DSA HW#3

Problem 1

Subproblem 1:

Ans：$1-\frac{n^2!}{n^{2n}*(n^2-n)!}$
所求=1-完全沒collision
完全沒collision = $\frac{n^2*(n^2-1)...*(n^2-n+1)}{n^{2n}}=\frac{n^2!}{n^{2n}*(n^2-n)!}$

Subproblem 2:

Subproblem 3:

(1) :

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
							18			
	34						18			
	34						18		9	
	34			37			18		9	
	34			37			18	40	9	
	34			37			18	40	9	32
	34	89		37			18	40	9	32

(2) :

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
							18			
	34						18			
	34						18		9	
	34			37			18		9	
	34			37			18	40	9	
	34			37			18	40	9	32
89	34			37			18	40	9	32

Subproblem 4:

Table 1

0	**1**	**2**	**3**	**4**	**5**	**6**
						6
			31			6
		2	31			6
		2	31			41
		30	31			6
		30	45			6
		44	31			6

Table 2

0	**1**	**2**	**3**	**4**	**5**	**6**
						-
						-
						-
6						
2					41	
2				31	41	
2				30	41	45

Problem 2

Subproblem 1:

```
ans[Q] = {0}
for j from 1 to Q:
    interval = l2-l1
    for i from l1 to l1 +n-1:
        if S[i] != S[i+interval]:
            ans[j] = False
    ans[j] = True
```

最糟狀況下會把搜尋的子字串全部跑過一次，長度為n
接著又有Q次查詢，所以最糟時間複雜度是$O(Qn)$
額外使用的空間只有儲存變數$interval$的而已和作為答案的ans陣列，所以是$O(Q)$
$Time\ complexity : O(QN)$
$Space\ complexity : O(Q)$

Subproblem 2:

$X = \{8, 0, 0, 0, 3, 0, 0, 0\}$

Subproblem 3:

ref :
https://wangwilly.github.io/willywangkaa/2018/03/19/Algorithm-Z-%E6%BC%94%E7%AE%97%E6%B3%95/
https://kenjichao.gitbooks.io/algorithm/content/z_algorithm.html

```
Calculate_X(S,X)
int L = 0
X[0] = 0
for i from 1 to N:
    if i>L+X[L]:
        X[i] = 0
    else:
        X[i] = min(Z[L]+L-i, X[i-L])
    while(S[i]==S[i+X[i]]):
        X[i]++
    if X[i]+i >X[L]+L:
        L = i
```

最糟情況下，內層迴圈總共會跑n次
外層迴圈固定是n次
因此時間複雜度為 $O(2n) = O(n)$
額外使用的空間只有儲存變數L，因此為 $O(1)$
若把X的空間計入則為$O(n)$
$Time\ complexity : O(n)$
$Space\ complexity : O(n)$

Subproblem 4:

```
len_p = len(p)
S[1...p] = p[1...p]
S[p+1] = '#'
S[p+2...p+N] = t[1...N]
X[1...N+p] = {0}

ans = 0
Calculate_X(S,X)
ans = 0
for i from p+2 to N+p:
    if X[i] == len_p:
        ans++
return ans
```

令p為字串p的長度
計算X陣列時間複雜度為$O(N+p)$
然後遍歷X的次數為N
因此時間複雜度為 $O(N+p)$
額外使用的空間只有儲存變數，因此為 $O(1)$
若把X的空間計入則為$O(N)$
$Time\ complexity : O(N+p)$
$Space\ complexity : O(N)$

Problem 3

Subproblem 1:

```
INIT(n)
    for i from 0 to n-1:
        set[i] = NULL
    No = 0

ADD-EDGE(x, y)
    x_set = -1
    y_set = -1
    if set[x]==NULL:
        MAKE-SET(x)
    else
        x_set = FIND-SET(x)
    if set[y]==NULL:
        MAKE-SET(y)
    else
        y_set = FIND-SET(y)

    if x_set == y_set and x_set!=-1:
        No++
    UNION(x, y)

IS-BIPARTITE():
    if No>0
        return false
    return True
```

每次增加邊的時候都呼叫$FIND-SET$確認是不是同個SET連線
如果是就紀錄下來，等到$IS-BIPARTITE()$時輸出

Subproblem 2:

```
INIT(n)
    for i from 0 to n-1:
        set[i] = NULL
    for i from 0 to 2:
        winlist[i][0] = -1
        winlist[i][1] = -1
    No = 0

WIN(x, y)
    if set[x]==NULL:
```

```
        MAKE-SET(x)
    if set[y]==NULL:
        MAKE-SET(y)

    x_set = FIND-SET(x)
    y_set = FIND-SET(y)
    In = 0
    for i from 0 to 2:
        if winlist[i][0] == x_set:
            if winlist[i][1] != y_set:
                No++
            In++
    if In == 0:
        for i from 0 to 2:
            if winlist[i][0] != -1:
                winlist[i][0] = x_set
                winlist[i][1] = y_set

TIE(x, y):
    if set[x]==NULL:
        MAKE-SET(x)
    if set[y]==NULL:
        MAKE-SET(y)
    x_set = FIND-SET(x)
    y_set = FIND-SET(y)
    for i from 0 to 2:
        if winlist[i][0] == x_set and winlist[i][1] == y_set:
            No++
        if winlist[i][0] == y_set and winlist[i][1] == x_set:
            No++
    UNION(x, y)

IS-CONTRADICT():
    check[6] = {0}

    if No>0
        return false
    return True
```

創建一個勝利紀錄表紀錄勝負關係，每次WIN和TIE時都利用這個表來確認有沒有出問題。

Subproblem 3:

ANS：否
只有$Path\ compression$不會改變$Find-Set$的時間複雜度(From lecture Disjoint Set)
Ex：

```
n = 1000000000000000 //any number
init(&djs, n);
add_edge(&djs, 2, 1);
add_edge(&djs, 3, 2);
add_edge(&djs, 4, 3);
.
.
.
add_edge(&djs, n-2, n-3);
add_edge(&djs, n-1, n-2);
add_edge(&djs, n, n-1);
add_edge(&djs, n, 1);
```

Subproblem 4:

$ANS : TRUE$
最開始會先呼叫$init$初始化，初始化時會跑回圈把$disjoint\ set$裡每個元素都先指到自己，所以是$O(N)$

接著考慮某個$element\ X$，最初他只有自己一個$element$
當要$union$時，因為有$union\ by\ size$所以X所在的set必會變成一個比原本大上兩倍以上的set
當X所在的set被$union$了M次後，而最下面的$element$要$FIND-SET$時最多得走上M次
令N為set中元素的數目
可以得知$N > 2^M \quad \therefore lg(N) > M$
所以時間複雜度為$O(lg(n))$
因此$FIND-SET$的時間複雜度可以被壓縮到$O(log(n))$

$SHOW-CC()$的複雜度為$O(1)$

$ADD-EDGE(x,y)$時會對$stack$做操作，時間複雜度為$O(1)$
接著在$djs\ union$時會呼叫兩次$FIND-SET$，時間複雜度為$O(log(n))$
因此$ADD-EDGE(x,y)$的時間複雜度為$O(log(n))$

$UNDO()$時對$stack$的操作也是$O(1)$

操作共有M次，因此時間複雜度為$O(N+Mlg(N)$

Subproblem 5: