1. 上海自來水來自海上

中山諸羅茶羅諸山中

乳提，這種「左 → 右」、「右 → 左」唸出來會相同的字串，小時候國文老師告訴我們它們稱作「回文」(Palindrome)，是不是美麗又絢爛呢。

今天阿穆小朋友寫了一個 C 語言函式來判斷一個字串是否為 palindrome，其程式碼片段如下。
(Note: It is guaranteed that the string `s` is always of length greater than or equal to 1.)

```
1  /*
2   * to check if a string is a palindrome
3   * return 0 for false (not a palindrome)
4   * return 1 for true (is a palindrome)
5   */
6  int is_palindrome(char* s) {
7      int index1 = 0;
8      int index2 = strlen(s) - 1;
9      while(index1 < index2) {
10         if(s[index1] != s[index2])
11             return 0;
12         index1 += 1;
13         index2 -= 1;
14     }
15     return 1;
16 }
```

請問聰明的你：

1. 若阿穆呼叫 `is_palindrome("abccba")` 則最後 `index1` 與 `index2` 之值各是？
2. 若阿穆呼叫 `is_palindrome("abcdcba")` 則最後 `index1` 與 `index2` 之值各是？
3. 你覺得第 `9` 行中的 comparison 應該要用 `<` 還是 `<=` 呢？為什麼？
4. 這個函式的 time complexity 與 space complexity 各為？

（For complexities, please answer with big-oh notation and make sure the bounds are as tight as possible.）

*如果你覺得太簡單， 下一頁有進階題, which is completely optional. ^_^*

1. (bonus)最長回文子字串

Longest Palindromic Substring

今天阿穆有點上癮了，遇到每個字串他都想要找出當中有沒有回文的部分，而且還要越長越好。

聰明的阿欣幫阿穆想到了一個又帥氣又炫砲的演算法（如下）來達到這件事情，但是阿欣其實是一個奸商，他故意留下了 3 個空格 ((a), (b), (c)) 來跟阿穆收錢，每格竟然還要 100 塊台幣！

變數用途與解釋

`s`: the string that 阿穆 encountered
`n`: the length of the string `s`
`z[]`: integer array, and `z[i]` is the length of the longest palindromic substring centered at `s[i]`, i.e., the LPS is: `s[(i-z[i]/2)...(i+z[i]/2)]`.

演算法流程

初始 `z[0] = 1` 後, 依序計算 `z[i], i = 1 … n-1`
計算 `z[i]`:

Find an index `j`, `0 ≤ j < i`, that maximizes `(j + z[j] / 2)`
Check if `i <= ___(a)___`:

(1) No => `z[i] = do_sym_search(s[i], s[i])`

(2) Yes

(a) Get index `i_sym` where the position `i` and `i_sym` are symmetric w.r.t. `j`

(b) Denote the left end of the LPS centered at `i_sym` as `l_sym`

(i) If `l_sym < j - z[j] / 2`:
`z[i] = ___(b)___`

(ii) If `l_sym == j - z[j] / 2`:

```
z[i] = z[i_sym] + do_sym_search(
        s[i - z[i_sym] / 2 - 1],
        s[i + z[i_sym] / 2 + 1]
)
```

(iii) If `l_sym > j - z[j] / 2`:
`z[i] = ___(c)___`

身為一個更專業的商人，你馬上就知道，聰明的你瞬間就可以想到這 3 格的答案，而且阿穆想要用每題 60 塊的價格跟你買。你今天想要賺錢了嗎？

1. Please fill directly in the blanks above.
2. 請問你覺得這個演算法有什麼樣的問題？(hint: 前一頁的 1. 與 2. 兩小題)
3. 你覺得這個問題可以怎麼解決？

解答

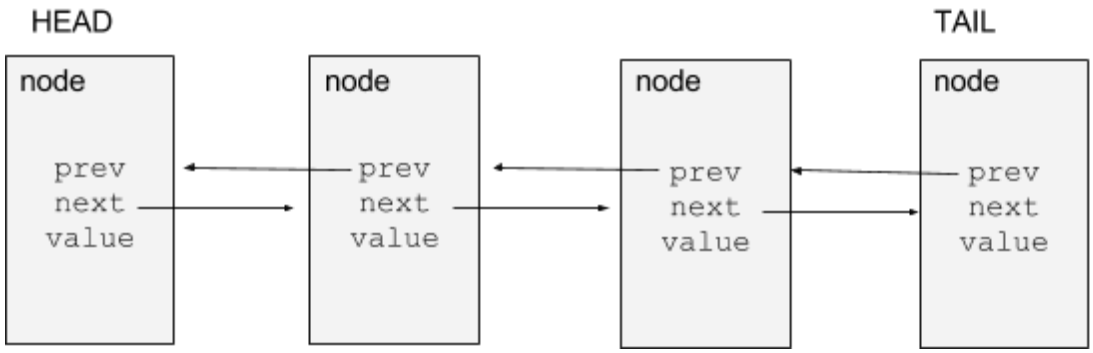
1. 上海自來水來自海上

1. 3; 2
2. 3; 3
3. 都是對的，因為 < 與 <= 差在字串長度為奇數的時候有沒有檢查最中間的字元，但其實檢查回文不需要檢查最中間的字元。
4. Time: O(n); Space: O(1), where n is the length of the given string.

1. (bonus) 最長回文子字串

1. 如下
    a. j + z[j] / 2
    b. ((j + z[j] / 2) - i) * 2 + 1
    c. z[i_sym]
2. 此演算法無法檢查到長度為偶數的回文子字串, i.e., 只會得到長度為奇數的最長回文子字串
3. 利用插入點點點 （abcddcbf => a.b.c.d.d.c.b.f）

2. 雙向Linked-list




```
struct node{
    int value;
    struct node* prev;
    struct node* next;
}HEAD, TAIL;

void Insert(struct node* a, int v){
    struct node* b        = a->next;
    struct node* new_node = malloc(sizeof(struct node));

    new_node->value = v;

    a->next = (1);
    b->prev = (2);

    new_node->prev = (3);
    new_node->next = (4);
}
void Delete(struct node* a){
    struct node* b = a->prev;
    struct node* c = a->next;
    b->(5) = (6);
    c->(7) = (8);
    free(a);
}
```

上面是一個雙向linked-list的實作，每一個節點都記錄了節點的數值(value)、前一個節點(prev)與下一個節點(next)。
Insert函式實作的是新增一個value=v的節點，然後將節點插入在節點a的後面；Delete函式會將指定的節點a刪除。
請在(1)~(8)空格中填入以下關鍵字完成程式：a,b,c,new_node,prev,next(每個關鍵字可能被填入0次或1次以上)。

解答

2. 雙向Linked-list

(1) new_node
(2) new_node
(3) a
(4) b
(5) next
(6) c
(7) prev
(8) b

```
void Insert(struct node* a, int v){
    struct node* b        = a->next;
    struct node* new_node = malloc(sizeof(struct node));

    new_node->value = v;

    a->next = new_node;
    b->prev = new_node;

    new_node->prev = a;
    new_node->next = b;
}
void Delete(struct node* a){
    struct node* b = a->prev;
    struct node* c = a->next;
    b->next = c;
    c->prev = b;
    free(a);
}
```

3. Linked Sublist

假設 L1, L2 是兩個（未必相異的）linked lists. 我們定義 **L1 是 L2 的一個 linked sublist** 的條件是以下任一成立：

- L1 is NULL
- L1->value == L2->value 並且 L1->next 是 L2->next 的一個 linked sublist
- L1 是 L2->next 的一個 linked sublist

例如：以下的 L1 就是 L2 的一個 linked sublist，但 L1 不是 L3 的一個 linked sublist。

L1 = 9 -> 4
L2 = 9 -> 4 -> 8 -> 7 -> 9 -> 4 -> 0
L3 = 4 -> 2 -> 8 -> 9

照上面的定義，我們可以直接寫出一個 (recursive) function 確認 L1 是不是 L2 的一個 linked sublist。當然事情不會這麼簡單！我們想要知道 L1 能**以幾種方式**作為 L2 的一個 linked sublist。

例如：按照以上的定義，L1 能以 3 種方式作為 L2 的一個 linked sublist，如下所示：

L2 = **9** -> **4** -> 8 -> 7 -> 9 -> 4 -> 0
= 9 -> 4 -> 8 -> 7 -> **9** -> **4** -> 0
= **9** -> 4 -> 8 -> 7 -> 9 -> **4** -> 0

以下這個 function 的大部分已經寫好了，請把挖空的五個括號處補完吧！

```
int sublist(Node* L1, Node* L2) {
  if (          ) return 1;
  int ans = 0;
  if (          ) {
    ans += sublist(            );
    if (                      ) ans += sublist(
);
  }
  return ans;
}
```

解答

3. Linked Sublist

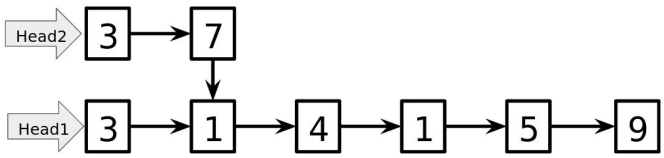
```
int sublist(Node* L1, Node* L2) {
  if (L1 == NULL) return 1;
  int ans = 0;
  if (L2 != NULL) {
    ans += sublist(L1, L2->next);
    if (L1->value == L2->value)
      ans += sublist(L1->next, L2->next);
  }
  return ans;
}
```

4. Copy on write linked list

來練習 Singly linked list 的複製(copy)。如果要複製整個 linked list 給某個函數，而這個函數只需要讀取 linked list 當中的內容，不需要修改。那麼，單純的把 head 傳過去似乎就解決問題了！

但如果是要複製出一個內容有一點點小差異的 linked list 呢？
例如原有一個 linked list，從 head 依序讀取會得到序列 <3, 1, 4, 1, 5, 9>，接著要複製它並在第一個元素後面加入數字 7，產生出額外的新序列<3, 7, 1, 4, 1, 5, 9>，並且讓我們同時保有兩個 linked list 能夠分別讀取到對應的序列。（只要求能順利讀取）

那麼其實如下的結構可以辦到這件事：



Head1 會得到 <3, 1, 4, 1, 5, 9>
Head2 會得到 <3, 7, 1, 4, 1, 5, 9>

```
struct node {
        int value;
        struct node* next;
};
(1) struct node Node;

void print(Node* head) {
        while(head != NULL) {
                printf("%d\n", head -> value);
                head = head -> next;
        }
}

Node* f(Node* head, int k, int a) {
        Node* new_head = (Node*)malloc(sizeof(Node));
        Node* working = new_head;
        for(int i = 0; i < (2); i++) {
                working -> value = (3);
                working -> next = (Node*)malloc(sizeof(Node));
                head = (4);
                working = (5);
        }
        working -> value = a;
        working -> next = head;
        return (6);
}
```

上面的程式碼中，print(head) 能把整個 linked list 的值依序輸出，而 f(head, k, a) 能複製一個 head 所指到的 linked list、在第 k 個數字後面加入一個數字 a（head 所指到的是第 1 個，k=0 表示在最開頭加入數字，k 不會小於 0 或者大於傳入的 linked list的長度），最後回傳所產生的 linked list 的開頭（傳入的 head 所指到的 linked list 之後還是要有效）。

1. 請完成上面的 6 個空格。
2. 請說明這個作法呼叫一次函式 f 的時間和空間複雜度（假設 head 所指的 linked list 長度為 n，操作為在第 k 個數值後面加入數值 a）。

解答

4. Copy on write linked list

1.

```
struct node {
        int value;
        struct node* next;
};
(typedef) struct node Node;

void print(Node* head) {
        while(head != NULL) {
                printf("%d\n", head -> value);
                head = head -> next;
        }
}

Node* f(Node* head, int k, int a) {
        Node* new_head = (Node*)malloc(sizeof(Node));
        Node* working = new_head;
        for(int i = 0; i < (k); i++) {
                working -> value = (head -> value);
                working -> next = (Node*)malloc(sizeof(Node));
                head = (head -> next);
                working = (working -> next);
        }
        working -> value = a;
        working -> next = head;
        return (new_head);
}
```

2.

時間和空間都是 θ(k)

5. Almost Same String Matching

Description

Eddy likes to play with strings. Especially, he likes to find some identical strings. However, finding two exactly same strings isn't easy. Thus, Eddy can accept some strings which are almost the same.
Exactly speaking, string A and string B are almost the same if their lengths are equal (|A|=|B|) and number of different position is less than or equal to p/q of their length (|A|*p/q). Number of different position is the number of i s.t. i-th character of A and i-th character of B are different.
Now, Eddy has two strings S and T. He wants to find the number of substrings of S which is almost the same with T. But, it's too difficult to Eddy. Thus, Let's help poor Eddy and find the answer for him!

Input format

The first line contains one positive integer C indicating that Eddy will be interested to C pairs of strings.
The following C lines each contains two strings S, T and two integers p, q separated by a space. That is, Eddy wants to find the number of substrings of S which is almost the same with T.

Input constraint

- S and T contain only lowercase English letters (a~z)
- 0 <= p <= 10^4, 1 <= q <= 10^4
- Total length of S and T among C pairs of strings is at most 10^4

Output format

For each pair of strings, you should output an integer in one line indicating the number of substrings of S which is almost the same with T.

Sample input

2
aaa a 1 1
abb ab 1 2

Sample output

3
2

解答

5. Almost Same String Matching

Iterate through all possible substring of S(which length equal to the length of T), count the number of different position, then compare with p/q.

```
#define N 10210
char s[ N ] , t[ N ];
int p , q;
void solve(){
  int cc = 0;
  int lens = strlen( s );
  int lent = strlen( t );
  for( int i = 0 ; i <= lens - lent ; i ++ ){
    int dif = 0;
    for( int j = 0 ; j < lent ; j ++ )
      if( s[ i + j ] != t[ j ] )
        dif ++;
    if( dif * q <= p * lent )
      cc ++;
  }
  printf( "%d\n" , cc );
}
int main(){
  int c; scanf( "%d" , &c ); while( c -- ){
    scanf( "%s%s%d%d" , s , t , &p , &q );
    solve();
  }
```