CSC3100 Assignment 3

Chen Ang (118010009)

Binary Search Trees

Problem 1

```
INSERT-NODE(node, z)
    /* z: a node with key to insert,
     *    z.left == z.right == NIL,
     *    assuming no repeated keys */
    if node == NIL
        return z
    if z.key < node.key
        node.left = INSERT-NODE(node.left, z)
    else
        node.right = INSERT-NODE(node.right, z)

/* Wrapper function */
TREE-INSERT(T, z)
    T.root = INSERT-NODE(T.root, z)
```

Problem 2

The inorder tree walk visits each node exactly once and at each visit does only constant work (printing the node). Hence the walk takes $\Theta(n)$ time. The total running time is then given by

$$T(n) = \sum_{i=1}^{n} \Theta(c_i) + \Theta(n)$$

where c_i is the number of comparisons made at i-th insertion.

The worst case is a tree skewed to the far-left or far-right in a straight chain. For example, if we insert integers 1 to n in an ascending order, at i-th insertion we have to compare key i with every node in the tree before ending up at the bottom-right. In this case we have that $c_i = i - 1$, and so

$$T(n) = \Theta\left(\sum_{i=1}^{n} i\right) + \Theta(n) = \Theta(n(n+1)/2) = \Theta(n^2)$$

The best case is when the tree is filled up in a level-by-level fashion, that is, when the sequence $a_1, a_2 \cdots a_n$ inserted satisfies

$$a_{2k} < a_k < a_{2k+1}, \quad \forall \text{ valid } k$$

In this case, the i-th insertion requires comparisons of a_i with some element on each non-leaf layer of the tree comprised of $a_1, \cdots a_{i-1}$ (except if i is a power of two, in which case an extra comparison with the leftmost leaf node is required). Therefore, the number of comparison made at i-th insertion is bounded by

$$\log i - 1 \le c_i \le \log i \implies c_i = \Theta(\log i)$$

and so the best-case running time is given by

$$T(n) = \Theta\left(\sum_{i=1}^{n} \log i\right) + \Theta(n) = \Theta(\log(n!)) = \Theta(n \log n)$$

using Stirling's approximation for factorials.

Red-Black Trees

Problem 3

Let L denote the longest path from node x to *some* descendent leaf, and let l denote the shortest such path. We wish to show

$$|L| \leq 2|l|$$

The property of RB-tree requires the number of black nodes on L and l to be equal, and that no two red nodes are parent-child on both paths. If x is a red node, the longest path L can make is with alternating B-R nodes (since no two consecutive nodes on L can both be red):

$$L : x(R) \to \overbrace{B \to R \to \cdots \to B}^{\mathrm{bh}(x) \text{ blacks}}$$

The shortest path l can make is with all black nodes (lower bounded by $\mathrm{bh}(x)$ black nodes on l):

$$l : x(R) \to \overbrace{B \to B \to \cdots \to B}^{\mathrm{bh}(x) \text{ blacks}}$$

In this case,

$$|L| = 2 \cdot \mathrm{bh}(x) - 1 \leq 2|l| = 2 \cdot \mathrm{bh}(x)$$

If x is itself a black node, then the longest path L can make is with alternating R-B nodes:

$$L : x(B) \to \overbrace{R \to B \to \cdots \to B}^{\mathrm{bh}(x) \text{ blacks}}$$

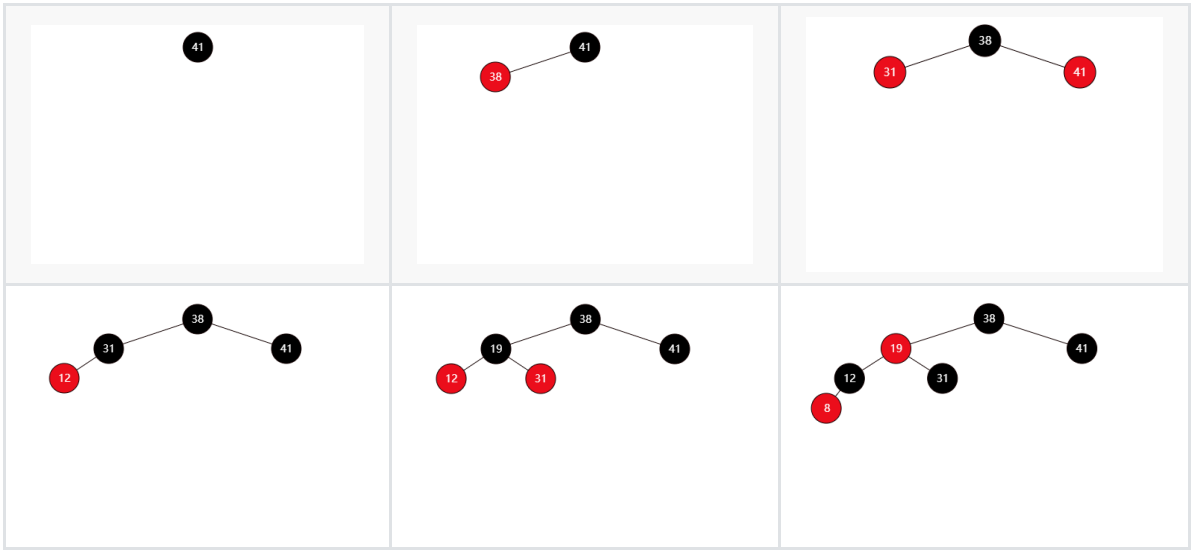
while the shortest path l can make remains all black. In this case,

$$|L| = 2 \cdot \mathrm{bh}(x) \leq 2|l| = 2 \cdot \mathrm{bh}(x)$$

establishing the desired inequality.

Problem 4

All NIL nodes (black) under the leaves are omitted from the graphs for clarity.



Graph Algorithms

Problem 5

```
BFS(G, s)
 1  for each vertex u ∈ G.V - {s}
 2      u.color = WHITE
 3      u.d = ∞
 4      u.π = NIL
 5  s.color = GRAY
 6  s.d = 0
 7  s.π = NIL
 8  Q = ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11      u = DEQUEUE(Q)
12      for each v ∈ G.Adj[u]
13          if v.color == WHITE
14              v.color = GRAY
15              v.d = u.d + 1
16              v.π = u
17              ENQUEUE(Q, v)
18  /*  u.color = BLACK  */
```

We claim that deleting line 18 does not affect the output. Notice that the only place where the control flow of the program could be affected by vertices' color is line 13, the if-condition checking whether the color is `WHITE`. All vertices were either colored `GRAY` when first examined as a neighboring vertex on line 14 or initialized at `GRAY` (for source `s`). At any point later in the execution it could either be `GRAY` or `BLACK` (both non-`WHITE`), thus always failing the if-condition. With line 18 removed, the only difference is that all vertices will remain `GRAY` from the moment they are colored so (still non-`WHITE`), meaning they will fail the if-condition just as before. Consequently the control flow of the program is unchanged. And since line 18 does not directly alter the final output (`d` and `π`), the result is the same as well.

Problem 6

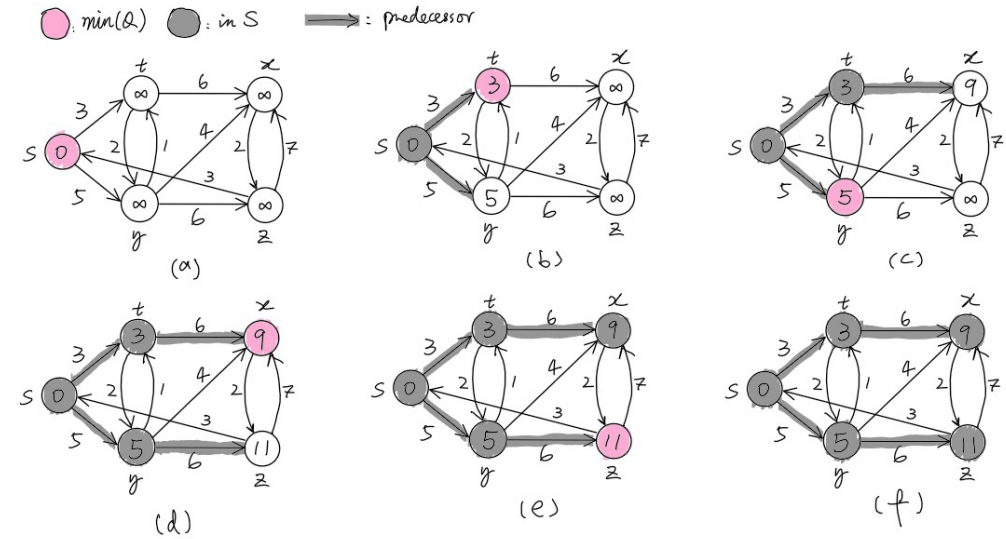
```
DFS(G)
1  for each vertex u ∈ G.V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6      if u.color == WHITE
7          DFS-VISIT(G, u)

DFS-VISIT(G, u)
 1  time = time + 1             /* white vertex u has just been discovered */
 2  u.d = time
 3  u.color = GRAY
 4  for each v ∈ G.Adj[u]       /* explore edge (u, v) */
 5      if v.color == WHITE
 6          v.π = u
 7          DFS-VISIT(G, v)
 8  /* u.color = BLACK */       /* blacken u; it is finished */
 9  time = time + 1
10  u.f = time
```

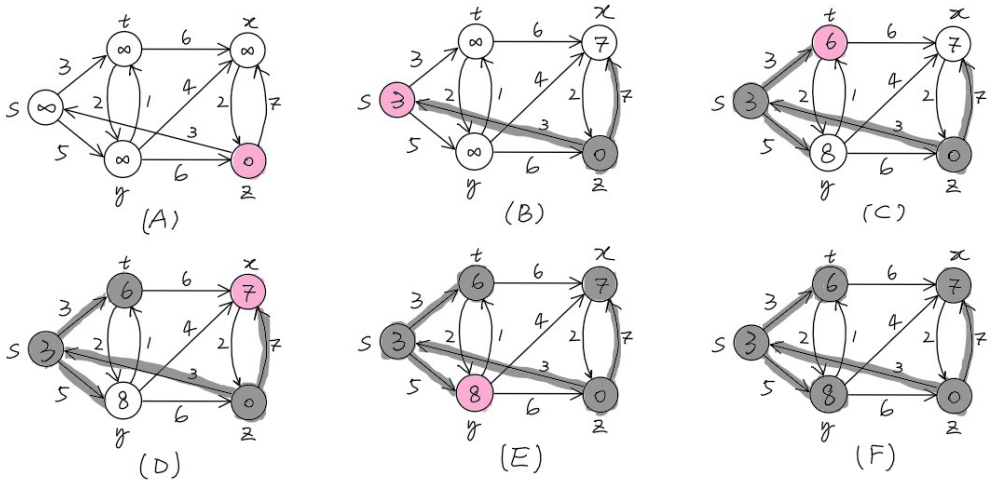
We claim that deleting line 8 of `DFS-VISIT` does not affect the final outcome of `DFS`. Two points in the control flow could be affected by the color of the vertices: line 6 of `DFS` which checks if the vertex is `WHITE`, and line 5 of `DFS-VISIT` which checks if the neighboring vertex is `WHITE`. Originally, vertices were colored `GRAY` when they were picked up as an unvisited `WHITE` vertex. From then they were either `GRAY` or `BLACK` (non-`WHITE`), both preventing them from entering the if-clauses. With line 8 of `DFS-VISIT` deleted, the vertices will behave exactly the same except they will remain `GRAY` (still non-`WHITE`) from the moment they are colored so. Thus they will not pass any of the if-conditions later in the program either. This means the control flow of the program remains unchanged. And since the deleted line does not directly alter the final output (`d` and `π`), the result is also the same.

Problem 7

By definition of minimum spanning trees, u must belong to some minimum spanning tree T of $G = (V, E)$. Let $A := \emptyset \subset T$. Then $C = (\{u\}, \emptyset)$ is a connected component in the forest $G_A = (V, A)$. Further, (u, v) is a light edge connecting C to another connected component $(\{v\}, \emptyset) \subset G_A$, as the edge clearly connects two components and is of the minimum weight possible. Therefore, (u, v) is safe for $A = \emptyset$ by ***Corollary 23.2*** of CLRS, i.e., (u, v) belongs to some minimum spanning tree of G.

Problem 8


: min(Q)
: in S
: predecessor
(a)
(b)
(c)
(d)
(e)
(f)


(A)
(B)
(C)
(D)
(E)
(F)