Homework 4

蘇俊杰、劉韋成、曾守瑜

Question1

(5.7) Write a program (or modify the code discussed in class) to recover the solution (i.e., enumerate the elements in the solution) to a knapsack problem using the *belong* flag. You should make your algorithm as efficient as possible.

Question1

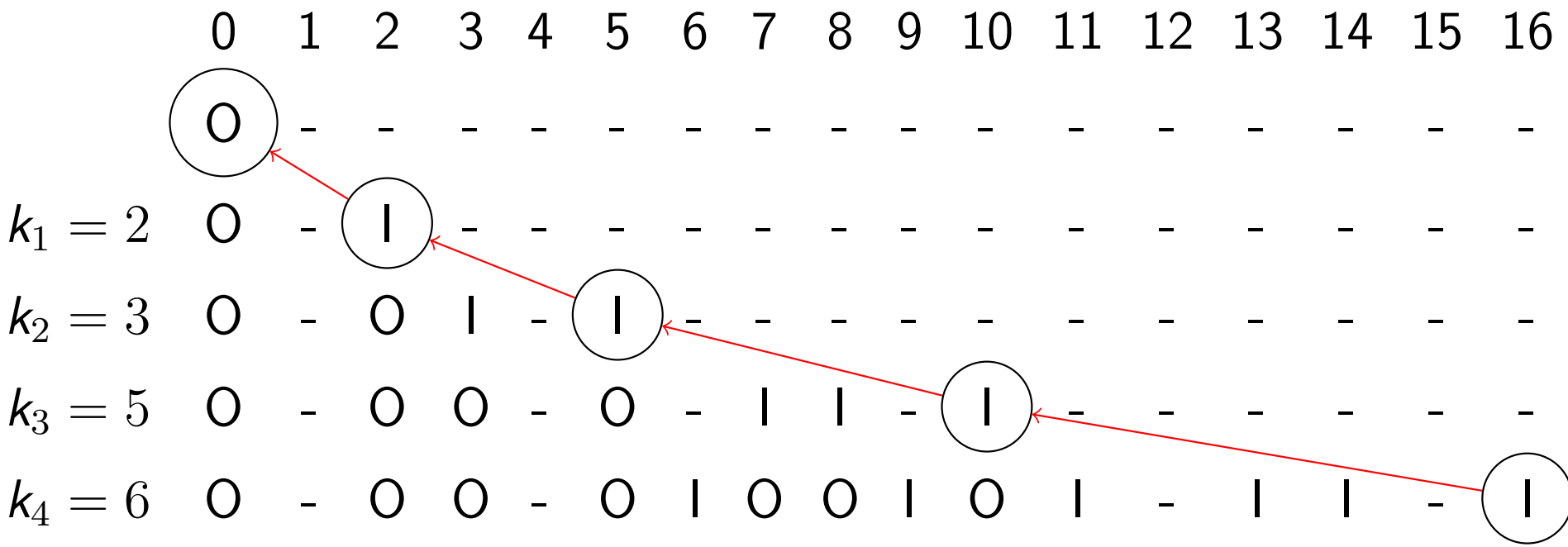
We can use the 2-dimension array P we got from the algorithm **Knapsack** to find the solution:

1. check $P[i,k].exist$, if it is *false*, return "no solution".
2. if $P[i,k].exist$ is *true*, check whether $S[i]$ is in the solution (whether $P[i,k].belong = true$).
3. if so, put $S[i]$ into the solution and next check $P[i, k - S[i]]$.
4. if not, next check $P[i-1,k]$.

Question1

```
1:  Algorithm KNAPSACK RECOVER(S, k, P);
2:      //k ≤ K
3:      solution := [];
4:      i := n;
5:      if P[i, k].exist = false then
6:          return "No such subset exists!";
7:      while k > 0 do
8:          if P[i, k].belong = true then
9:              solution.append(S[i]);
10:             k := k − S[i];
11:         i := i − 1;
12:     return solution;
```

Question1



$I = [6, 5, 3, 2]$

Question2

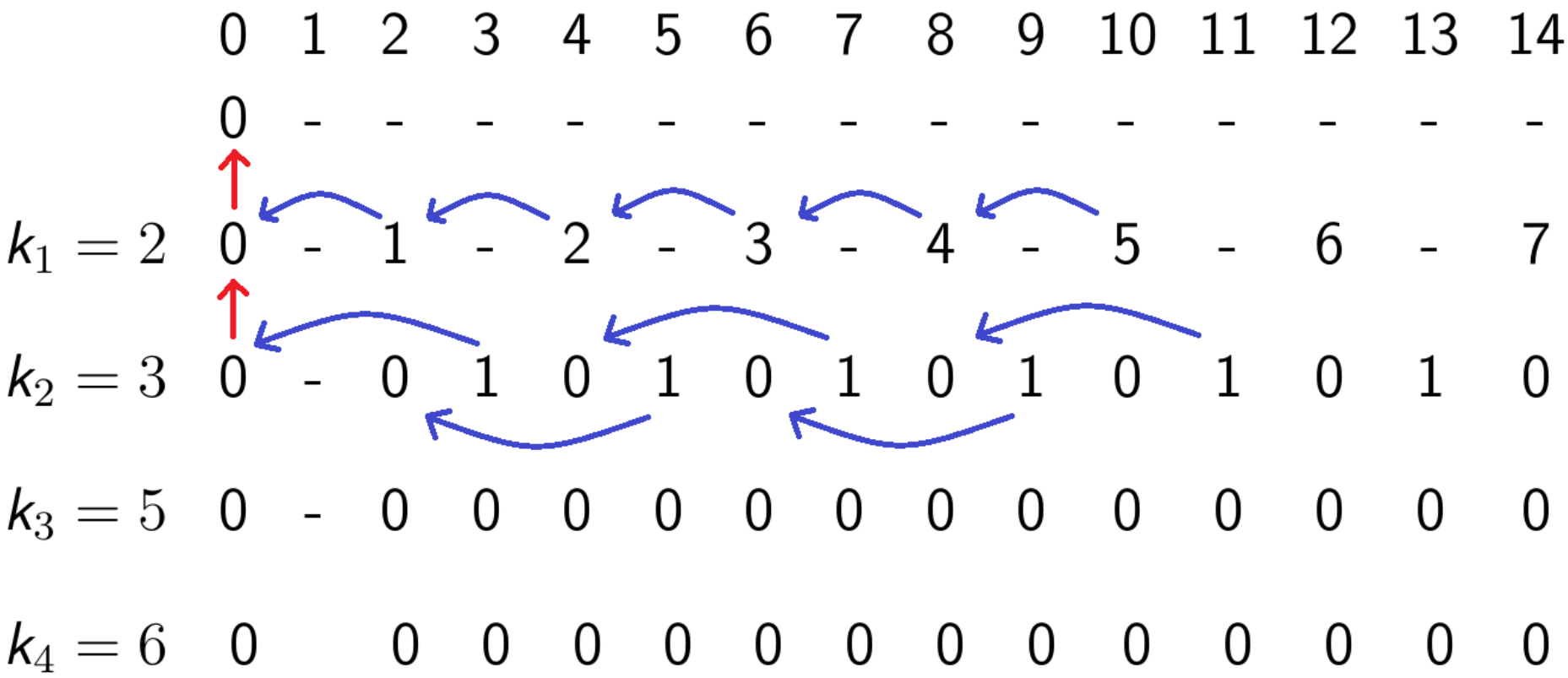
(5.17) The Knapsack Problem that we discussed in class is defined as follows: Given a set S of n items, where the ith item has an integer size $S[i]$, and an integer K, find a subset of the items whose sizes sum to exactly K or determine that no such subset exists.

We have described in class an algorithm to solve the problem. Modify the algorithm to solve a variation of the knapsack problem where each item has an *unlimited* supply. In your algorithm, please change the type of $P[i, k].belong$ into integer and use it to record the number of copies of item i needed.

Question2

```
1:  Algorithm KNAPSACK UNLIMITED(S, K);
2:      P[0, 0].exist := true;
3:      P[0, 0].belong := 0;
4:      for k := 1 to K do
5:          P[0, k].exist := false;
6:      for i := 1 to n do
7:          for k := 0 to K do
8:              P[i, k].exist := false;
9:              if P[i − 1, k].exist then
10:                 P[i, k].exist := true;
11:                 P[i, k].belong := 0;
12:             else if k − S[i] ≥ 0 then
13:                 if P[i, k − S[i]].exist then
14:                     P[i, k].exist := true;
15:                     P[i, k].belong := P[i, k − S[i]].belong + 1;
```

Question2



$I = [6, 5, 3, 2]$

Question3

(5.20) Let $x_1, x_2, \ldots, x_n$ be a set of integers, and let $S = \sum_{i=1}^{n} x_i$. Design an algorithm to partition the set into two subsets of equal sum, or determine that it is impossible to do so. The algorithm (presented in suitable pseudocode) should run in time $O(nS)$.

Question3

Let $x_1, x_2, ..., x_n$ be a set of non-negative integers, and let $S = \sum_{i=1}^{n} x_i$. Design an algorithm to partition the set into two subsets of equal sum, or determine that it is impossible to do so. The algorithm (presented in suitable pseudocode) should run in time $O(nS)$.

Question3(cont'd)

function ALGORITHM PARTITION(X)
    if S is odd **then**
        print "impossible";
    else
        $halfS := \frac{S}{2}$;
        $Knapsack(n, halfS)$;
        if $P[n, halfS].exist$ **then**
            add elements found by Problem2's algorithm to set1
            add the rest to set2
        else
            print "impossible";

Question4

(5.22) In the **towers of Hanoi** puzzle, there are three pegs A, B, and C, with n (generalizing the original eight) disks of different sizes stacked in decreasing order on peg A. The objective is to transfer all the disks on peg A to peg B, moving one disk at a time (from one peg to one of the other two) and never having a larger disk stacked upon a smaller one.

(a) Give an algorithm to solve the puzzle. Explain how induction works here.

(b) Compute the total number of moves in the algorithm. Show the details of your calculation.

Question4(a)

```
Algorithm Hanoi(n, A, B, C);
1 begin
2    if n=1 then
3        move from A to B
4    else if n>1
5        Hanoi(n-1, A, C, B);
6        move the n-th disk from A to B
7        Hanoi(n-1, C, B, A);
8 end
```

[Base case] 1 disk (move from A to B)

[Inductive step]**move n disks**

[Induction hypothesis]Moving **n-1** disks is available

1. move n-1 disks from A to C (induction hypothesis)
2. move the n-th disk to B (base case)
3. move n-1 disk from C to B (induction hypothesis)

Question4(b)

假設 M(n) 為搬動次數

$$\begin{cases} M(1) = 1 \\ M(n) = M(n-1) + 1 + M(n-1) = 2 \cdot M(n-1) + 1 \end{cases}$$

M(n)=2M(n-1)+1

$=2[2\text{M(n-2)}+1]+1$

$=2^2 \cdot \text{M(n-2)}+(2+1)$

$=2^2 \cdot [2\text{M(n-3)}+1]+(2+1)$

$=2^3 \cdot \text{M(n-3)}+(2^2+2+1)$

$\vdots$

$=2^i \cdot \text{M(n-i)}+(1+2+2^2+\ldots+2^{i-1})$

$=2^i \cdot \text{M(n-i)}+(\frac{2^i-1}{2-1})$

$\because$M(1)=1 $\therefore$i 以 (n-1) 代入

$=2^{n-1} \cdot \text{M(1)}+2^{n-1}\text{-1}$

$=2^n\text{-1}$

Question5

(5.23) Write a non-recursive program (in suitable pseudocode) that prints the moves of the solution to the towers of Hanoi puzzle.

Question5

基於之前得到的結果（遞迴版本演算法當中，移動盤子的次數），寫出不用遞迴的版本

Question5

首先，考慮到遞迴的運作就是 stack 的概念，最直接的方法就是改用 stack 實作河內塔
因為演算法 call 了兩次遞迴，所以要考慮三段情境：call 第一次遞迴之前、第一與第二次遞迴之間，與第二次遞迴結束後

Question5

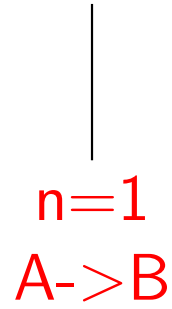
```
function HANOI(n, source, target, auxiliary)
    Stack s
    s.push({h=n, s=source, t=target, a=auxiliary, i=0})
    while not s.empty() do
        p := s.pop()
        if p.h = 1 then
            move from p.s to p.t
            continue
        if p.i = 0 then
            s.push({h=p.h, s=p.s, t=p.t, a=p.a, i=1})
            s.push({h=p.h - 1, s=p.s, t=p.a, a=p.t, i=0})
        else if p.i = 1 then
            s.push({h=p.h - 1, s=p.a, t=p.t, a=p.s, i=0})
```

Question5

利用 i 值去儲存現在是在程式執行當中的哪段情境
在河內塔演算法中，第二次遞迴結束後就不會做任何事情，所以可以忽略
還有，注意 stack 放入的順序
先放入 stack 的會比較晚執行到

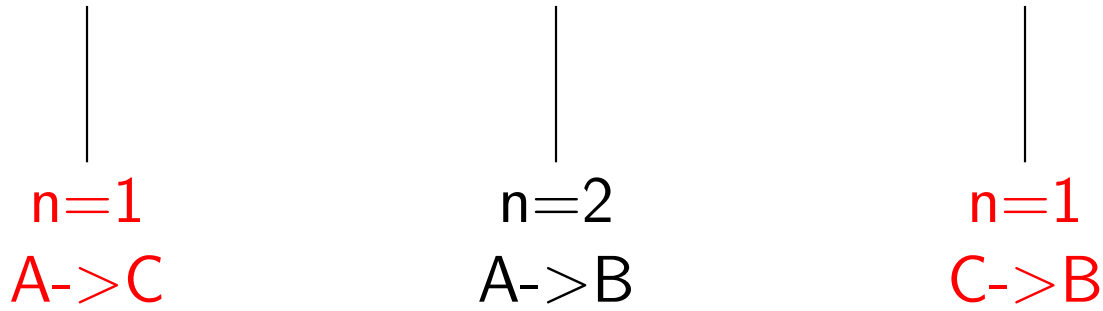
Question5

若是不想直接用 stack 轉寫遞迴呢？
先不考慮大部分同學的寫法，如果要我從零開始想起，我會怎麼思考這問題呢？
思考演算法的遞迴程序：
如果 n 為 1：



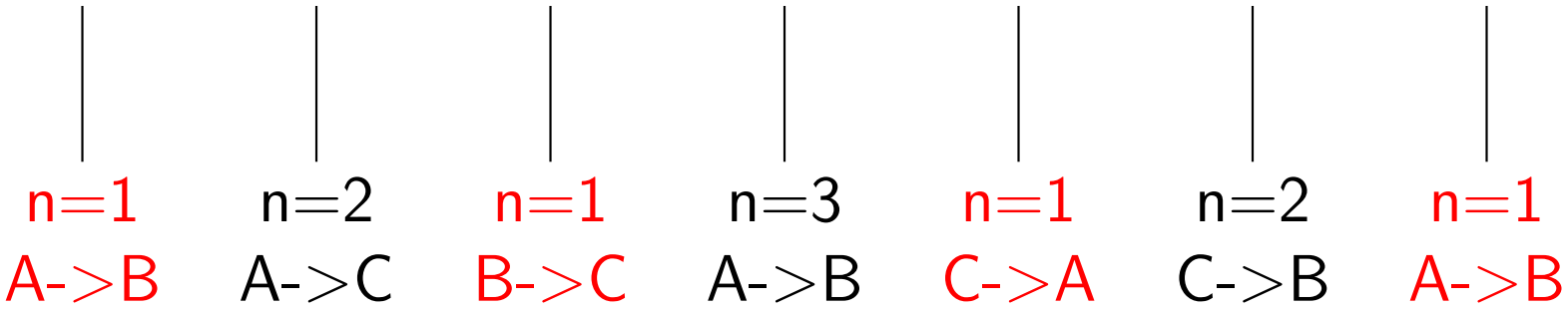
Question5

若是不想直接用 stack 轉寫遞迴呢？
先不考慮大部分同學的寫法，如果要我從零開始想起，我會怎麼思考這問題呢？
思考演算法的遞迴程序：
如果 n 為 2：



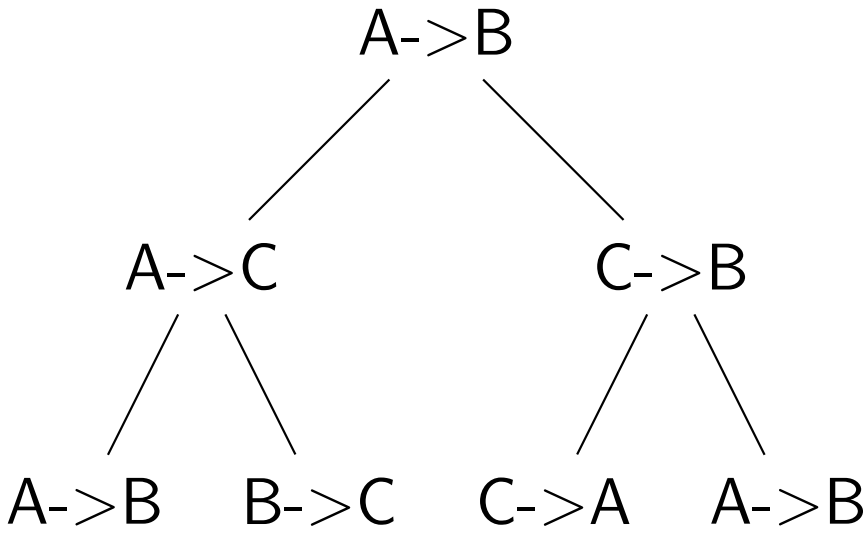
Question5

若是不想直接用 stack 轉寫遞迴呢？
先不考慮大部分同學的寫法，如果要我從零開始想起，我會怎麼思考這問題呢？
思考演算法的遞迴程序：
如果 n 為 3：



Question5

這樣的過程也可以畫成一棵樹來表示：



若是層數 $+1$，則從目前的 leaf 各自向下延伸兩個子節點
並且左子節點是從目前的起點到中繼點；右子節點是從目前的中繼點到終點
河內塔的遞迴演算可以用這樣的 full binary tree 來呈現，節點數也能得出是 $2^h - 1$，呼應前一題的答案
演算法執行步驟：對這棵樹進行 inorder traverse

Question5

演算法 1 (概念)

1. 依照 h 的大小建立一棵樹：時間複雜度 $O(2^h)$，空間複雜度 $O(2^h)$
2. 對樹進行 traverse：時間複雜度 $O(2^h)$

Question5

而事實上，我們也可以觀察到，奇數步的移動都是移最小的盤子 (對應樹的 leaf)
至於要怎麼移動？
可以觀察到，在展開子節點的過程中，相當是把往某邊移動一步拆成往另外一邊移動兩次
若 A->B 是「往右一步」，那拆成 A->C 與 C->B 就是「往左兩步」
也就是所有盤子的移動都會有固定方向，且相鄰大小的盤子移動方向相反
固定住最大盤子的移動方向，則最小盤子的移動方向取決於盤子總數

Question5

那偶數步的移動要怎麼知道是移動誰、移到哪裡呢？
想像上一步把最小的盤子移到柱 x，下一步一定是移另一個盤子
能移的盤子一定在柱子的頂頭，而且移動必須符合規則，那麼考慮以下情境：

1. 另外兩個柱子都沒盤子，偶數步卻沒有其他盤子可移，這只會發生在演算法結束時
2. 其中一個柱子 y 有盤子，另外一個柱子 z 沒有，就把這盤子從 y 移動到 z
3. 另外兩個柱子都有盤子，由於這兩個盤子一定比最小的盤子大，所以不可能是移動到柱 x
   那應該是誰移到誰？由於盤子之間一定有嚴格的大小關係，所以只會有一種移法是合法的
   若比較小的那個盤子在 y 柱，比較大的在 z 柱，那就是從 y 移動到 z

Question5

演算法 2 (概念)

1. 演算法過程中隨時儲存盤面：空間複雜度 $O(h)$
2. 若步數為奇數，移動最小的盤子 (往右或往左)
3. 若為偶數，偵測盤面上除了最小盤子之外的合法移動 (只會有一種)
4. 走到 $2^h - 1$ 步為止

儲存盤面就可以只靠「方向」的訊息就知道最小的盤子是從哪走到哪

利用 stack 就可以避免操作盤面的過程產生額外的複雜度

Question5

同學常寫的解答，利用除以 3 的餘數來判斷
問題：大部分人的寫法只能告訴使用者該在哪兩個柱子之間移動盤子
儘管合法移動的方法只會有一種，但只靠餘數無法直接告訴使用者最準確的移法，究竟是從 A 到 B 還是 B 到 A
教授認為題目要的是：一個非遞迴版本的演算法，給定同樣的輸入，它的輸出結果要和原本的版本一樣

Question5

演算法 3 (概念)

1. 演算法過程中隨時儲存盤面：空間複雜度 $O(h)$
2. 若層數為偶數，則將下面輸出的「目的地」與「經由地」互換
3. 若步數除 3 餘 0，找出目的地與經由地的合法移動
4. 若步數除 3 餘 1，找出起點與目的地的合法移動
5. 若步數除 3 餘 2，找出起點與經由地的合法移動
6. 走到 $2^h - 1$ 步為止

儲存盤面就可以精準找到「合法移動」的方法，使得輸出資訊與原版河內塔一樣

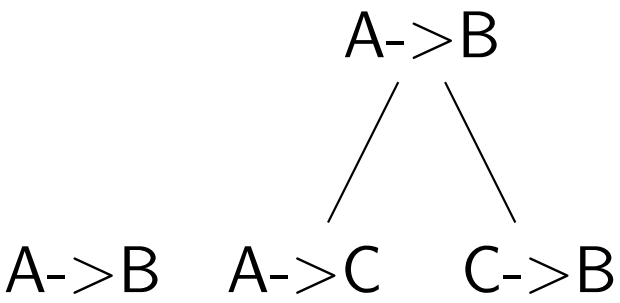
Question5

只寫了這個演算法會扣一些分，得到的分數當做演算法本身概念是對的
寫了這個演算法且有解釋為何除 3 的餘數有道理就少扣一些
寫了這個演算法，且有在函數內儲存盤面資訊者給過

函數的輸入 “A B C” 只是單純的字串，其實不該有 push pop 之類的 stack 運算，或甚至 “A > C” 之類的運算
不過寫 push/pop 就當對，寫 A > C 的請說明 A > C 是什麼

Question5

最後來講解一下為何除以 3 的餘數是對的
也就是證明 h 為奇數時，0->BC 1->AB 2->AC；而 h 為偶數時，0->BC 1->AC 2->AB
從樹狀圖下手，考慮 h=1 與 h=2 兩種 case



當 h=1 時，樹根的 A->B 是第 1 步，餘數為 1
當 h=2 時，樹根的 A->B 是第 2 步，餘數為 2；第一步 A->C 餘數 1；第三步 C->B 餘數 0

Question5

如果 h=k 時是對的，那 k+1？
已知樹會往下伸一層，這會使原本存在的 node 對應的步數乘以 2
對於原本存在的 node 而言：

1. 原本餘 0 的依然會餘 0，餘 1 與餘 2 會互換
2. 原本餘 0 對應 BC，依然對應 BC；如果原本餘 1 對應 AB，則變成對應 AC；原本餘 2 對應 AC 則變為對應 AB

這就是為何這個演算法開頭會先偵測 h 的奇偶

Question5

而對於 h=k+1 的新葉子？
若是 k 為奇數，新葉子的父節點只會是 A->B B->C C->A 當中的任一可能（與樹根同一個方向）
而根據上一頁的結論，父節點 A->B 在新樹（k+1 偶數）對應的餘數是 2
此時左子樹葉子的餘數是 1，右子樹葉子的餘數是 0（2+1）
而父親是 A->B，左子樹就會是 A->C，右子樹就會是 C->B
看得出（k+1 為偶數時）餘 1 對應 AC，餘 0 對應 BC
同理可以對新葉子的父節點為 B->C C->A 兩種狀況進行同樣的推論
證明 k 為奇數時，k+1 是對的
若 k 為偶數，證法也是同理可得