Homework 6

蘇俊杰、劉韋成、曾守瑜

Problem1

(6.62) You are asked to design a schedule for a round-robin tennis tournament. There are $n = 2^k$ $(k \geq 1)$ players. Each player must play every other player, and each player must paly one match per round for $n-1$ rounds. Denote the players by $P_1, P_2, \ldots, P_n$. Output the schedule for each player. (Hint: use divide and conquer in the following way. First, divide the players into two equal groups and let them play within the groups for the first $\frac{n}{2} - 1$ rounds. Then, design the games between the groups for the other $\frac{n}{2}$ rounds.)

Problem1

```
1:  Algorithm TOURNAMENT(n);
2:  // The number of players n = 2^k for some k ≥ 1.
3:  // O[r, i] = j indicates that in Round r (1 ≤ r ≤ n − 1) the opponent of P_i is P_j.
4:      ROUND-ROBIN SCHEDULE(1, n, n − 1);
5:
6:  Algorithm ROUND-ROBIN SCHEDULE(L, R, r);
7:      if R − L = 1 then
8:          O[L, r], O[R, r] := R, L;
9:      else
10:         M := (L + R)/2;
11:         ROUND-ROBIN SCHEDULE(L, M, r/2);
12:         ROUND-ROBIN SCHEDULE(M + 1, R, r/2);
13:     for round_num from r/2 + 1 to r do
14:         for shift from 0 to (M − L) do
15:             P1 := L + shift;
16:             P2 := M + 1 + [(round_num + shift)%(r/2 + 1)];
17:             O[P1, round_num] := P2;
18:             O[P2, round_num] := P1;
```

Problem2

Consider the solutions to the union-find problem discussed in class. Suppose we start with a collection of ten elements: A, B, C, D, E, F, G, H, I, and J.

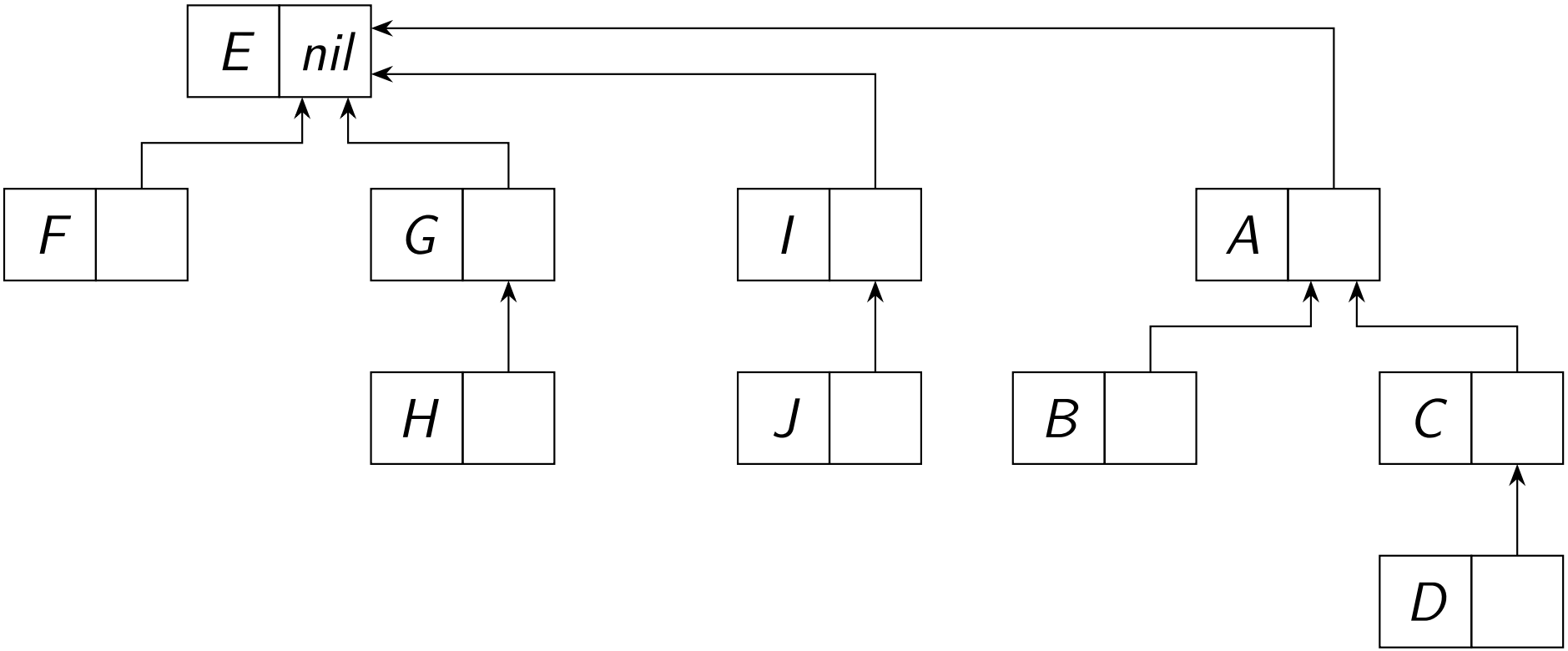
(a) Assuming the balancing, but not path compression, technique is used, draw a diagram showing the grouping of these ten elements after the following operations (in the order listed) are completed:

i. union(A,B)
ii. union(C,D)
iii. union(A,D)
iv. union(E,F)
v. union(G,H)
vi. union(F,G)
vii. union(I,J)
viii. union(H,I)
ix. union(D,J)

In the case of combining two groups of the same size, please always point the second group to the first.

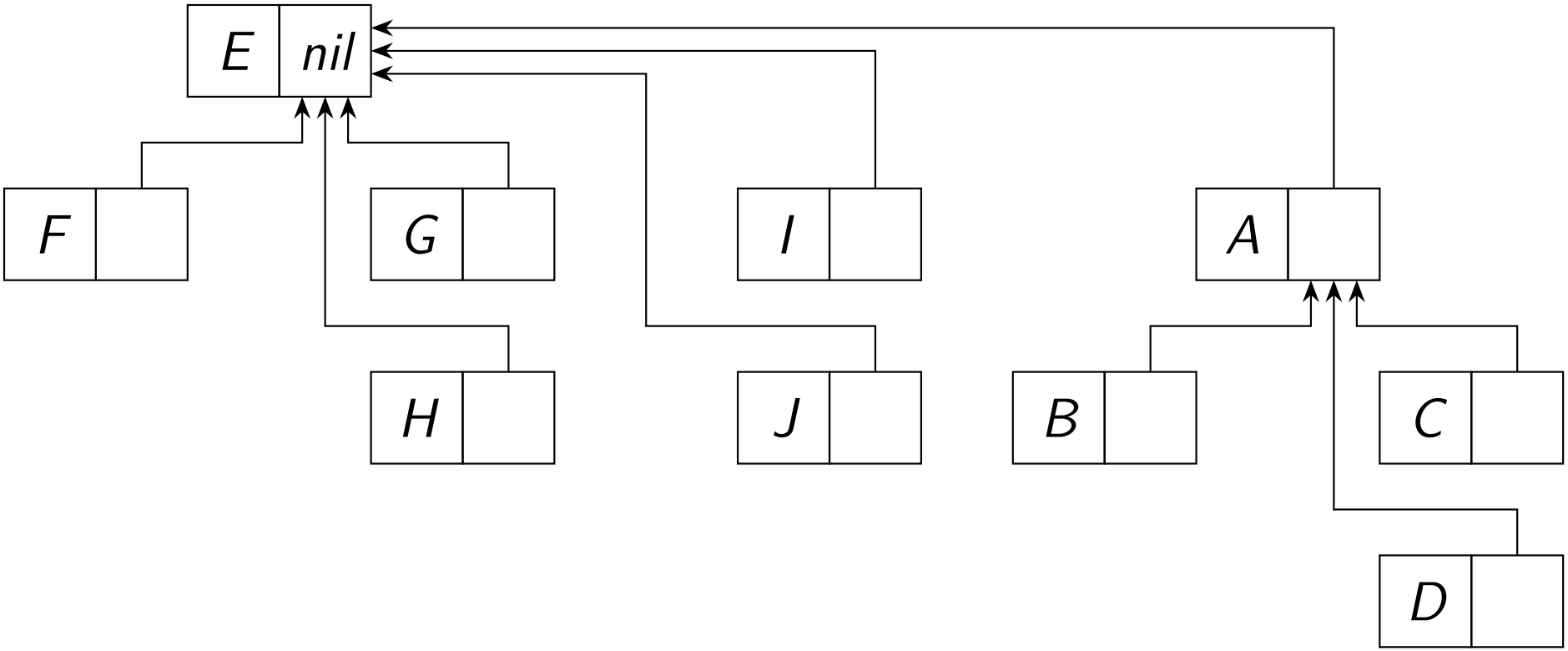
(b) Repeat the above, but with both balancing and path compression.

Problem2(a)



Problem2(b)

After the whole sequence of operations are performed:



Problem3

(6.21) The input is a set S with n real numbers. Design an $O(n)$ time algorithm to find a number that is *not* in the set. Prove that $\Omega(n)$ is a lower bound on the number of steps required to solve this problem.

Problem 3

- 找最小最大做處理
- 找到兩個最靠近的數座平均
- 很明顯為 $\Omega(n)$，因為 set 中每一個數字一定至少會被 scan 一次。

Problem4

(6.32) Prove that the sum of the heights of all nodes in a complete binary tree with n nodes is at most $n - 1$. (A complete binary tree with n nodes is one that can be compactly represented by an array A of size n, where the root is stored in $A[1]$ and the left and the right children of $A[i]$, $1 \leq i \leq \lfloor \frac{n}{2} \rfloor$, are stored respectively in $A[2i]$ and $A[2i + 1]$. Notice that, in Manber's book a complete binary tree is referred to as a balanced binary tree and a full binary tree as a complete binary tree. Manber's definitions seem to be less frequently used. Do not let the different names confuse you. "Balanced binary tree" in the original problem description is the same as "complete binary tree")

Problem 4

Prove *by induction* that the sum of the heights of all nodes in a complete binary tree with n nodes is at most $n - 1$. You may assume it is known that the sum of the heights of all nodes in a *full* binary tree of height h is $2^{h+1} - h - 2$. (Note: a single-node tree has height 0.)

Problem 4

From the solution to the preceding problem, we see that a complete binary tree is:

1. a single-node tree of height 0,
2. a two-node tree of height 1 where the root has a left child,
3. composed from a full binary tree of height h with n_l nodes and a complete (possibly full) binary tree of height h with n_r nodes as the left and the right subtrees of the root, resulting in a tree of height $h+1$ with $n_l + n_r + 1$ nodes, or
4. composed from a complete (possibly full) binary tree of height h with n_l nodes and a full binary tree of height $h-1$ with n_r nodes as the left and the right subtrees of the root, resulting also in a tree of height $h+1$ with $n_l + n_r + 1$ nodes.

Problem 4

Let $G(n)$ denote the sum of the heights of all nodes in a complete binary tree with n nodes. For a full binary tree (as a special case of complete binary trees) with $n = 2^{h+1} - 1$ nodes where h is the height of the tree, we already know that $G(n) = 2^{h+1} - (h+2) = n - (h+1) \leq n - 1$. With this as a basis, we prove that $G(n) \leq n - 1$ for the general case of arbitrary complete binary trees by induction on the number n (≥ 1) of nodes.

Problem 4

Base case ($n = 1$ or $n = 2$): When $n = 1$, the tree contains a single node whose height is 0. So, $G(n) = 0 \leq 1 - 1 = n - 1$. When $n = 2$, the tree has one additional node as the left child of the root. The height of the root is 1, while that of its left child is 0. So, $G(n) = 1 \leq 2 - 1 = n - 1$.

Inductive step ($n > 2$): If n happens to be equal to $2^{h+1} - 1$ for some $h \geq 1$, i.e., the tree is full, then we are done; note that this in particular covers the case of $n = 3 = 2^{1+1} - 1$. Otherwise, suppose $2^{h+1} - 1 < n < 2^{h+2} - 1$ ($h \geq 1$), i.e., the tree is a "proper" complete binary tree with height $h + 1 \geq 2$.

Problem 4

There are two cases to consider:
Case 1: The left subtree is full of height h with n_l nodes and the right one is complete also of height h with n_r nodes (such that $n_l + n_r + 1 = n$). From the special case of full binary trees and the induction hypothesis, $G(n_l) = 2^{h+1} - (h+2) = n_l - (h+1)$ and $G(n_r) \leq n_r - 1$. $G(n) = G(n_l) + G(n_r) + (h+1) \leq (n_l - (h+1)) + (n_r - 1) + (h+1) = (n_l + n_r + 1) - 2 \leq n - 1$.
Case 2: The left subtree is complete of height h with n_l nodes and the right one is full of height $h-1$ with n_r nodes. From the induction hypothesis and the special case of full binary trees, $G(n_l) \leq n_l - 1$ and $G(n_r) = 2^h - (h+1) = n_r - h$. $G(n) = G(n_l) + G(n_r) + (h+1) \leq (n_l - 1) + (n_r - h) + (h+1) = (n_l + n_r + 1) - 1 = n - 1$.

Problem5

Consider the *next* table as in the KMP algorithm for string $B[1..9] = abaababaa$.

1	2	3	4	5	6	7	8	9
a	b	a	a	b	a	b	a	a
-1	0	0	1	1	2	3	2	3

Suppose that, during an execution of the KMP algorithm, $B[6]$ (which is an a) is being compared with a letter in A, say $A[i]$, which is not an a and so the matching fails. The algorithm will next try to compare $B[next[6]+1]$, i.e., $B[3]$ which is also an a, with $A[i]$. The matching is bound to fail for the same reason. This comparison could have been avoided, as we know from B itself that $B[6]$ equals $B[3]$ and, if $B[6]$ does not match $A[i]$, then $B[3]$ certainly will not, either. $B[5]$, $B[8]$, and $B[9]$ all have the same problem, but $B[7]$ does not.

Please adapt the computation of the *next* table so that such wasted comparisons can be avoided.

Problem5

請改善 next 表，使得題目敘述的無用比較不會一直出現

最簡單的思路：
以題目中的 $B[6]$ 為例，next 表會帶到與 $B[3]$ 比較，但這兩個是一樣的字，到時還是得再查看 $next[3]$ 是什麼
那麼就不如讓 $next[6]$ 等於 $next[3]$ 吧，省了一次無用的比較
如果新的 $next[6]$ 不為 0，那就再看看 $B[next[6]+1]$ 是不是與 $B[6]$ 一樣，直到不同字，或者 $next[6]$ 變成 0 為止

需要注意的是，這個調整需要在所有的 next 值算完之後再進行，而不是算好原始的 $next[6]$ 就直接調整 $next[6]$，不然 $next[7]$ 的值會變得太小，使演算法出問題

Problem5

基礎版本

function ADJUST_NEXT(B, m)
  for $i := 3$ to m **do**
    $j := next[i]$;
    while $j > 0$ and $B[i] = B[j+1]$ **do**
      $j := next[j+1]$;
    $next[i] := j$;

Problem5

Before

1	2	3	4	5	6	7	8	9
a	b	a	a	b	a	b	a	a
-1	0	0	1	1	2	3	2	3

Problem5

After

1	2	3	4	5	6	7	8	9
a	b	a	a	b	a	b	a	a
-1	0	0	1	0	0	3	0	1

Problem5

After

1	2	3	4	5	6	7	8	9
a	b	a	a	b	a	b	a	a
-1	0	0	1	0	0	3	0	1

如果調整的動作在算 next 時同步進行呢？

$next[7]$ 將會初始化成 1 ($next[6]+1$)，然後又會因為 $B[7]=B[2]$ 使 $next[7]:=0$!

原本的 $next[7]=3$ 保留的訊息就被破壞掉了

Problem5

這已經有效減少無謂的比較發生的次數，對每個字元而言頂多出現 1 次
但當 $next[i] = 0$ 時，還是可能有 $B[i]$ 與 $B[1]$ 相同的問題
這種時候可以使 $next[i] := -1$

為了確保這麼做不會有事，回顧 String_Match 函數：
如果比 $B[1]$ 失敗，就使 i 加 1。若是提前觸發 i 加 1 的條件就能省去這次比較，使 $next[i] := -1$ 可以成功達成效果

如果要完全消除，則也別忘了 $B[2] = B[1]$ 的情境

Problem5

第二個版本

function ADJUST_NEXT(B, m)
  for $i := 2$ to m **do**
    $j := next[i]$;
    while $j \neq -1$ and $B[i] = B[j+1]$ **do**
      $j := next[j+1]$;
    $next[i] := j$;

Problem5

1	2	3	4	5	6	7	8	9
a	b	a	a	b	a	b	a	a
-1	0	-1	1	0	-1	3	-1	1

Problem5

如果 next 值的更新是由左往右做的，對右邊的值而言，它們與左邊的字元若是一樣的，抽它們的 next 值後，這個新的 next 值就必然不會產生多餘的比較
意思是，這函數裡頭的 while 迴圈可以用 if 替代

Problem5

第三個版本（以第二個版本為基底做修改）

function ADJUST_NEXT(B, m)
  for $i := 2$ to m **do**
    if $B[i] = B[next[i] + 1]$ **then**
      $next[i] := next[next[i] + 1]$;