遞迴寫法:

```
int fun(int m, int n)
{
    if (m == 0)
    {
        return n + 1;
    }
    else if (n==0)
    {
        return fun(m - 1, 1);
    }
    else
    {
        return fun(m - 1, fun(m, n - 1));
    }
}
```

左圖是以遞迴的寫法，如果M等於0直接回傳N加1，那如果N等於0會去呼叫fun這個函式，然後M減1，以上都不是的話，呼叫fun(m-1,fun(m,n-1))。

非遞迴法:

```
const int MAX_M = 3; // 可以輸入的最大值
const int MAX_N = 3000;// 顯示最大值
```

右圖為非遞迴寫法，先定義大小為**MAX_M+1 MAX_N+1**的二維陣列，通過動態規劃的方法，逐步計算並儲存 Ackermann 函數的中間結果，從而避免了深度遞迴調用帶來的效能問題。這樣的實現方式可以大大提高計算效率，特別是在處理較大輸入值時。

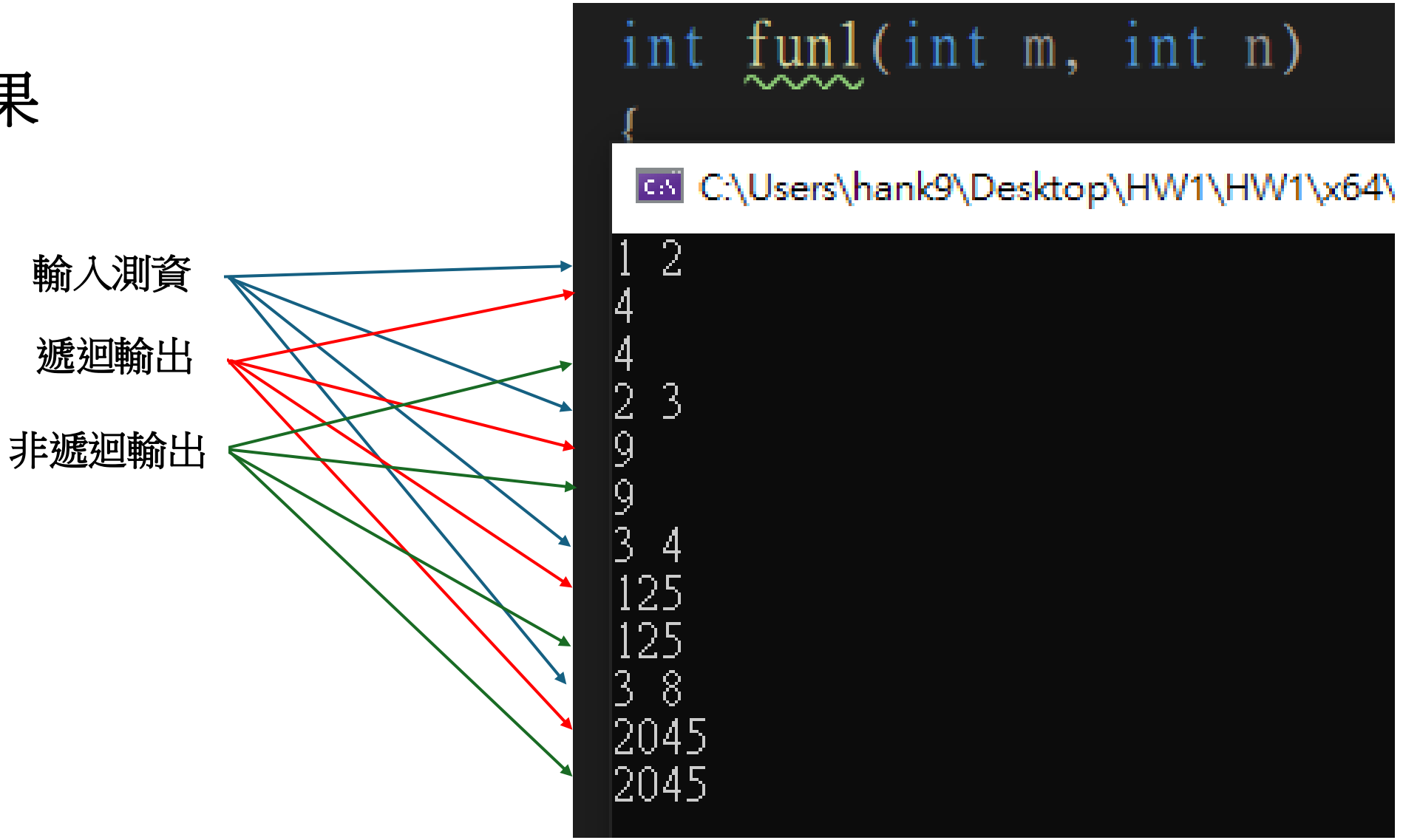
```
int fun1(int m, int n)
{
    int x[MAX_M + 1][MAX_N + 1];

    for (int i = 0; i <= MAX_M; ++i)
    {
        for (int j = 0; j <= MAX_N; ++j)
        {
            if (i == 0)
            {
                x[i][j] = j + 1;
            }
            else if (j == 0)
            {
                x[i][j] = x[i - 1][1];
            }
            else
            {
                x[i][j] = x[i - 1][x[i][j - 1]];
            }
        }
    }

    return x[m][n];
}
```

測試驗證:

右圖為測試結果



申論心得:

這次的作業讓我對遞迴與非遞迴有更深入的了解，這次花比較久的地方是以非遞迴的寫法，聽過老師的講解，以二維陣列的方向去思考，才一點一點的寫出來。
這次的作業以遞迴的做法我的測資最大為**M=3N=8**，以非遞迴的作法最大測資為**M=3N=12**。