



進階程式設計學習紀錄

課程名稱：進階程式設計

By 林侑則



目錄

1

摘要

我選擇這門課的理由及動機，以及希望在這堂課裡學習到的能力

2

作品說明

簡述作品內容及大綱

3

過程與成果

呈現我的學習過程及成果

4

心得反思

對於這堂課我的想法及反思

摘要

了解我的選課動機及希望在這堂課學到的能力

「我為何要選這堂課？想藉此學到甚麼能力？」

在一年級時我接觸到了資訊課並學了基礎的C++程式設計，由此清楚了我的性向及未來的規劃，因此在高三的選修課程我便毫無遲疑的選了這堂「進階程式設計」，希冀在這堂課我能學習到更進階的演算法，並強化我的邏輯、思考能力，也希望可以為未來的大學課程打下基礎。

作品說明

簡述作品內容及大綱

這門課我們主要以解高中生程式設計系統 ZeroJudge 的題目為主，並且在這過程中學習解題或優化的技巧，因此我將在這份作品中挑選我印象特別深刻的題目，以我最初的想法及在此課程學習到的技巧作對比的方式，呈現我的學習過程及收穫

課程主要方向：

1. C++ 資料結構 STL 容器的用法
2. 排序演算法
3. 搜尋演算法
4. 貪婪演算法
5. 分治演算法
6. 樹搜尋演算法
7. 動態規劃



過程與成果

呈現我的學習過程及成果

1) ZeroJudge a982. 迷宮問題#1

題目敘述:

給你一個 $N \times N$ 格的迷宮，迷宮中以#代表障礙物，以.代表路，你固定在(2,2)出發，目的地是(n-1,n-1)，求包括起點和終點，最少路徑的長度。

解題思路:

第一次看到這個題目我幾乎完全沒有想法，但在我們老師簡介了一下DFS及BFS演算法後我總算看到解題的希望了，肯定就是使用這兩個演算法的其中一個！

然而，這題要使用DFS或是BFS呢？

在我思考一番後採用了BFS演算法，因為這個方式是讓各個路徑同時往下走一步，走完後再依序走第二步...以此類推，這表示第一個走到出口的路徑長度也將是最短的，這即是使用BFS演算法的理由。

而在實踐BFS演算法時我選擇以學過的STL容器queue搭配pair，最後解決本題。

程式碼實踐

```
char maze[105][105];
int len[105][105];
queue<pair<int,int>> q;           // 宣告一個存放 pair 的 queue
memset(len,-1,sizeof(len));      // 將len陣列的初值都設為-1，表示沒走過
int dr[4]={0,0,1,-1},dc[4]={1,-1,0,0}; //將四個方向用陣列儲存
for(int i=0;i<n;i++)
    for(int j=0;j<n;j++)
        cin >> maze[i][j];

q.push({1,1}); //將初始點放入 queue
len[1][1]=1;
maze[1][1]='#';

while(!q.empty()) // 當 queue 內還有點
{
    pair<int,int> p=q.front(); // 讀取 queue 內的點
    q.pop();                  // pop 掉這個點
    int r=p.first,c=p.second;

    for(int i=0;i<4;i++){ //開始跑迷宮
        int nr=dr[i]+r,nc=dc[i]+c;
        if(maze[nr][nc]=='.')
        {
            q.push({nr,nc});
            len[nr][nc]=len[r][c]+1;
            maze[nr][nc]='#';
        }
    }
}
```

結果: AC (8ms, 372KB)

2) ZeroJudge d908: 4. 最佳路徑

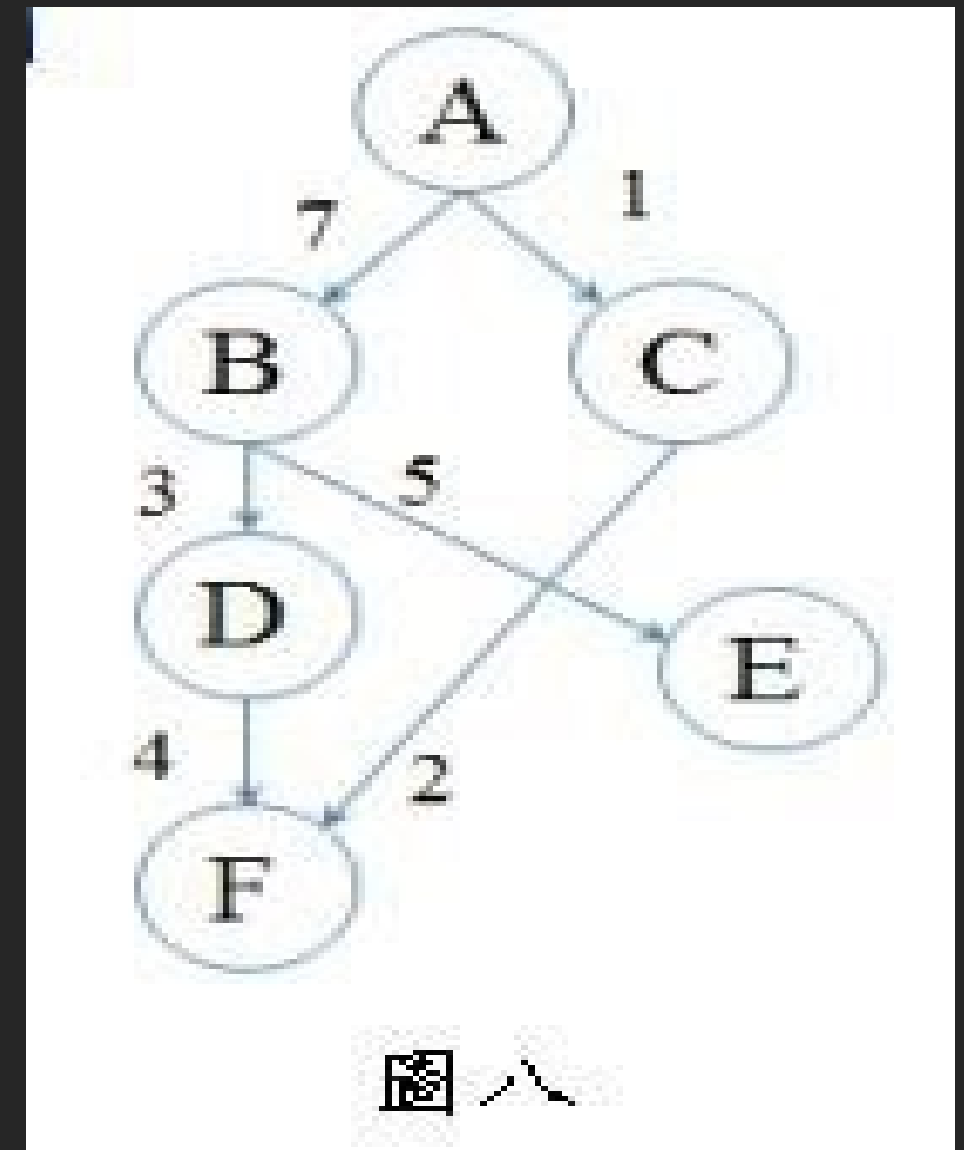
題目敘述:

依據題目所給定的有向圖，在以有向圖中的某一個節點做為起始節點的所有可能路徑中，計算其中擁有最大權重總和的路徑之權重總和值為多少？我們假設在此討論的有向圖都不會存在有一條路徑的起始節點與終止節點是同一個。如圖八的有向圖中，所有以A為起點的路徑之中，以A->B->D->F這條路徑的權重總和值最大(等於14)。

解題思路:

學會了BFS及DFS演算法後很自然的清楚這題必須使用其中一樣，而使用DFS的原因是能在遍歷各個節點時紀錄路徑的權重，這是BFS無法如此簡單做到的。

程式碼的實踐使用相鄰矩陣及遞迴，值得一提的是我寫完後送出卻發現只有80%的測資正確，經過老師的提點後才發現我忘記考慮到形成“環”的情況，因此我加上了VST陣列紀錄節點是否有造訪過，並且順利通過100%測資。



程式碼實踐

結果: AC (2ms, 340KB)

```
int g[26][26];
int vst[26][26]; //增加 vst 陣列

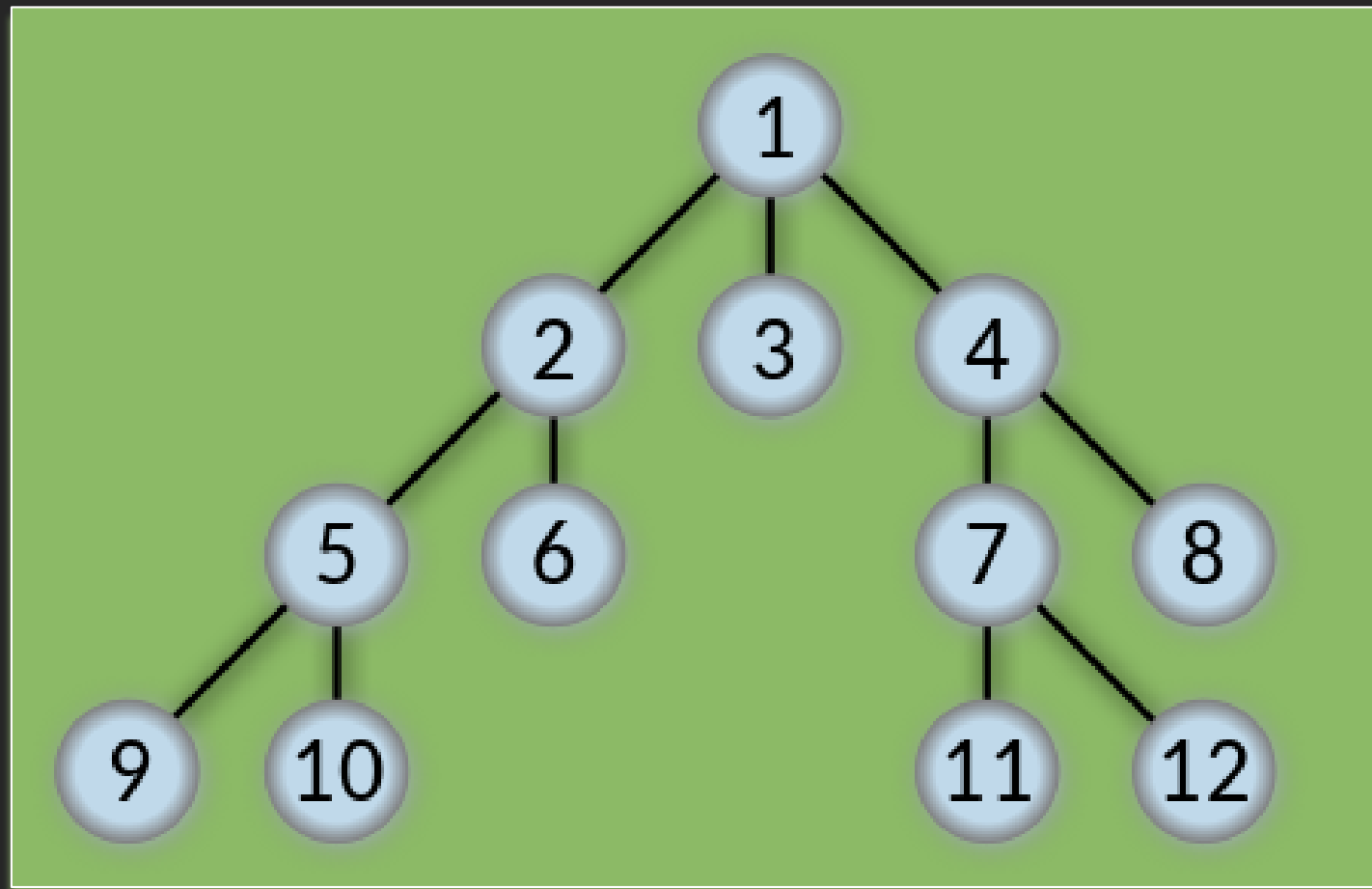
int dfs(int now)
{
    int sum=0; //從此點往下的路徑總和
    for(int i=0;i<26;i++) //從此點(now) 往下檢查 26 個字母是否相連
        if(g[now][i]!=0 && vst[now][i]==0) // 如果 now 有連到 i (而且 i沒走過)
        {
            vst[now][i]=1; // i 設為走過
            sum=max(sum,(g[now][i]+dfs(i))); // 從 i 繼續往下走，路徑的權重總和為 now->i 這一段的權重 + 從 i dfs 下去的權重和。取最大值。
            vst[now][i]=0; // i 取消走過
        }

    return sum;
}

int main()
{
    char s,u,v;
    int n,w;
    while(cin >> s >> n)
    {
        memset(g,0,sizeof(g));
        memset(vst,0,sizeof(vst)); // vst 陣列歸零
        for(int i=0;i<n;i++)
        {
            cin >> u >> v >> w;
            g[u-65][v-65]=max(g[u-65][v-65],w); // 將相鄰矩陣g[u(轉為數字)][v(轉為數字)]的值設為w，A
            轉為 0，B 為 1 ...，
        }
        cout << dfs(s-'A') << endl; // 從 s 開始走
    }
}
```

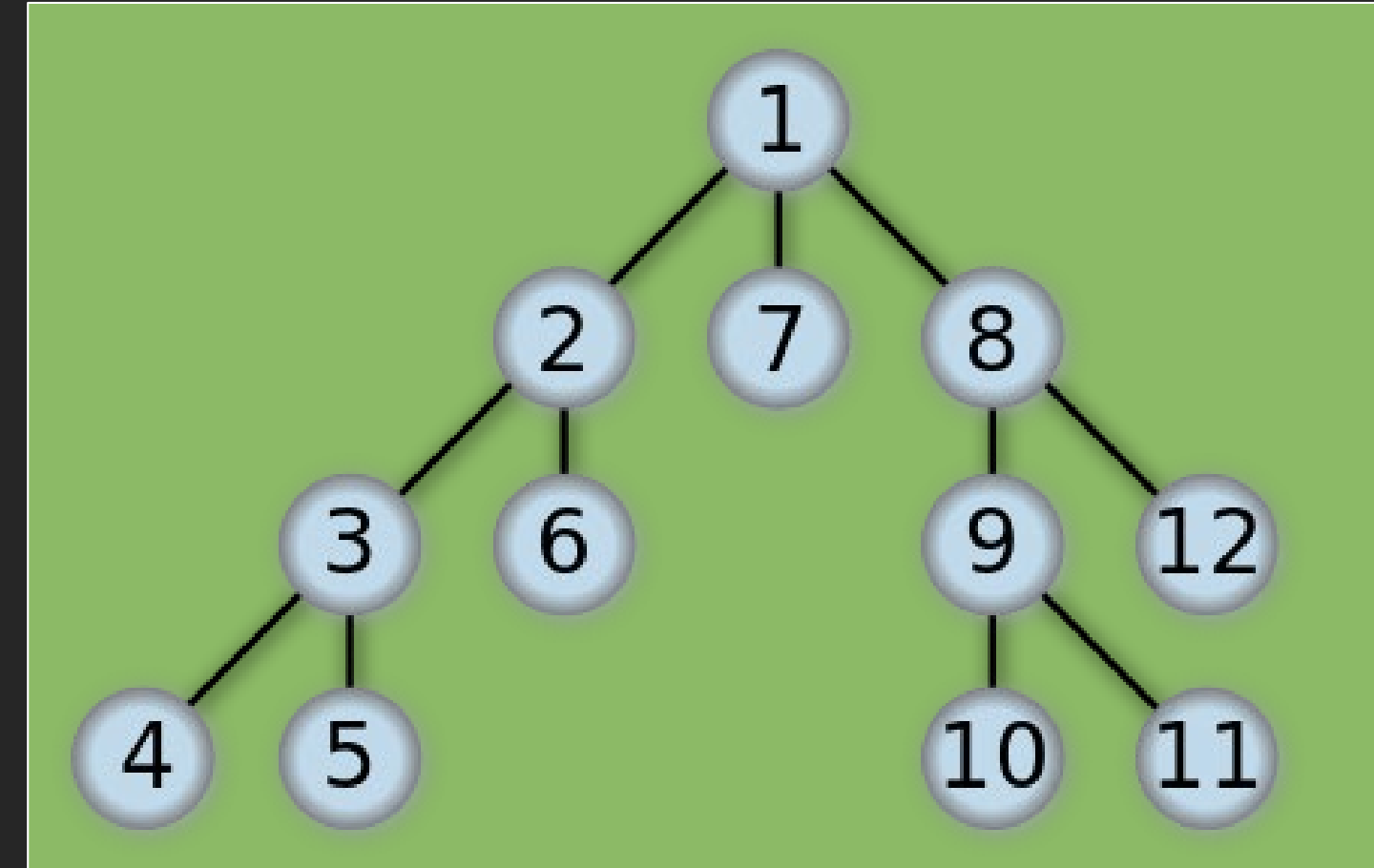
比較BFS及DFS的差異

在這兩題裡我學到了DFS及BFS這兩個演算法，這邊整理他們的差異及使用的時機



BFS演算法示例

BFS由頂點(根)開始往下溯，並且在往下的過程中會依次拜訪子樹下相同深度的節點，最後遍歷各節點完成演算法。



DFS演算法示例

DFS亦是從頂點開始，但不同於BFS的是DFS會依序拜訪同一個子樹中的各子樹，在遍歷完一條路徑後再用遞迴的方式拜訪下一個節點，如此重複最後完成遍歷。

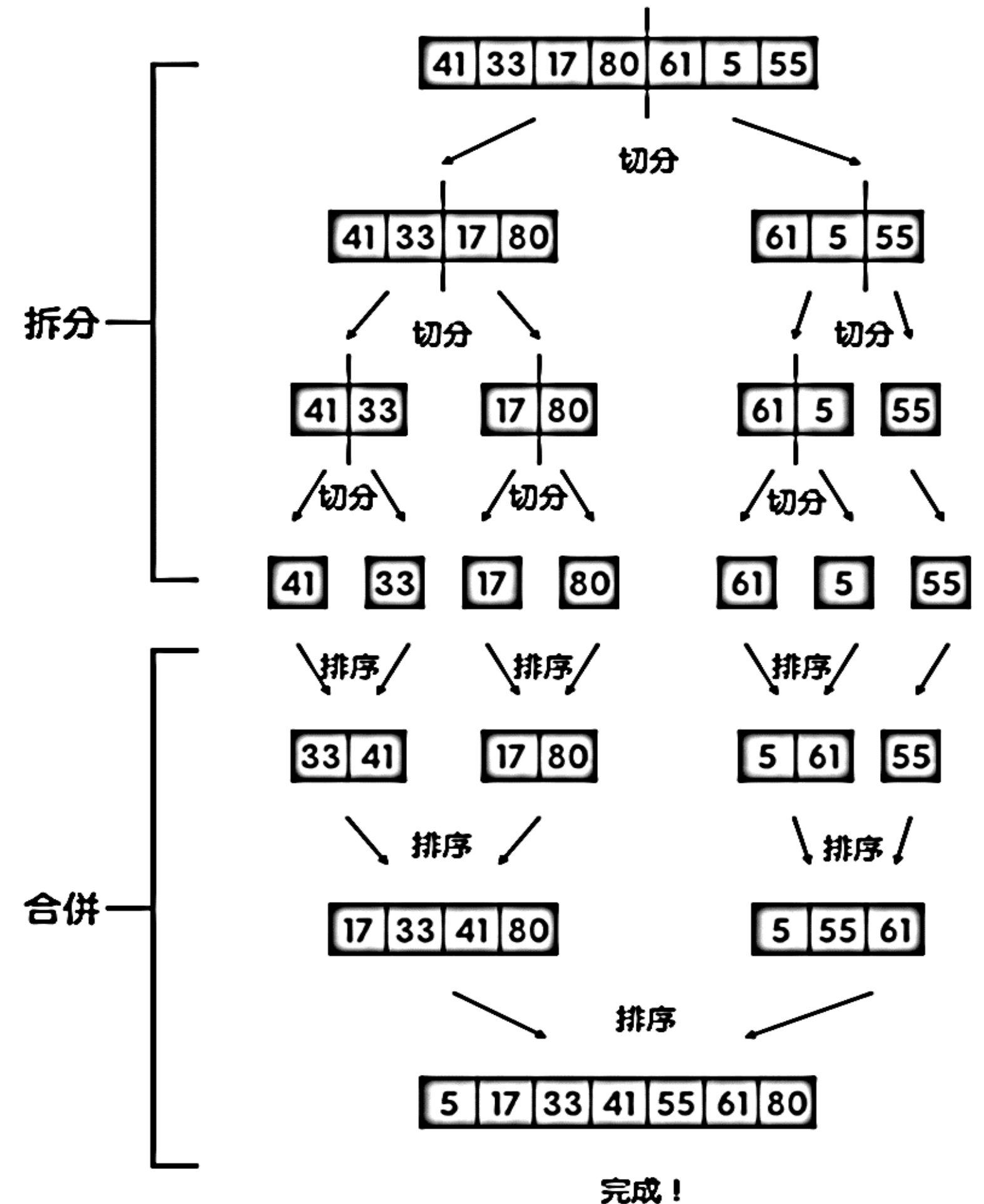
合併排序(Merge Sort)

一年級的C++基礎課程我們學到了Bubble Sort、Insertion Sort、Selection Sort，而這堂課裡我們介紹了更快的排序演算法-「合併排序演算法」

原理：

利用分治(divide & conquer)演算法將排序的過程拆成多個子問題並依序解決。

如圖，將欲排序的矩陣對半切，並且在重新組合的過程依大小排序，如此並能在最後得到排序好的矩陣，並且是遠比之前教的演算法快得多，時間複雜度為 $O(n\log n)$ 。



3) ZeroJudge d153: 六、智力測驗

題目敘述:

給定一串整數數列，求出位於中間的數字，如果數列的數目是偶數的話，處於中間的位置的分數將會有兩個，此時所求是其中較小的一個。

解題思路:

這題的想法十分簡單，單純只是排序並找中間值，但難處在於實作的部分，題目給的側資是不能用 Bubble Sort 等時間複雜度為 $O(n^2)$ 的排序演算法否則會超時，因此必需使用其他更快地演算法，例如我最近剛學會的合併演算法。

實作的部分我選擇使用陣列搭配遞迴函式實踐，作法與上一頁圖例相同(合併排序演算法)。

4) ZeroJudge f627: 1. 礦砂採集

題目敘述:

給定 n 個礦砂重量及其單位重量的價格、背包可容納最重 M 求背包能裝下的總價值最多是多少？

解題思路:

利用這一題我學到了何謂貪婪演算法，即是在每一次做抉擇時選擇當前最有利的情況，這個方法簡單、直覺，而且能利用在很多種情境。

策略:

價值與重量的比值最高的物品，優先放進背包。

總是用當下最好的物品填滿背包空隙，最後沒有留下任何空隙。每一份背包空間，都是最有價值的物品，就算是交換物品也無法增加總價值——顯然是最佳解。

時間複雜度 $O(N)$ 。 N 是物品數量。



```
#include<bits/stdc++.h>
using namespace std;

bool cmp(pair<int,int> p1 ,pair<int,int> p2)
{
    return p1.first>p2.first;
}

int main()
{
    int n,m;
    while(cin >> n >> m)
    {
        int ans=0,x=0;
        pair<int,int> p[n]; //利用pair將砂石的重量及其單位重量的價值存入陣列當中
        for(int i=0;i<n;i++)
        {
            cin >> p[i].second >> p[i].first;
            x+=p[i].second;
        }
        sort(p,p+n,cmp); //將pair由大而小排序
        int t=0; //從陣列中第一個pair開始抉擇
        while(m>0&&x>0)
        {
            if(p[t].second>0) //挑選單價最高的砂石放入背包
            {
                ans+=p[t].first;
                p[t].second--;
                m--;
                x--;
            }
            else t++;
        }
        cout << ans << endl;
    }
    return 0;
}
```

動態規劃(Dynamic Programming、DP)

最後一個學到的主題是動態規劃，這個方法能優化很多題目，將一個較大的問題定義為較小的子問題組合，先處理較小的問題並將結果儲存起來(通常使用表格)，再進一步以較小問題的解逐步建構出較大問題的解。

凡是符合以下幾個條件的問題都可以適用：

- 1.最佳子結構：最佳解可由子問題的最佳解求得。
- 2.無後效性：子問題的解一旦確定，就不會受到更大的問題的求解決策影響。
- 3.重複子問題：子問題為同樣的問題。

規劃步驟：

- 1.定義子問題(狀態定義)。
- 2.找出問題與子問題之間的遞迴關係(狀態轉移方程)。
- 3.規劃初始狀態及轉移順序，避免以遞迴的方式進行計算。(以空間換取時間)

備註：一個DP如果狀態有 $O(n^x)$ 而狀態轉移方程涉及 $O(n^y)$ 個狀態，一般可稱為 $xDyD$ 的 DP。

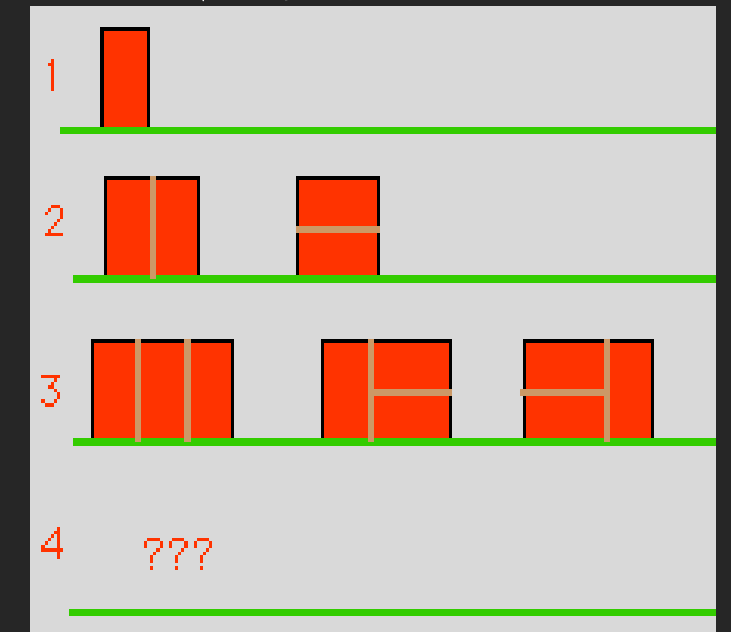
5) ZeroJudge d038: 00900 - Brick Wall Patterns

題目敘述:

如果我們要用常見的長度為高度兩倍的磚塊建一道磚牆，並且牆的高度為兩個單位，根據牆的長度，我們可以建出不同數量的花樣。從圖一我們可以看出

- 寬度為 1 單位的牆只有一種花樣——就是讓磚塊直立。
- 長度為 2 的牆有 2 種花樣——兩個平躺的磚塊疊在一起以及兩個直立的磚塊併在一起。
- 長度為 3 的牆有三種花樣。

長度為 4 的牆你可以找出幾種花樣？那長度為 5 的牆呢？



你的工作是要寫一個程式，給它牆的長度，它就算出這道牆可以有幾種花樣。

解題思路:

觀察規律可以知道這其實是遵循著費氏數列的規則，因此我們可以用很簡單的遞迴函數解決。


但是!! 有更快的方式嗎？

有的，就是屬於1D0D的動態規劃:

將每個整數對應的費氏數列函式拆解成更小的子函式，並且建表後以小問題的解逐步建構出大問題的解。

程式碼實踐

結果: AC (2ms, 336KB)



```
#include<bits/stdc++.h>
#define LY ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
using namespace std;

long long fibo(int n)
{
    long long dp[n+1];
    dp[1]=1;dp[2]=2;
    for(int i=3;i<=n;i++)
        dp[i]=dp[i-1]+dp[i-2];
    return dp[n];
}

int main()
{
    LY
    int m;
    while(cin >> m && m!=0)
        cout << fibo(m) << endl;
    return 0;
}
```


6) ZeroJudge c001:10405-Longest Common Subsequence

題目敘述:

給定一個字串，求其中最大的共通子序列長度。

解題過程:

我的想法是暴力解，但顯然會超出時間限制。

經過老師的教學後我豁然開朗，這就是2D0D的動態規劃經典題呀！

第一步是定義狀態:

- $lcs(i, j)$ 為 x 的前 i 個字元 ($x[0] \sim x[i-1]$)，與 y 的前 j 個字元 ($y[0] \sim y[j-1]$)，最長共同子序列的長度。
- x 字串的 index 由 0 開始，長度 i 的最元素為 $x[i-1]$ 。

第二步定義轉移方程:

$$lcs(i, j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ lcs(i - 1, j - 1) + 1 & \text{if } x[i] = y[j] \\ \max(lcs(i - 1, j), lcs(i, j - 1)) & \text{otherwise} \end{cases}$$

程式碼實踐

結果: AC (9ms, 4.1MB)

```
#include <iostream>
#include <cstring>
using namespace std;
int dp[1005][1005];

int main()
{
    string x,y;
    while (cin >> x >> y)
    {
        memset(dp,0,sizeof(dp));
        for(int i=1;i<=x.size();i++)           // x 的所有元素
            for(int j=1;j<=y.size();j++)       // y 的所有元素
                if(x[i-1]==y[j-1])             // x,y 最後一個元素相同
                    dp[i][j]=dp[i-1][j-1]+1;
                else                             // x,y 最後一個元素不同
                    dp[i][j]=max(dp[i-1][j],dp[i][j-1]);

        cout << dp[x.size()][y.size()] <<endl ;
    }

    return 0;
}
```



心得反思

對於這堂課我的想法及反思

在高一時我第一次接觸到程式設計這堂課也拿了不錯的成績，然而二年級開始我沒有放那麼多心在解題目、寫程式上，轉而研究其他領域的興趣。

就當我以為在課業的壓力下，或許下一次認真學習程式設計的時候已經是畢業後的暑假時，這堂進階程式設計課程設立了。

在我經歷了一個學期的學習後，我學到了更多更有趣的演算法，邏輯能力也更上了一層樓，並且我認為收穫最大的是我再次找到了寫程式的熱忱，這也讓我接下來的備考能更加積極、專注。

另外，隨著演算法難度的精進，需要的邏輯能力也不斷提升。

我常常在解題過程中腦袋一直轉不過來，但我們老師的一句我大大的改變了我的人生：

「現在想也想不起來，不如去睡覺，隔天再想就沒問題了。」

這提醒了我休息是為了走更長遠的路，不論是在寫程式或是讀書，這都是一種很值得學習的人生態度。

最後，我希望未來我能隨時保持對程式設計的熱愛，不是為了成績或是工作，而是能透過寫程式不斷的訓練我的邏輯能力及解決事情的能力，這些都將在任何地方改變我的人生。