Homework 6

Yu-Ju Teng Ling-Hsuan Chen

Problem 1

1. 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: union(A,B), union(C,D), union(E,F), union(G,H), union(I,J), union(A,D), union(F,G), union(D,J), union(J,H).

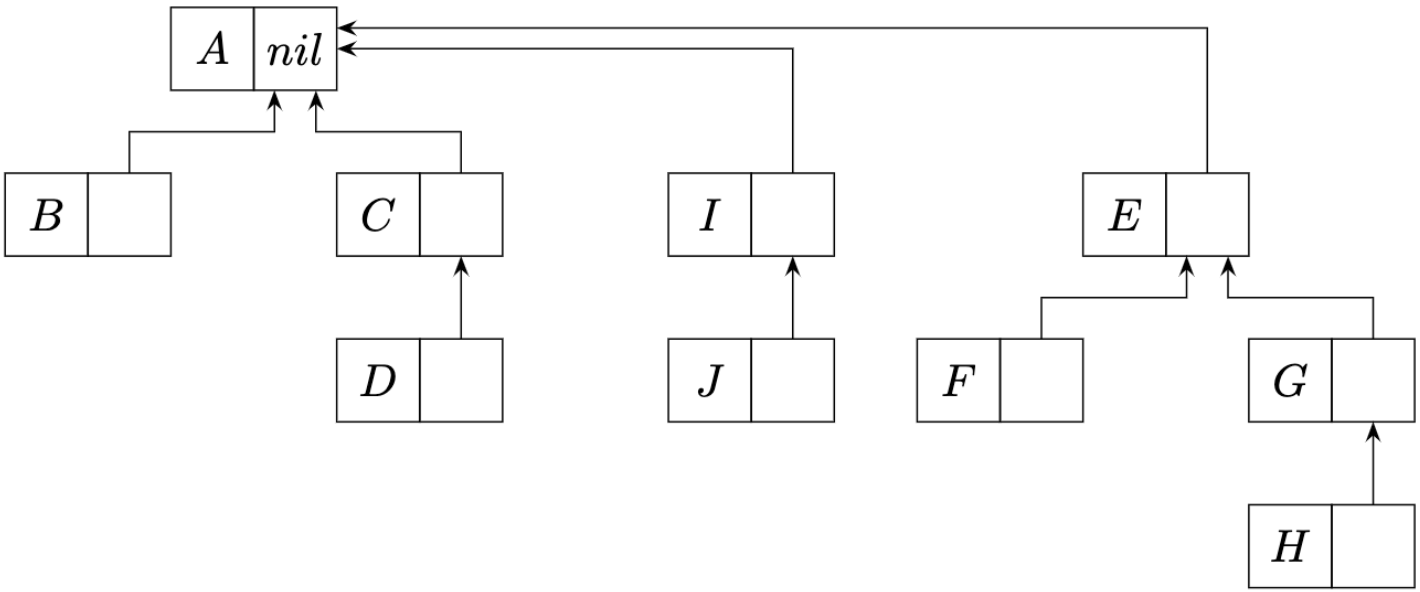
   In the case of combining two groups of the same size, please always point the second group to the first.

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

Problem 1(a)

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

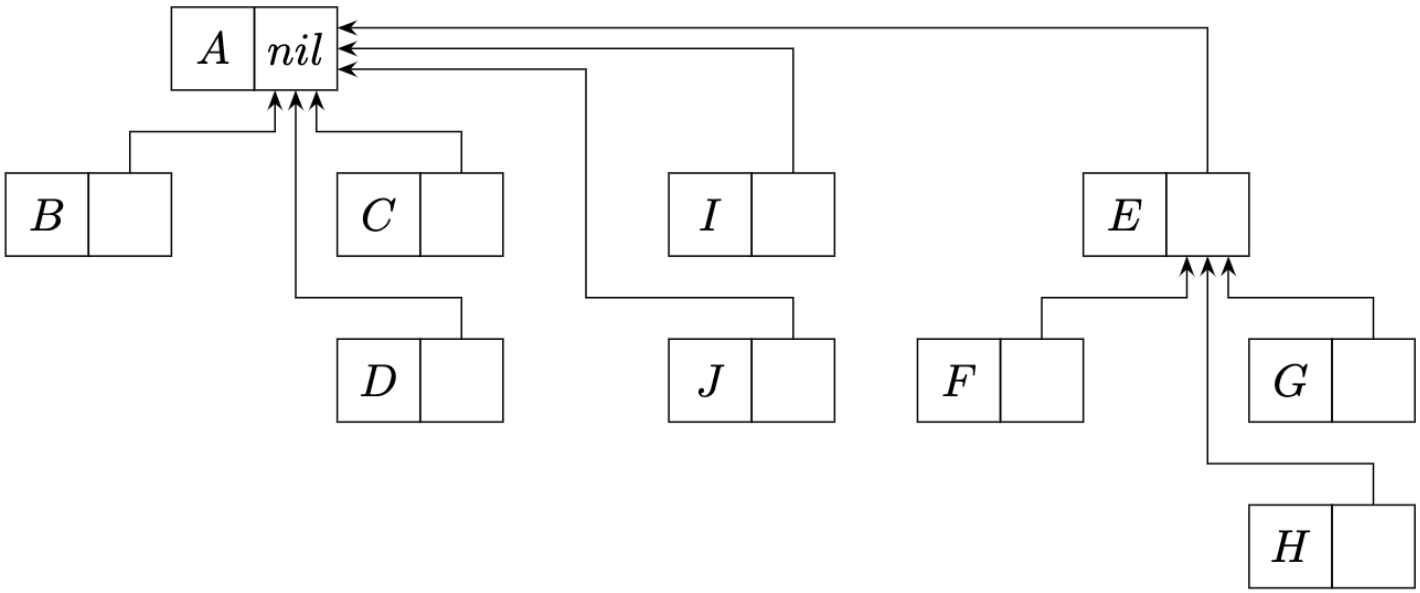
Solution.



Problem 1(b)

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

Solution.



Problem 2

2. (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 2

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 $2^{h+1} - h - 2$.

Problem 2

There are several patterns of complete binary tree:

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 and a complete (possibly full) binary tree of height h as the left and the right subtrees of the root, resulting in a tree of height $h + 1$.
4. composed from a complete (possibly full) binary tree of height h and a full binary tree of height $h - 1$ as the left and the right subtrees of the root, resulting also in a tree of height $h + 1$.

Problem 2

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 (a special case of complete binary tree) of height h with $n = 2^{h+1} - 1$ nodes, we already know that:

$$G(n) = 2^{h+1} - (h+2) = 2^{h+1} - 1 - (h+1) = n - (h+1) \leq n - 1.$$

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 2

Base case ($n = 1$ or $n = 2$):

When $n = 1$, $G(n) = 0 \leq 1 - 1 = n - 1$.
When $n = 2$, $G(n) = 1 \leq 2 - 1 = n - 1$.

Inductive step ($n > 2$):

If n happens to be equal to $2^{h+1} - 1$ for some $h \geq 1$, the tree is a full binary tree, then we are done.

Otherwise, suppose $2^{h+1} - 1 < n < 2^{h+2} - 1$ ($h \geq 1$), the tree is a "proper" complete binary tree with height $h + 1 \geq 2$, There are two cases to consider:

Problem 2

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 tree and the induction hypothesis:

$$\begin{aligned} G(n_l) &= 2^{h+1} - (h+2) = n_l - (h+1) \\ G(n_r) &\leq n_r - 1, \end{aligned}$$

we have

$$\begin{aligned} G(n) &= G(n_l) + G(n_r) + (h+1) \\ &\leq (n_l - (h+1)) + (n_r - 1) + (h+1) \\ &= (n_l + n_r + 1) - 2 \\ &\leq n - 1. \end{aligned}$$

Problem 2

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:

$$\begin{aligned} G(n_l) &\leq n_l - 1 \\ G(n_r) &= 2^h - (h+1) = n_r - h, \end{aligned}$$

we have

$$\begin{aligned} G(n) &= G(n_l) + G(n_r) + (h+1) \\ &\leq (n_l - 1) + (n_r - h) + (h+1) \\ &= (n_l + n_r + 1) - 1 \\ &\leq n - 1. \end{aligned}$$

Problem 3

3. (6.40) Design an algorithm that, given a set of integers $S = \{x_1, x_2, \ldots, x_n\}$, finds a nonempty subset $R \subseteq S$, such that

$$\sum_{x_i \in R} x_i \equiv 0 \pmod{n}.$$

Before presenting your algorithm, please argue why such a nonempty subset must exist.

Problem 3

Let S_i represent the subset sum of x_i for $i = 1, \ldots, n$.
We have

$$
\begin{aligned}
S_1 &= x_1 \\
S_2 &= x_1 + x_2 \\
&\vdots \\
S_k &= x_1 + x_2 + \ldots + x_k \\
&\vdots \\
S_n &= x_1 + x_2 + \ldots + x_k + \ldots + x_n.
\end{aligned}
$$

We want to prove the statement "There must exist a non-empty subset satisfying the condition $S_k \equiv 0 \pmod{n}$." is true.

Problem 3

Case 1: Directly find a subset S_k that satisfied the condition.

If there exists a subset S_k satisfying the condition $S_k \equiv 0 \pmod{n}$, it is trivial to demonstrate the existence of a nonempty subset.

Case 2: If there is not exist a subset S_k satisfied the condition.

Assume there does not exist a subset S_k satisfying the condition.
By the Pigeonhole principle, there must be at least two sets with the same remainder value among $\{1, \ldots, n-1\}$.
Consider the pair (i, j) where S_i and S_j have the same remainder, which is less than or equal to $n-1$ but not equal to 0, with $i < j$.
We can construct a new set satisfying the condition $S_k \equiv 0 \pmod{n}$ by removing the common elements between S_i and S_j. In other words, the set of differences between S_i and S_j, denoted as $\{x_{i+1}, \ldots, x_j\}$, is our solution.

In conclusion, there must exist a non-empty subset.

Problem 3

function SubSet_R(S, n)

// S is the set and n is the number of elements in set S

$R := [0]$ // Record the remainder of each subset

$Appear := [-1]$ // Record the appearance of remainder

$R[0] = 0$

for $i := 1$ **to** n **do**

$R[i] = (R[i-1] + x_i) \pmod{n}$

if $R[i] = 0$ **then**

return $\{x_1, ..., x_i\}$

else if $Appear[R[i]] \neq -1$ **then**

return $\{x_{Appear[R[i]]+1}, ..., x_i\}$

else

$Appear[R[i]] = i$

Problem 4

4. 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. Also, please give, for string $B[1..9] = abbaabbaa$, the values of the original *next* table and those of the new *next* table according to the adaptation.

Problem 4

To avoid wasted comparisons, the simplest idea is to recheck repeated the comparison when the next value is calculated.

Note: The repeated check must be started after the next values have been calculated, and cannot be conducted directly during the calculation.

Problem 4

Original **Compute_Next** function:

function COMPUTE_NEXT(B, m)
    $next[1] := -1; next[2] := 0;$
    for $i := 3$ **to** m **do**
        $j := next[i-1] + 1;$
        while $B[i-1] \neq B[j]$ and $j > 0$ **do**
            $j := next[j] + 1;$
        $next[i] := j;$

Problem 4

New **Compute_Next_Adapted** function:

```
function COMPUTE_NEXT_ADAPTED(B, m)
    next[1] := −1; next[2] := 0;
    for i := 3 to m do
        j := next[i − 1] + 1;
        while B[i − 1] ≠ B[j] and j > 0 do
            j := next[j] + 1;
        next[i] := j;
    // Add the following lines for optimization but less efficient.
    for i := m down to 2 do
        j := next[i] + 1;
        while B[i] = B[j] and j > 0 do
            j := next[j] + 1;
        next[i] := j − 1;
```

Problem 4

Another way:

function COMPUTE_NEXT_ADAPTED(B, m)

```
next[1] := −1; next[2] := 0;
for i := 3 to m do
    j := next[i − 1] + 1;
    while B[i − 1] ≠ B[j] and j > 0 do
        j := next[j] + 1;
    next[i] := j;
// Add the following lines for optimization.
for i := 2 to m do
    j := next[i] + 1;
    if B[i] = B[j] and j > 0 then
        next[i] := next[j];
```

Problem 4

For string $B[1..9] = abbaabbaa$, the original *next* table:

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

New *next* table:

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

Problem 5

5. (6.17 adapted) Given two strings $A = bbaaa$ and $B = bbbaba$, what is the result of the minimal cost matrix $C[0..5, 0..6]$, according to the algorithm discussed in class for changing A character by character into B? Aside from giving the cost matrix, please show the details of how the entry $C[4, 5]$ is computed from the values of $C[3, 4]$, $C[3, 5]$, and $C[4, 4]$.

Problem 5

Let $C(i,j)$ denote the minimum cost of changing $A(i)$ to $B(i)$, where $A(i) = a_1 a_2 \cdots a_i$ and $B(i) = b_1 b_2 \cdots b_j$.

- For $i = 0$ or $j = 0$,
$$\begin{cases} C(i,0) = i \\ C(0,j) = j \end{cases}$$
- For $i > 0$ and $j > 0$,
$$C(i,j) = \min \begin{cases} C(i-1,j)+1 & (\text{deleting } a_i) \\ C(i,j-1)+1 & (\text{inserting } b_j) \\ C(i-1,j-1)+1 & (a_i \to b_j) \\ C(i-1,j-1) & (a_i = b_j) \end{cases}$$

Problem 5

		1	2	3	4	5	6
		b	**b**	**b**	**a**	**b**	**a**
1	**b**	0	1	2	3	4	5
2	**b**	1	0	1	2	3	4
3	**a**	2	1	1	**1**	**2**	3
4	**a**	3	2	2	**1**	**2**	2
5	**a**	4	3	3	2	2	2

$$C[4,5] = \min \left\{ \begin{array}{ll} C[3,5]+1 = 2+1 = 3 & (\text{deleting } a_4), \\ C[4,4]+1 = 1+1 = 2 & (\text{inserting } b_5), \\ C[3,4]+1 = 1+1 = 2 & (a_4 \neq b_5) \end{array} \right\} = 2.$$