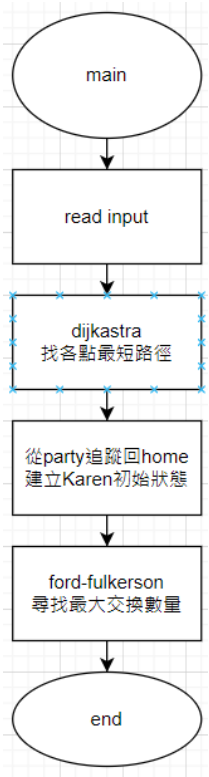
主程式流程圖&策略分析:



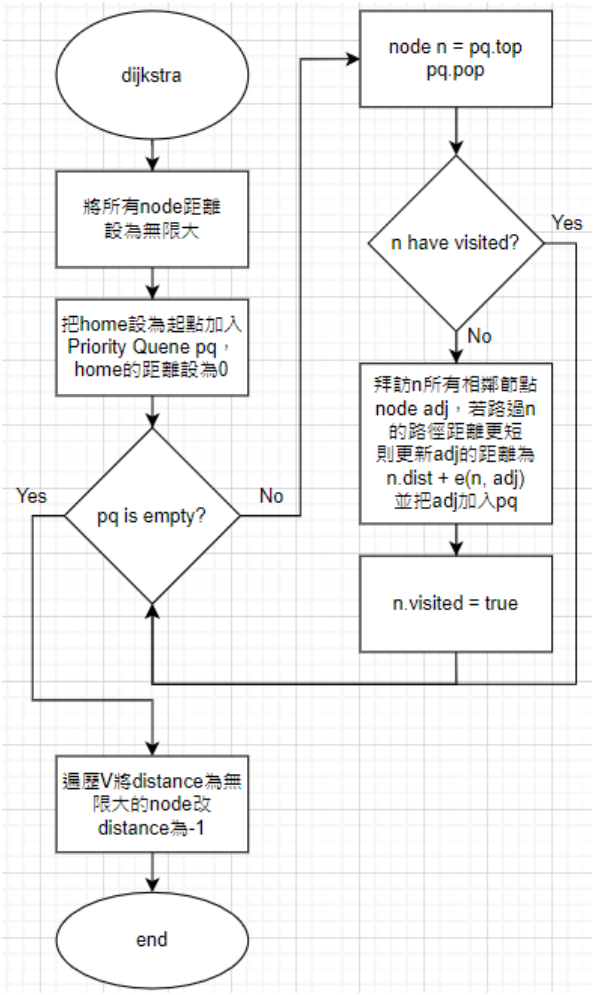
因為圖中不存在 negtive weight edge，所以使用 shortest Path 的部分使用 dijkstra 完成。

交換湯圓的部分因為任何人只要有任何一個那種口味的湯圓就不再有交換該口味的情形發生，所以任何 augmenting path flow f 都為 1，且 f 總和 F 最糟情況也小於 V 和 E，因此以 ford-fulkerson 實現為最優解。

Shortest path 流程圖&時間複雜度:

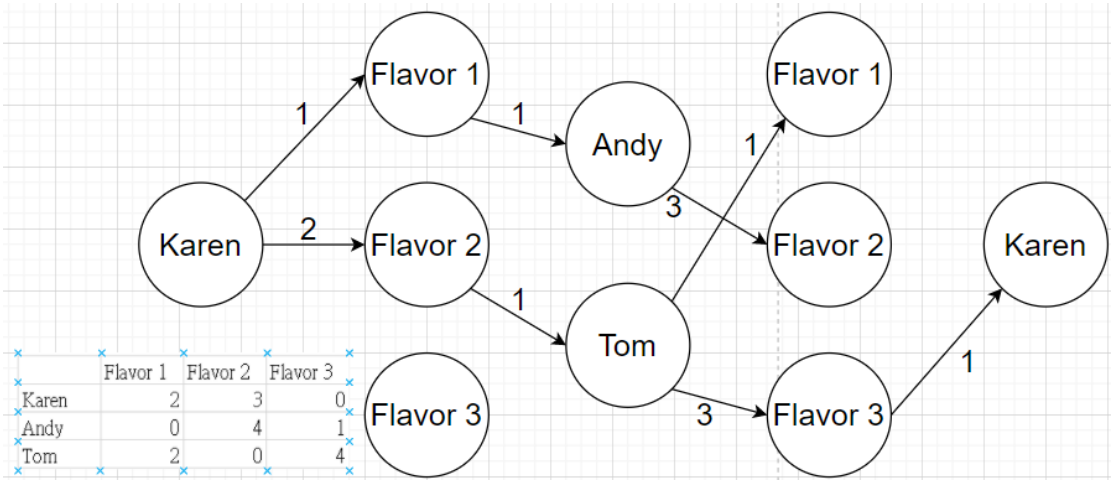
dijkstra 的時間複雜度為(V-1)*(extract_min + delete_key + decrease_key * k)，k 表示在一輪 relax 中更新 distance 的 node 數量，且 decrease_key 最多進行 E 次，若直接以描述的情形實現的話為(V – 1) * (1 + lgV + klgV) = O((V+E) * lgV)。

另一種實現方式是以直接 push relax 後的 key 到 priority quene 代替 decrease_key，因此 priority quene 中可能會有多個重複的 node，但重複的 node 會略過 relax，時間複雜度為 O((V+E) * lgV')，V' 表示 priority quene 的大小，紙面上略差於 decrease_key 的方式，但根據網路上的資訊表示實際效能會更好，因此我最後以這種方法實現。

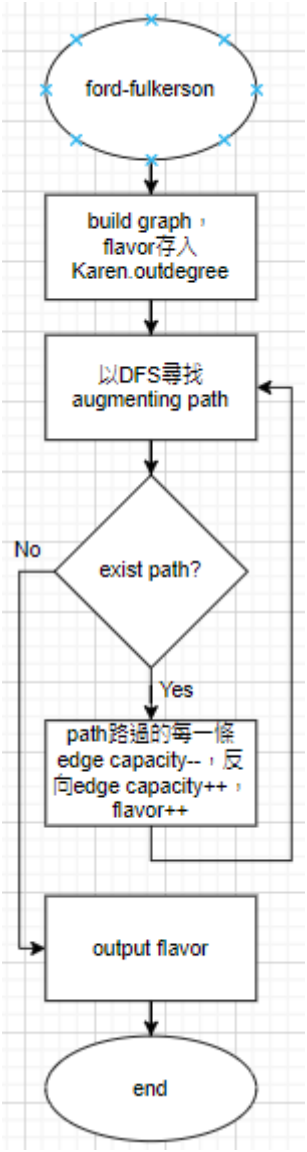


Maximum flow 流程圖&時間複雜度:

見下圖例， Karen->flavor 間若有 edge 表示 Karen 那一個口味有多的可以拿出來交換，且自己會至少留一顆所以 capacity 為該口味擁有的數量 – 1。flavor->friend 之間若有 edge 表示 friend 沒有那個口味，且因為只要有換到一顆就不再接受交換所以 capacity 為 1。friend->flavor 若有 edge 表示 friend 那種口味有多的可以換給 Karen，capacity 為該口味擁有的數量 – 1。最後 flavor->Karen 若有 edge 表示 Karen 沒有那種口味，且只要換到一顆即可，因此 capacity 為 1。



	Flavor 1	Flavor 2	Flavor 3
Karen	2	3	0
Andy	0	4	1
Tom	2	0	4



綜上所述，任何 augmenting path flow 均等於 1，因此以 DFS 尋找 augmenting path 是最有效率的，時間複雜度為 O(V+E)，當找不到 augmenting path 時表示不存在交換可能，最多換到 Karen 擁有所有口味為止，flow 總和等於 Karen 沒有的口味數量，記為 F。

friend 可能擁有多種口味都是 Karen 想要的，如果 friend 給錯 Karen 口味會使找不到 augmenting path 時並不是 Karen 擁有最多口味的狀態，所以必須在上圖中每條邊都添加反向 edge，問題等價於以 ford-fulkerson 尋找 maximum flow，時間複雜度為 O((V + E) * F)。