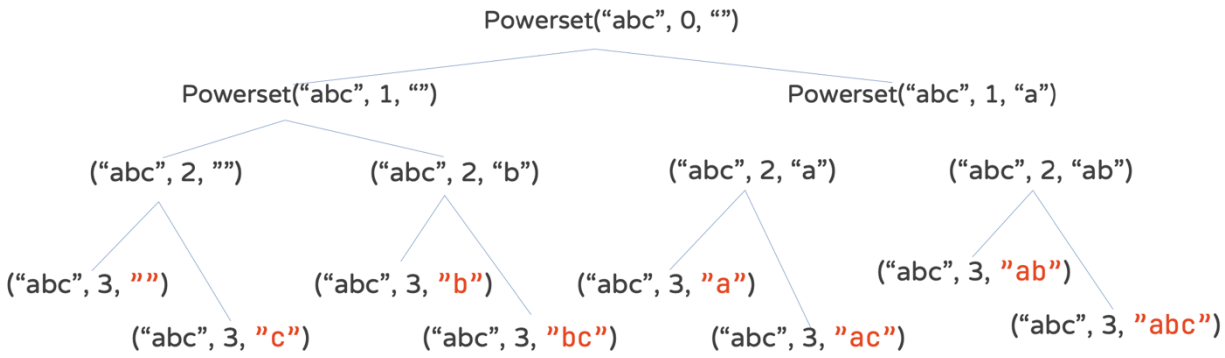
July 26, 2024

# CONTENTS

# CHAPTER 1

## 解題說明

以powerset函數實作，已知powerset函數計算公式如下：




實作參見檔案 powerset.cpp，其powerset函式：

```
int powerset( string a, int b, string c) {
      if (b == a.size()) {
            cout << c << " ";
            return 0;
      }
      powerset(a, b + 1, c);
      powerset(a, b + 1, c+a[ b ]);
}

```

Figure 1.1: powerset.cpp

# CHAPTER 2

# 演算法設計與實作

```
int main() {
	string a, c;
	a = "abc";
	c = "";
```

Figure 2.1: main.cpp

# CHAPTER 3

# 效能分析

$$f(n) = O(n)$$

## 時間複雜度

$$O(2^n)$$

Powerset函式使用遞迴，並且在每一次呼叫時會進行兩次遞迴；一次是跳過當前的字元(b+1)，一次是將當前字元添加到結果集合中（b+1,c+a[b]）。這會導致每個字元有兩種型態：取或不取。對於一個長度為n的字串，遞迴的數量將會是2^n，因此時間複雜度為O(2^n)。

## 空間複雜度

$$S(P) = O(n*2^n)$$

遞迴的深度最多會是n，而每次呼叫會創建一個新的結果字串（即c+a[b]）。由於最多會有2^n個結果集合，因此額外的空間複雜度與這些結果集合成正比。結果集合長度最多為n，因此空間複雜度為O(n*2^n)。

# CHAPTER 4

# 測試與過程

```
$ g++ main.cpp -o main.exe && ./main.exe“
“ ” “c” “b” “bc” ” a” ” ac” ” ab”
” abc”
```

Figure 4.1: shell command

## 驗證

powerset 函數接受三個參數：

a：輸入的字串（在例子中為 ″abc″`）、b：索引（用於跟蹤字串中當前處理的位置）、c：結果變量（用於累積當前子集）。然而該函數在每一個位置上有兩個選擇：1.跳過當前字元，直接遞迴到下一個索引位置、2.選擇當前字元並將它加入當前子集（即 `c + a[b]`），然後遞迴到下一個索引位置。而終止條件為if (b == a.size())，這是遞迴的終止條件。當 b 等於字串長度時，表示已經處理完所有的字元，這時將當前的子集 c 輸出並返回。

以 a = ″abc″ 為例來驗證：

1. 當 a = ″abc″`，主程式呼叫 powerset(a, 0, ″″)`，開始遞迴。

# CHAPTER 4

2. 程式從索引 b = 0 開始，當前字元為 `"a"`：第一種情況（跳過 "a"）：呼叫 powerset(a, 1, "")，繼續處理後續的字元。第二種情況（選擇 "a"）：呼叫 powerset(a, 1, "a")，將 "a"加入子集中。

3. 重複這個過程對 "b" 和 "c" 進行處理，直到所有子集被生成出來。

以下是遞迴的每一步：

開始時 c = ""。

首先第一層：是否加入 "a"，結果為：跳過 "a" → ""，選擇 "a" → "a"，然後進入第二層：是否加入 `"b"`，結果為：跳過 "b" → ""，"a"，選擇 "b" → "b"，"ab"，最後進入第三層：是否加入 "c"，結果為：跳過 "c" → ""`，"a"，"b"，"ab"，選擇 "c" → "c"`，"ac"，"bc"，"abc"，最終輸出："" "c" "b" "bc" "a" "ac" "ab" "abc" 。