1. Linked queue

Microsoft Visual Studio 偵錯主控台

```
Do what???
1. Insert
2. Pop
3. Front number
4. Back number
5. Size
6. Print queue
7. Quit

1
Insert value: 2
Queue: 2

1
Insert value: 22
Queue: 2 22

1
Insert value: 222
Queue: 2 22 222

3
the first element is 2

4
the last element is 222

5
the size of the queue is 3

6
Queue: 2 22 222

2
Number 2 has been popped.
Queue: 22 222

2
Number 22 has been popped.
Queue: 222

2
Number 222 has been popped.
Queue Is Empty!!!!!

2
Queue Is Empty!!!!!
```

Microsoft Visual Studio 偵錯主控台

```
3
the first element is 2

4
the last element is 222

5
the size of the queue is 3

6
Queue: 2 22 222

2
Number 2 has been popped.
Queue: 22 222

2
Number 22 has been popped.
Queue: 222

2
Number 222 has been popped.
Queue Is Empty!!!!!

2
Queue Is Empty!!!!!

3
Queue Is Empty!!!!!

4
Queue Is Empty!!!!!

5
the size of the queue is 0

6
Queue Is Empty!!!!!

7
Bye bye!!!

C:\Users\User\vs\hw2\Debug\hw2.exe (處理序 29332
若要在偵錯停止時自動關閉主控台，請啟用 [工具] ->
```

```cpp
#include<iostream>
using namespace std;
template<typename T> struct Node
{
    Node<T>* nxt;
    T val;
    Node() {
        nxt = NULL;
    }
};
template<typename T> class Queue {
public:
    int size = 0;
    Node<T>* head, * tail;
    Queue() {
        head = new Node<T>;
        head->nxt = NULL;
        tail = head;
    }
    void push(T val) {
        size++;
        Node<T>* add;
        add = new Node<T>;
        add->val = val;
        add->nxt = NULL;
        tail->nxt = add;
        tail = tail->nxt;
    }
    void pop() {
        if (size == 0)  cout << "Queue Is Empty!!!!!\n";
```

```cpp
    }
    void pop() {
        if (size == 0)  cout << "Queue Is Empty!!!!!\n";
        else
        {
            cout << "Number " << (head->nxt)->val << " has been popped.\n";
            size--;
            head = head->nxt;
            print();
        }

    }
    void front() {
        if (size == 0)  cout << "Queue Is Empty!!!!!\n";
        else cout << "the first element is " << (head->nxt)->val << "\n";
    }
    void back() {
        if (size == 0) cout << "Queue Is Empty!!!!!\n";
        else cout << "the last element is " << tail->val << "\n";
    }
    void size_q() {
        cout << "the size of the queue is " << size << "\n";
    }
    void print() {
        Node<T>* tmp = head;
        if (size == 0)  cout << "Queue Is Empty!!!!!\n";
        else
        {
            cout << "Queue: ";
            while (tmp != tail) {
```

找不到任何問題

```cpp
            cout << "Queue: ";
            while (tmp != tail) {
                cout << (tmp->nxt)->val << " ";
                tmp = tmp->nxt;
            }
            cout << "\n";
        }
    }
};

int main() {
    Queue<int> q;
    cout << "Do what???\n";
    cout << "1. Insert\n2. Pop\n3. Front number\n4. Back number\n5. Size\n6. Print queue\n7. Quit\n\n";
    int flag = 1;
    while (flag) {
        int n;
        cin >> n;
        switch (n)
        {
        case 1:
            int m;
            cout << "Insert value: ";
            cin >> m;
            q.push(m);
            q.print();
            break;
        case 2:
            q.pop();
```

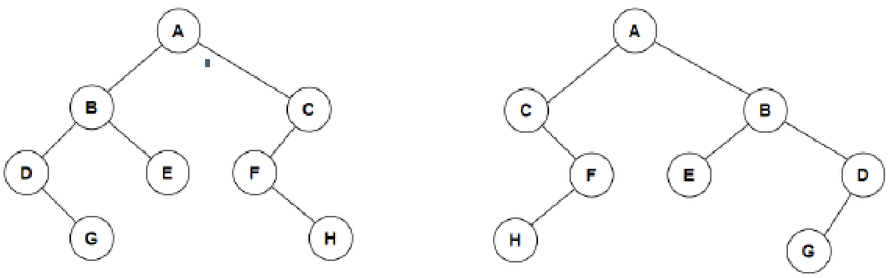
找不到任何問題

```
                cin >> m;
                q.push(m);
                q.print();
                break;
            case 2:
                q.pop();
                break;
            case 3:
                q.front();
                break;
            case 4:
                q.back();
                break;
            case 5:
                q.size_q();
                break;
            case 6:
                q.print();
                break;
            case 7:
                flag = 0;
                cout << "Bye bye!!!";
                break;
            default:
                cout << "Please input number 1 to 7 !!!\n";
                break;
            }
            cout << "\n";
        }
}
```

2. Swaptree()

Microsoft Visual Studio 偵錯主控台

```
Traverse the tree: D G B E A F H C

Traverse the swaptree: C H F A E B G D
```



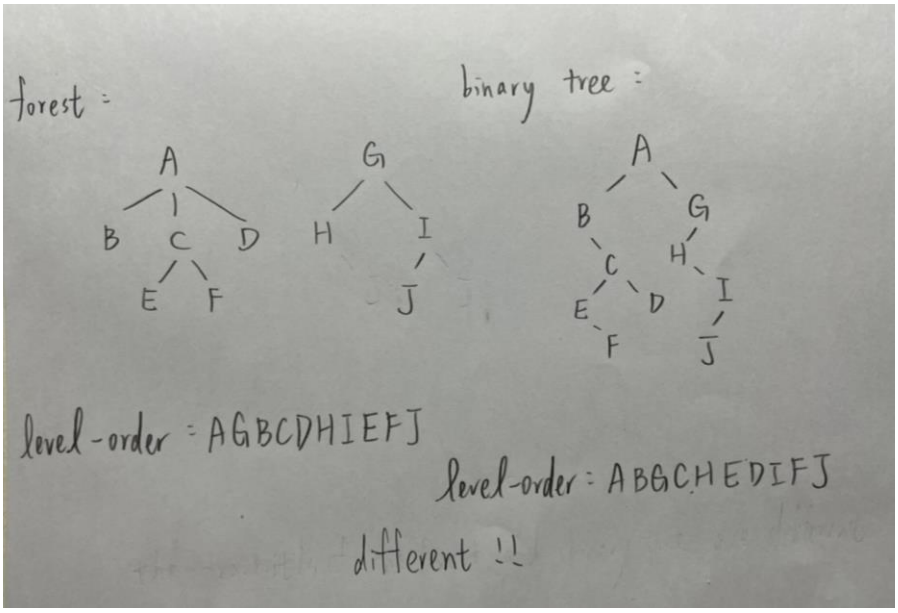
t *t.SwapTree()*

```cpp
#include<iostream>
using namespace std;
struct Node {
    char data;
    struct Node* left;
    struct Node* right;
};
struct Node* insert_node(char data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    return node;
}
void swaptree(struct Node* node) {
    if (node == NULL) {
        return;
    }
    else {
        swaptree(node->left);
        swaptree(node->right);
        struct Node* tmp;
        tmp = node->left;
        node->left = node->right;
        node->right = tmp;
    }
}
void traversal(struct Node* node) {          //inorder traversal
    if (node == NULL)    return;
    traversal(node->left);
```

找不到任何問題

```cpp
}
void traversal(struct Node* node) {          //inorder traversal
    if (node == NULL)    return;
    traversal(node->left);
    cout << node->data << " ";
    traversal(node->right);
}
int main() {
    struct Node* root = insert_node('A');
    root->left = insert_node('B');
    root->left->left = insert_node('D');
    root->left->left->right = insert_node('G');
    root->left->right = insert_node('E');
    root->right = insert_node('C');
    root->right->left = insert_node('F');
    root->right->left->right = insert_node('H');

    cout << "Traverse the tree: ";
    traversal(root);
    cout << "\n\n";
    swaptree(root);
    cout << "Traverse the swaptree: ";
    traversal(root);
    cout << "\n\n";
}
```

3. Level-order traversal



forest :

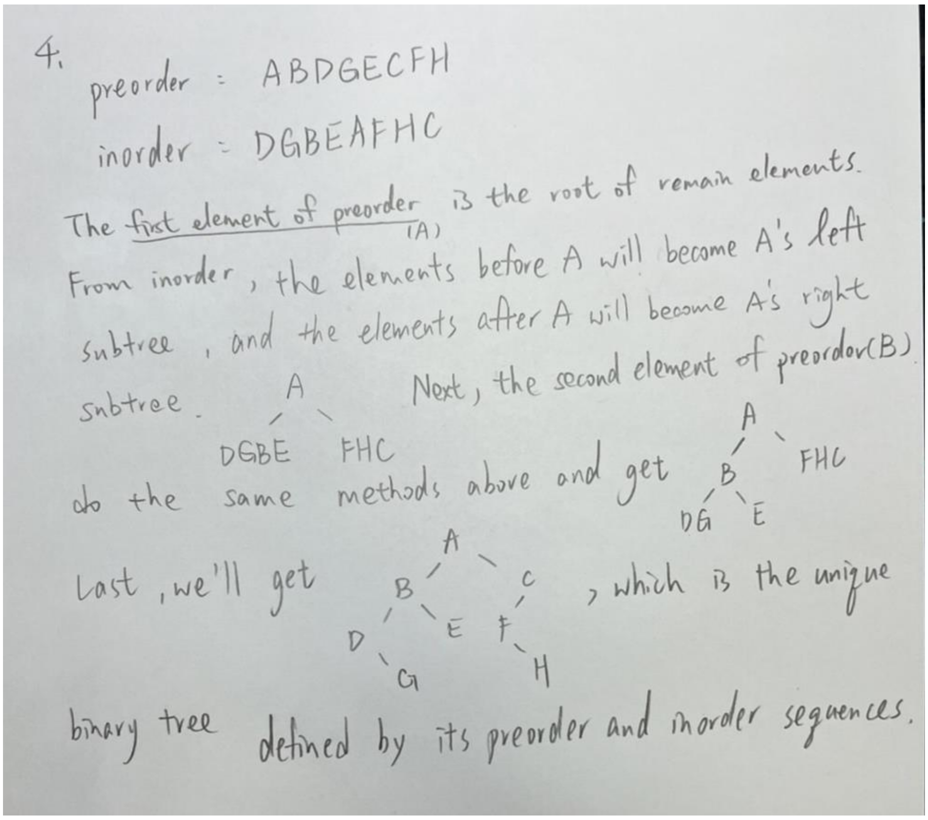
binary tree :

level-order = AGBCDHIEFJ

level-order : ABGCHEDIFJ

different !!

4. Unique tree from preorder and inorder



4.

preorder : ABDGECFH

inorder : DGBEAFHC

The first element of preorder (A) is the root of remain elements.

From inorder, the elements before A will become A's left subtree, and the elements after A will become A's right subtree.

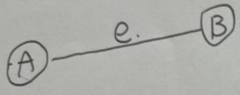
Next, the second element of preordor(B)

do the same methods above and get

Last, we'll get , which is the unique binary tree defined by its preorder and inorder sequences.

5. Sum of degrees

5.



e is an edge of the graph which has 2 ends (A, B)

we count twice when we sum the vertex degrees

(deg (A) and deg (B))

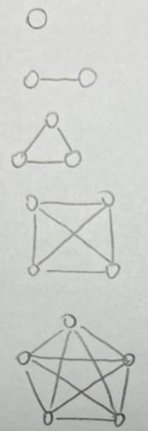
Hence, every edge counts twice in the graph.

$\Rightarrow$ Sum of degrees = 2E

↑ number of edges.

6. n(n-1)/2

6.



	edges
one :	0
two :	1
three :	3
four :	6
five :	10

each vertex has n-1 edges times n vertex,

every edge will count twice for two vertex, so.

the number of edges in an n-vertex complete graph is

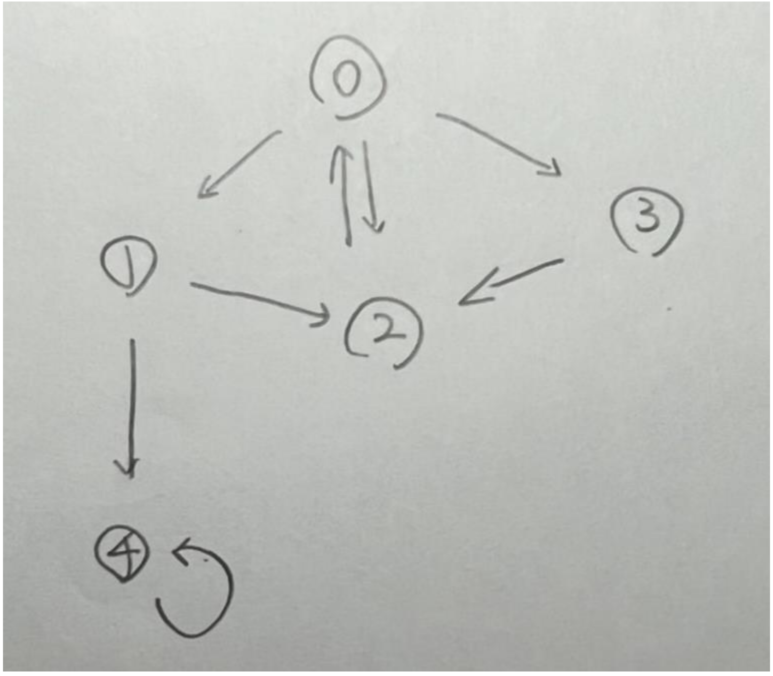
$\frac{n(n-1)}{2}$ ※

7. bfs

```
t main() {
  Graph g(5);
  g.add(0, 1);
  g.add(0, 2);
  g.add(0, 3);
  g.add(1, 2);
  g.add(1, 4);
  g.add(2, 0);
  g.add(3, 2);
  g.add(4, 4);
  cout << "BFS from vertex 2: ";
  g.bfs(2);
```

Microsoft Visual Studio 偵錯主控台

```
BFS from vertex 2: 2 0 1 3 4
C:\Users\User\vs\hw2\Debug\hw2.
若要在偵錯停止時自動關閉主控台，
按任意鍵關閉此視窗…
```



```cpp
#include<iostream>
#include<list>
using namespace std;

class Graph {
    int vertex;
    list<int>* adj;
public:
    Graph(int vertex) {
        this->vertex = vertex;
        adj = new list<int>[vertex];
    }
    void add(int value, int index) {
        adj[value].push_back(index);
    }
    void bfs(int s) {
        bool* visited = new bool[vertex];
        for (int i = 0; i < vertex; i++) {
            visited[i] = false;
        }

        list<int> queue;
        visited[s] = true;
        queue.push_back(s);
        list<int>::iterator i;
        while (!queue.empty()) {
            s = queue.front();
            cout << s << " ";
            queue.pop_front();
            for (i = adj[s].begin(); i != adj[s].end(); i++) {
```

找不到任何問題


```cpp
            cout << s << " ";
            queue.pop_front();
            for (i = adj[s].begin(); i != adj[s].end(); i++) {
```

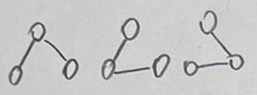

```cpp
                if (!visited[*i]) {
                    visited[*i] = true;
                    queue.push_back(*i);
                }
            }
        }
    }
};

int main() {
    Graph g(5);
    g.add(0, 1);
    g.add(0, 2);
    g.add(0, 3);
    g.add(1, 2);
    g.add(1, 4);
    g.add(2, 0);
    g.add(3, 2);
    g.add(4, 4);
    cout << "BFS from vertex 2: ";
    g.bfs(2);
}
```

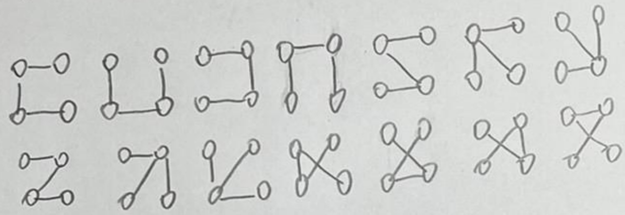
8. number of spanning tree

8、

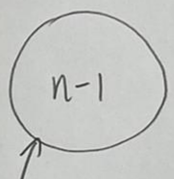
$n=1$, o , 1 種 $\ge 2^{1-1}-1=0$

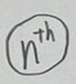
$n=2$, o—o , 1 種 $\ge 2^{2-1}-1=1$

$n=3$,  , 3種 $\ge 2^{3-1}-1=3$

$n=4$,  , 14種 $\ge 2^{4-1}-1 = 7$

$n \ge 5$





連接 n^{th} 到前 $n-1$ 個至少 $n-1$ 種

$f(n-1)$ 種

$\Rightarrow f(n) \ge f(n-1)\cdot(n-1)$

同理，$f(n-1) \ge f(n-2)(n-2)$

因此，$f(n) \ge f(n-1)\cdot(n-1) \ge f(n-2)\cdot(n-2)(n-1) \ge f(n-3)\cdot(n-3)(n-2)(n-1) \ge (n-1)!$

$\Rightarrow$ prove $(n-1)! \ge 2^{n-1}-1 \quad \forall n \ge 5$

$n=5$, $4! = 24 \ge 2^{4-1}-1 = 7$ true.

假設 $n=k$ 時 $(k-1)! \ge 2^{k-1}-1$ 成立

$\Rightarrow k! = k(k-1)! \ge k\cdot 2^{k-1}-1 > 2^{k}-1 \quad (\because k \ge 5)$ 成立

由數學歸納法得証 $(n-1)! \ge 2^{n-1}-1 \quad \forall n \ge 5$

$\Rightarrow$ # of spanning trees in a complete graph with n vertices $\ge 2^{n-1}-1$ ※

9. TopoIterator

```
vector<int> adj_list[105];  //input the graph with adjancy list
int in_degree[105]; // calculate the indegree in advance
int n;  // n nodes in the DAG

class TopoIterator {
public:
    void topo_sort() {
        queue<int> q;
        vector<int> order;
        for (int i = 0; i < n; i++) {
            if (in_degree[i] == 0)  q.push(i);
        }
        while (q.size()) {
            int tmp = q.front();
            q.pop();
            order.push_back(tmp);
            for (int i = 0; i < adj_list[tmp].size(); i++) {
                int next = adj_list[tmp][i];
                in_degree[next]--;
                if (in_degree[next] == 0) {
                    q.push(next);
                }
            }
        }
        for (int i = 0; i < order.size(); i++) {
            cout << order[i] << " ";
        }
        cout << "\n";
    }
};
```

10. ShortestPath

1) If we want to get ShortestPath(4) ,we can only get 4 to 6 and can't get the others shortest path. Similarly, we can't get any ShortestPath of 6 to other vertices because of the directed relationship between the vertices.
2) 0 -> 2 -> 1 -> 3 -> 4 -> 6 or 0 -> 2 -> 1 -> 3 -> 5 -> 6 distance = 8