1. **Flow Chart**

maxflow += pathflow

no more augmenting paths

found path

update residual network

no

yes

return maxflow

(Terminate Program)

do BFS

(find augmenting path)

Program Start

warehouses and stores have capacity?

read input and initialize graph

1. **Design Concept**  
   **1. Overall Design (Ford-Fulkerson Implementation)**  
    有一點要提到的是，在解典型的maximum flow的問題時，都是一個source和一個sink，但是這次的作業有好幾個store，所以為此我又多加了一個pseudo sink，然後把所有的store和這個sink相連接，然後把這些連接的edge weight capacity設為無限，這樣就等效於一個source和一個sink，同時也不影響到max flow的計算。

如上所示，從原本的graph開始，不斷地用BFS尋找augmenting path，再把找到的residual flow更新到residual network上，並且把residual flow加到maximum flow，再對這張residual network做重覆的事，一直到沒有augmenting path，回傳maximum flow就可以得到答案了。

**2. Bonus Design (Modifications to solve for capacity problem)**

原本的題目只有對edge的capacity做限制，vertices本身沒有，而若vertices有容量限制的時候，我們其實可以把它拆成兩個node來看，而這兩個node之間的edge capacity就是這個vertex的capacity，原本流入的path就從其中一個節點進入、另外一個出去，如下圖所示。這樣的修正好處就是不用重新設計BFS和Ford-Fulkerson的函式，而對原本的graph做更動就好了，其餘都是在做一樣的事。

…

Max Capacity

Max Capacity

Adjust

…

…

…

1. **Discussion or Problems in Implement Time**
2. **Time Complexity**  
    Ford-Fulkerson迴圈會在有augmenting path的情況下不斷地工作，而最壞狀況就是當每跑一次迴圈只增加了大小為1的flow，且path把每條邊都走一次，若是這樣的話，就一共要跑max\_flow次，而每一次都要跑E個邊，所以Ford-Fulkerson的time complexity為O(max\_flow\*E)。  
    但是對於找augmenting path這件事若能利用比較好的演算法，就能改善time complexity的worse case，所以在這次的作業中，我使用了Edmonds-Karp方法，也就是利用BFS去找augmenting path，這樣找到的augmenting path會在每跑一次迴圈逐漸遞增，每做一次就會拿掉一個critical path，而這樣每邊有V/2次是critical paths，另外一共有O(E)對vertices，而如上述對於每對vertices之間的邊能為critical path的次數為O(V)，所以一共要做O(VE)的累加，又BFS為O(E)，因此，Edmonds-Karp的複雜度為O(VE2)。
3. **Code Implementation**  
    除了演算法對execution time有影響之外，實際在用程式語言實現演算法的方式也會有舉足輕重的影響。在設計的一開始，我存flow graph是利用二維的adjacency matrix，但這樣其實很容易浪費memory，因為若graph的邊數不多的話矩陣很容易變成大部分為0的sparse matrix，一旦跑很大的測資就會出現像下圖的錯誤：  
    因此，後來我改良了自己的寫法，將原本大小為 (所有節點)2 的二維vector改成二維的map，只把有存在的邊存到記憶體，就可以節省許多空間，在跑回圈的時候也可以省掉很多時間因為就不用去檢查不存在的邊了。