TUTORIAL-05

Q1. What is difference between DFS and BFS. Write the applications of both algorithms.

Ans:

BFS	DFS
• BFS stands for Breadth first search	• DFS stands for Depth first search
• BFS uses queue to find the shortest path	• DFS uses stack to find the shortest path
• BFS is better when target is close to source	• DFS is more suitable for decision tree. As with one decision, we need to traverse further to argument decision if we reach conclusion.
• As BFS considers all neighbour so it is not suitable for decision tree used in puzzle games.	
• BFS is slower than DFS	• DFS is faster than BFS.

Applications of DFS -

(1) If we perform DFS on unweighted graph, then it will create minimum spanning tree for all pair shortest path tree.

(2) We can detect cycle in a graph using DFS if we get one back edge during BFS then there must be one edge.

(3) Using DFS we can find path between two given vertices u & v.

Applications of BFS -

(1) Like DFS, BFS may also used for detecting cycles in a graph.

(2) Finding shortest path and minimal spanning tree in unweighted graph.

(3) Finding a route through GPS navigation system with minimum number of crossings.

(4) In networking finding a route for packet transmission.

(5) In building the index of search engine transist

Q2. What data structures are used to implement BFS and DFS and why?

Ans:- BFS (Breadth first Search) use queue data structure and DFS (Depth first Search) uses stack data structures.

A queue (FIFO - First In first Out) data structure is used by BFS. you mark any node in the graph as root and start traversing the data from it. BFS traverse all the nodes in the graph and keep dropping them as completed. BFS visits an adjacent unvisited node, mark it as done, and insert it into a queue.

DFS algorithm traverse a graph in depthward motion and uses a stack to remember to get the next vertex to start a search, when a dead end occurs in any iterations.

Q3:- What do you mean by sparse and dense graphs? which representation of graph is better for sparse and dense graphs?

Ans:- Sparse Graph:- A graph in which the number of edges is much less than the possible no. of edges.

Dense Graph :- A graph in which the no. of edges is close to the minimal no. of edges.

If the graph is sparse we should store it as a list of edges. Alternatively, if the graph is dense, we should store it as a adjacency Matrix.

Q:4 How can you detect a cycle in a graph using BFS and DFS?

Ans. The existence of a cycle is directed and undirected graph can be determined by whether depth-first search (DFS) finds an edge that points to an ancestors of the current vertex (it contains a back-cycles). All back edge which DFS skips over are part of cycles.

* Detect cycle in a directed Graph.

DFS can be used to detect a cycle in graph. DFS for a connected graph produce a tree. There is a cycle in a graph only if there is a back edge that is from a node to itself (self-loops). Or one of its ancestors in the tree produced by DFS, then the for a disconnected graph, Get the DFS forest as output to detect cycle check a cycle in individuals trees by checking back edge.
To detect a back edge, Keep track of vertices currently in the recursion stack of function for DFS traversal. If a vertex is reached that is already in the recursion stack, then there is a cycle in the tree. The edge that connects the current vertex to the vertex

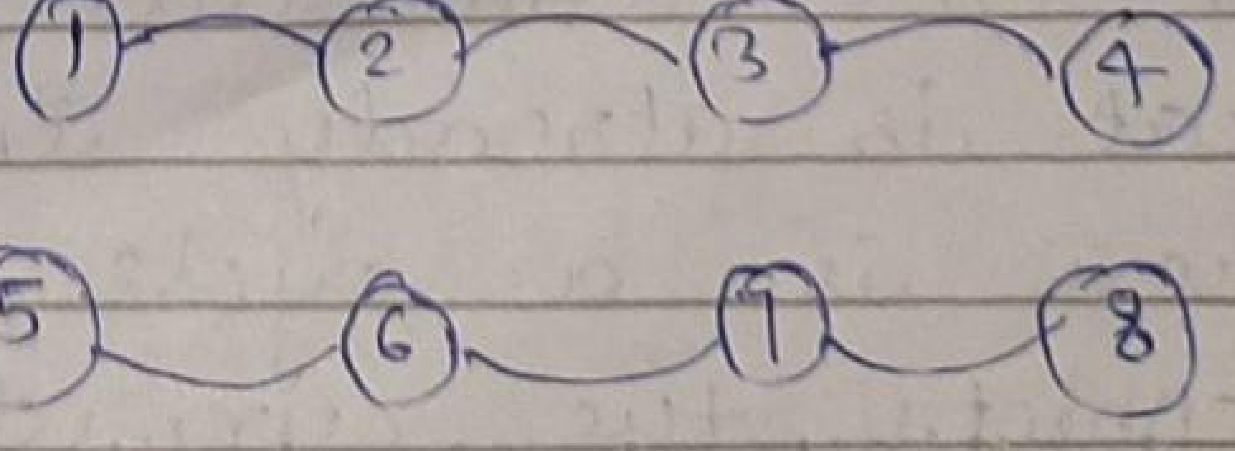
in the recursion stack is a back edge. Use rec. stack [ ] array to keep track of vertices in the rec. stack.

* Detect cycle in an undirected graph.
Run a DFS from every unvisited node. DFS can be used to detect a cycle in a graph. DFS for a connected graph produces a tree. There is a cycle in a graph only if there is a back edge present in the graph. A back edge is an edge that is joining a node to itself or one of its ancestor in the tree produced by DFS. To find back edge to any of its ancestor keep a visited array and if there is a back edge to any visited node then there is a loop and return true.

Q5. What do you mean by disjoint set data structure? Explain 3 operations along with examples, which can be performed on disjoint sets.

Ans. Disjoint set data structure :- It allows to find out whether the two element are in the same set or not efficiently.
The disjoint set can be defined as the subset where there is no common element between the two sets.

eg:- $S_1 = \{1, 2, 3, 4\}$
$S_2 = \{5, 6, 7, 8\}$



Operations performed:-

(i) find:- can be implemented by recursively a traverse the parent array untill we hit a node who is present to itself.

```
int find( int i) {
        if ( parent [i] == i) {
                return i;
        } else {
                return find(parent [i]);
        }}
```

(ii) Union:- It takes, as input, two elements and finds the representation of their sets using the finds operation and finally puts either one of the trees under the root node of the other tree, effectively merging the tree and the sets.

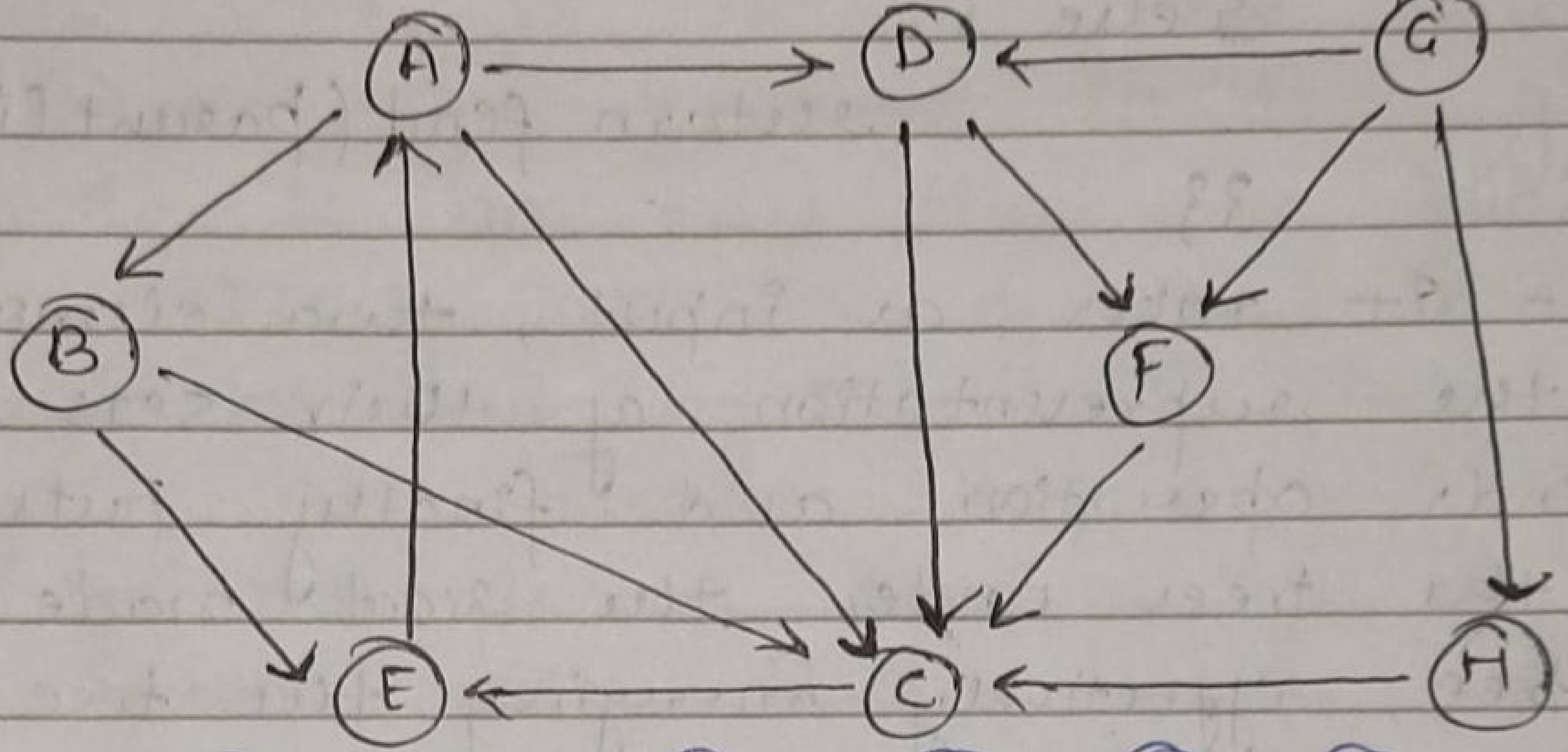
```
void union( int i , int j ) {
        int  irep = this. find (i);
        int  jrep = this. find (j);
        this. parent [irep] = jrep;
}
```

(iii) Path compression (Modification to find());
It speeds up the data structure by compressing the height of the trees. It can be achieved by interesting a small caching mechanism into find operation.

```
int find (int i)
{       if (parent[i]==i) {
                return i;
}
else {
```

```
            int result = find(parent[i]);
            parent[i] = result;
            return result;
        }
    }
```

Q.6 Run BFS and DFS on graph shown on right side

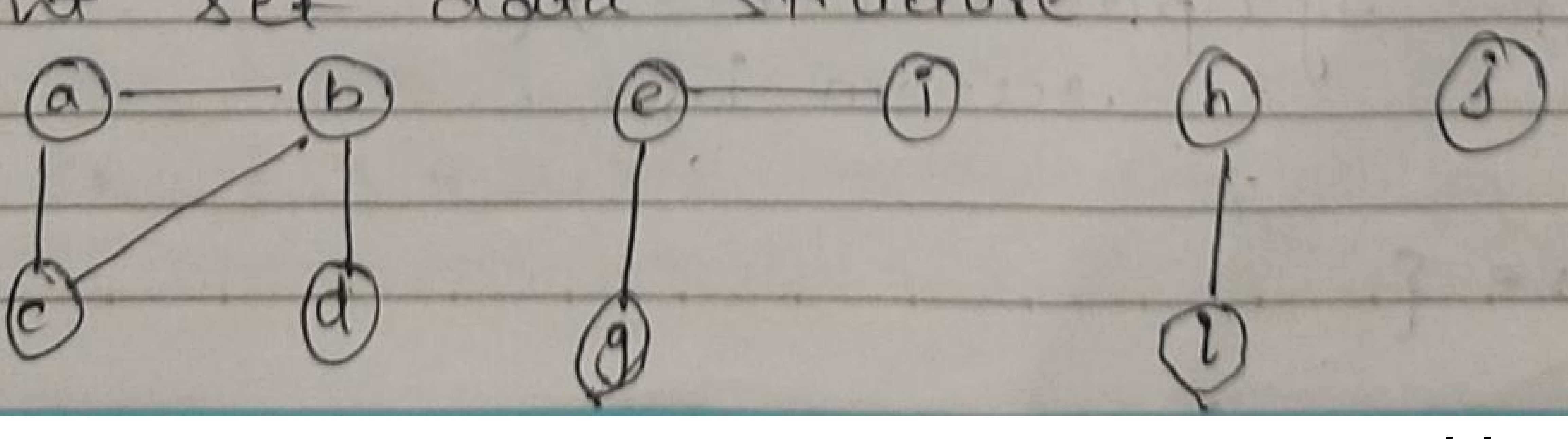


BFS :- (B) (E) (C) (A) (D) (F)

parent :- B B E A D
unvisited nodes :- G and H
path → B → E → A → D → F

DFS :-
Node processed :- B B C E A D E
stack :- B CE EE AE DE FE E
path :- B → C → E → A → D → F

Q.7 Find out the no. of connected components and vertices in each component using disjoint set data structure.



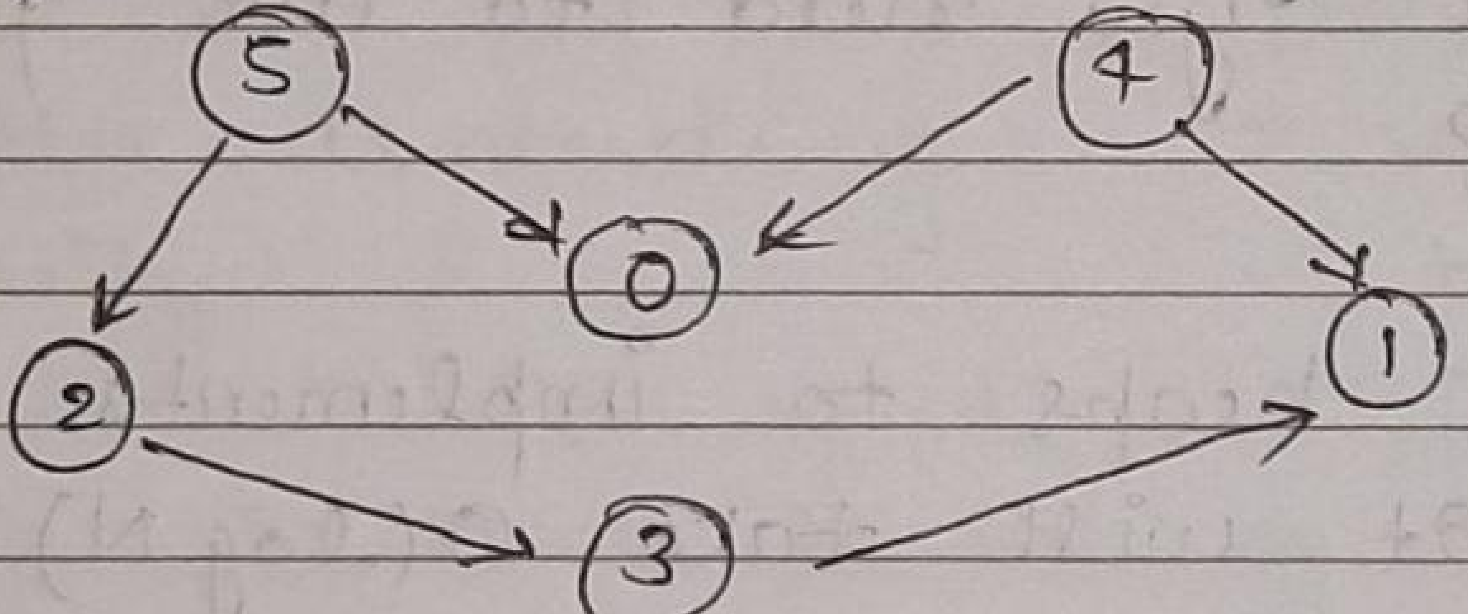
Ans:- V = {a} {b} {c} {d} {e} {f} {g} {h} {i} {j}

E = {a, b} {a, c} {b, c} {b, d}, {e, f}, {e, g}, {h, i}

(a, b) - {a, b} {c} {d} {e} {f} {g} {h} {i} {j}
(a, c) - {a, b, c} {d} {e} {f} {g} {h} {i} {j}
(b, c) - {a, b, c} {d} {e} {f} {g} {h} {i} {j}
(b, d) - {a, b, c, d} {e} {f} {g} {h} {i} {j}
(e, f) - {a, b, c, d} {e, f} {g} {h} {i} {j}
(e, g) - {a, b, c, d} {e, f, g} {h} {i} {j}
(h, i) - {a, b, c, d} {e, f, g} {h, i} {j}

No. of connected component = 3 Ans

Q.8. Apply topological storing and DFS on graph having vertices from 0 to 5.



Adjacency list
0 →
1 →
2 → 3
3 → 1
4 → 0, 1
5 → 2, 0

visited:-

0	1	2	3	4	5
f	f	f	f	f	f

stack (empty)

step 1:- Topological sort (0) visited [0] = true, list is empty, No more recursion call
stack $\boxed{0}$

step 2:- Topological sort (1), visited [1] = true. list is empty. No more recursion call
stack $\boxed{0 \mid 1}$

M T W T F S S
Page No.:
Date:
YOUVA

step 3:- Topological sort (2) visited [2] = true
Topological sort (3) visited [3] = true
'1' is already visited, No more recursionC
stack | 0 | 1 | 3 | 2 |

step 4:- Topological sort (4), visited [4] = true
'0' '1' are already visited, No more recursionC
stack | 0 | 1 | 3 | 2 | 4 |

step 5:- Topological sort (5), visited (5) = true
'2', '0' are already visited. No more rec. call
stack | 0 | 1 | 3 | 2 | 4 | 5 |

step 6:- Print all the elements of stack from top to bottom.
5, 4, 2, 3, 1, 0

Q9:- Heap data structure can be used to the implement priority queue? Name few graph algorithms where you need to use priority queue and why?

Ans:- We can use heaps to implements the priority queue. It will take O(log N) time to insert and delete each element in the priority queue. Based on heap structure, priority queue also has two types:- max priority and min priority.

Some algorithms where we need to use the priority queue.

(i) Dijkshtra Shortest path Algorithm:- When the path is sorted in the form of adjacency list or matrix, priority queue can be used entract minimum efficiently when implementing

Dijkstra's algorithm.

(ii) Prim's algorithm :- It is used to implement prim's algorithm to store key of nodes and extract minimum key node at every step.

Data compression :- It is used in Huffman's code which is used to compress data.

Q10:- What is the difference between Max and Min Heap?

Ans:-

Min Heap	Max Heap
→ In a min heap the key present at root must be less than or equal to among the keys present at all of its children.	→ In max heap the key present at root node must be greater than or equal to among keys present to all of its children.
→ minimum key element present at the root	→ maximum key element present at the root
→ uses the ascending priority	→ uses the descending priority
→ In a construction of a min-heap, smallest element has priority.	→ In a construction of max-heap, largest element has priority.
→ the smallest element is the first to be popped from heap.	→ the largest element is the first to be popped from heap.