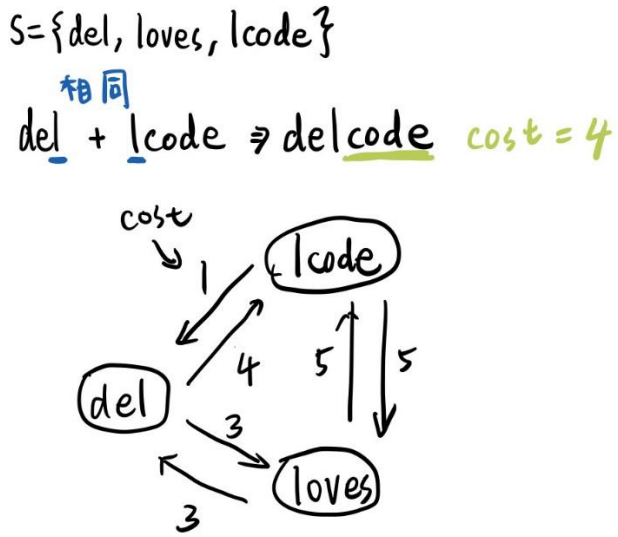
Q1. Design a greedy or DP algorithm to solve the problem.

我用了 DP algorithm 來解決這個問題。這邊會解釋一下我的 code。
首先，先建立一個大小為 size (word 的數量) 的二維 vector，**cost**，cost 會記錄把 S [i] 接到 S[j] 的 cost，例如，現在有兩個 word，{del}和{lcode}，如果把{lcode}接到{del}後面，因為它們有共同的字母 l，所以{del}其實只要接上{code}就好，這樣從{del}接到{lcode}的 cost 就會是{code}的長度 4。所以一開始的 3 個 for 迴圈就是在做這件事，倆倆比對，找出它們之間的 cost。



再來，就是做 DP，這裡會用到狀態壓縮，先把 size 左移 1 位，假設 size = 3，左移 1 位，有 size 位的二進制數就會變成 $2^3 = 8$ (這裡用 **n** 代表)，總共會有 **n-1** 種狀態。
再來建立一個大小為 n*n 的二維 vector，**dp**，並初始化為 INT_MAX/2 (代表最大 cost)，dp[state][i] 表示經過路徑 state，並且是以 i (i 是 word 編號)結尾的最小 cost，例如，state = 3 = $2^0 + 2^1$，表示訪問了 S[0] 和 S[1]，i = 2，表示以 S[2]作為結尾。還有一個大小為 n*n 的二維 vector，**path**，並初始化為 -1，path 是用來記錄 i 的前一個 word，作為最後恢復路徑所使用。

先初始化 dp，因為 S 中的每個 word 都有可能是開頭，所以要初始化各種情況，一旦使用了某個 word，要在 state 中標記對應的位置，因為目前只有一個 word，所以直接用 1<<i 來表示 state，並把 S[i] 加上對應位置上去。

之後就是要遍歷 state 的所有值，對於每一個 state 值，需要遍歷其二進制每一個為 1 的對應位的 word，變量 **cw** (CurrentWord) 從 0 遍歷到 size - 1，假如 state 的二進制數對應的第 cw 位上為 0 (可以用 AND 運算來判斷)，則說明 S[cw] 未被使用，直接跳過，接下來求出 Previous State (**prevState**)，可以用 CurrentState - (1<<cw)，把第 cw 位變 0，得到上一個狀態(沒有訪問過 CurrentWord 的狀態)，如果 prevState 為 0，代表現在是第一個 word (第一個狀態)，就把 dp[CurrentState][CurrentWord]設為 S[CurrentWord]的長度，如果 prevState 不為 0，取出 A[CurrentWord]，並讓其他所有 word 依次當作結尾 word，變量 PriviousWord (pw) 從 0 遍歷到 size – 1，依次取出所有其他的 word，並且判斷 prevState 到 PriviousWord 的 cost 加上 PriviousWord 到 CurrentWord 的 cost 有沒有小於 CurrentState 到 CurrentWord 的 cost，此狀態轉移方程可以用 ：

$$\mathbf{dp[CurrentState][CurrentWord]} =$$
$$\mathbf{min\{dp[CurrentState} - 2^i][\boldsymbol{PriviousWord}] + \boldsymbol{cost}[\boldsymbol{PriviousWord}][\boldsymbol{CurrentWord}]$$

$(\text{CurrentState} - 2^i)$表示當第 i 個 word 沒訪問過

最後要來恢復路徑，當到達最後一個狀態(代表所有都訪問過)，從 dp 裡找出最小的 cost，**min**，和最後是以哪一個 word 結尾，**last**。因為現在只有 last，所以從最後一個狀態 **s** 開始，找出上一個 word，**PrevWord**，如果 PrevWord 是最後一個 word，就把 last，整個接上 **ans**，如果 PrevWord 不是最後一個 word，就把 overlap 的部份去掉，再把 substring 取出來加上 ans，之後，因為 last 已經訪問完了，把 s 和 last 做 XOR 運算，回到上一個狀態。最後 return ans 就結束了。
P.S. 因為 bit 最多只有 32 位元，所以這個程式最多只能處理 32 個字串。

Q2. Prove the optimality of your algorithm.

Greedy 的作法是合併重疊最多的兩個 word，把得到的 word 加入原有的 word set，一直重複直到最後只剩一個 word。這個 word 就是結果。
但這並不能保證正確答案，舉個反例，有一個 set S = {vvcab, cabttt, abzzca}，用 Greedy 的話會先合併重疊最多的{vvcab}和{cabttt}，變成{vvcabttt}，再和{abzzca}合併，最後變成{vvcabtttabzzca}，長度為 14。如果用 DP 的話，會選 2 次 2 個重疊的合併，先選{ vvcab }和{ abzzca }合併，變成{vvcabzzca}，再和{cabttt}合併，最後變成{ vvcabzzcabttt }，長度為 13。
所以這題用 DP algorithm 是最優解。

Q3. Analyze the time complexity of your algorithm.

假設 n 是 word 的數量，size 是 word 的長度，計算 cost 需要 $n^2 * \text{size}$ 的時間，因為有 $2^n - 1$ 種狀態，每一狀態需要枚舉所有可能 n^2，因此遍歷 dp 需要 $(2^n - 1) * n^2$ 的時間，總時間複雜度為$O(n^2 * \text{size} + (2^n - 1) * n^2) = 2^n n^2$