# Chapter 5
# Trees

(課本 p.210 Program 5.4)

## Non-recursive Inorder Traversal

Need a stack (because we have no pointer that points to parent)

*s:* stack.

*current_node:* tree node pointer.

```
void iter_inorder(tree_pointer current_node)
{
    do {
        while(current_node != NULL) {
            push(current_node, s);
            current_node = current_node->left_child;
        }
        if( Not stack_empty(s) ) then {
            current_node = pop( s);
            printf("%d", current_node->data);
            current_node = current_node->right_child;
        }
        else
            done = true;
    }
    while (done == false);
}
```

(課本 p. 211 Program 5.5)

## Non-recursive Levelorder Traversal

Need a queue.

*q*: queue

*current_node:* tree node pointer.

```
void level_order(tree_pointer current_node)
{
    if(current_node == NULL) then
        return (emptytree());
    else {
        addq(current_node, q);
        while(Not empty(q)) {
            current_node = deleteq(q);
            printf("%d", current_node->data);
            if(current_node->left_child != NULL) then
                addq(current_node->left_child, q);
            if(current_node->right_child != NULL) then
                addq(current_node->right_child, q);
        }
    }
}
```

# Chapter 6 Graphs

Un-directed graph

(課本 p. 295 Program 6.7)

## Kruskal algorithm

```
1.  T = NULL; /*edge set*/
2.  while(|T| < n-1) {
3.      Choose an edge (V, W) from E of least cost.
4.      delete (V, W) from E
5.      if((V, W) doesn't create a cycle)
6.          then  add (V, W) to T
7.          else  discard(V, W)
    }
```

(課本 p. 297 Program 6.8)

## Prim's algorithm

```
1.  T = NULL  /*edge set*/
2.  TV = {1}  /*vertices set*/
3.  while ( |T|  <  n-1) {
4.      Let (u, v) be a least cost edge such that u in TV and v not in TV;
5.      T = T ∪ {(u, v)};
6.      TV = TV ∪ {v};
7.      update the near_to_tree array.
    }
```

Without step 7: time complexity : O(n*e)

n: number of vertices

e: number of edges. In each iteration, at most e edges are examined

With step 7: time complexity : O(n)

(課本 p. 298 Figure 6.24 6.25)

## Sollin's algorithm

Good for parallel processing.

step1 : Initially all vertices are tree by itself

While (there is more than one tree ) {

step2: For each tree, select one minimum cost edge which has one end in the tree and the other outside the tree

step3: Delete the edge with the duplicate copies and if two edges with the same cost connecting two trees, retain only one (to avoid the situation that two trees select to different edges that have the same cost)

step4: add the edges

}