자료구조론 CC343_2207

Reading assignment 12

경기대학교 컴퓨터공학부

201511837 이상민

Chapter 12 Review Questions

1. Define a binary heap.

A binary heap is a that takes the form of a. Binary heaps are a common way of implementing.

The binary heap was introduced by in 1964, as a data structure for.

A binary heap is defined as a binary tree with two additional constraints:

Shape property: a binary heap is a; that is, all levels of the tree, except possibly the last one (deepest) are fully filled, and, if the last level of the tree is not complete, the nodes of that level are filled from left to right.

Heap property: the key stored in each node is either greater than or equal to (≥) or less than or equal to (≤) the keys in the node's children, according to some.

Heaps where the parent key is greater than or equal to (≥) the child keys are called max-heaps; those where it is less than or equal to (≤) are called min-heaps. Efficient () algorithms are known for the two operations needed to implement a priority queue on a binary heap: inserting an element and removing the smallest or largest element from a min-heap or max-heap, respectively. Binary heaps are also commonly employed in the, which is an in-place algorithm because binary heaps can be implemented as an, storing keys in an array and using their relative positions within that array to represent child-parent relationships.

2. Differentiate between a min-heap and a max-heap.

Min-Heap – Where the value of the root node is less than or equal to either of its children.

Max-Heap – Where the value of the root node is greater than or equal to either of its children.

3. Compare binary trees with binary heaps.

heap은 높은 수준의 요소가 낮은 수준의 요소보다 크거나 작다는 것을 보장하는 반면 tree는 "왼쪽"에서 "오른쪽"으로의 순서를 보장합니다.

heap은 findMin / FifindMax(O(1))에서 더 효과적이고 tree는 모든 위치(O(logN))에서 더 우수합니다. 두 구조물에 대해 삽입은 O(logN)입니다.

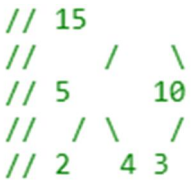
4. Explain the steps involved in inserting a new value in a binary heap with the help of a suitable example.

Insert: 삭제를 위해 아래에서 설명한 것과 유사한 접근 방식에 따라 요소를 heap에 삽입할 수 있습니다. 그 목적은 다음과 같습니다.

- 먼저 새 요소를 저장할 수 있도록 heap의 크기를 1만큼 늘리십시오.
- heap의 끝에 새 요소를 삽입합니다.
- 새로 삽입된 이 요소는 부모에게 heap의 속성을 왜곡할 수 있습니다. 따라서 heap의 속성을 유지하려면 상향식 접근 방식에 따라 새로 삽입된 이 요소를 Heapifying 하십시오.

Example:

to 15 insert



5. Explain the steps involved in deleting a value from a binary heap with the help of a suitable example.

delete: heap의 중간 위치에서 요소를 삭제하면 비용이 많이 들기 때문에 마지막 요소로 삭제할 요소를 교체하고 heap의 마지막 요소를 삭제하기만 하면 됩니다.

마지막 요소에서 삭제할 루트 또는 요소를 교체합니다.

heap에서 마지막 요소를 삭제합니다.

이제 마지막 요소가 루트 노드의 위치에 배치됩니다. 따라서 heap 속성을 따르지 않을 수 있습니다. 따라서 루트 위치에 배치된 마지막 노드를 지혈한다.

Example :

Root delete to output : 5 4 3 2

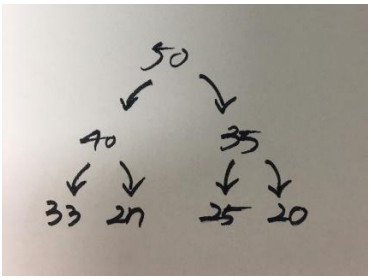
6. Discuss the applications of binary heaps.

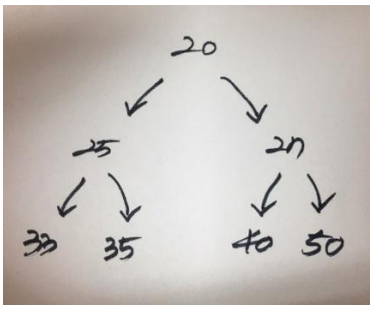
힙 데이터 구조는 일반적으로 힙 포트를 사용하여 학습합니다. Heapsort알고리즘은 Quicksort가 실제로 더 낫기 때문에 제한적으로 사용됩니다. 그럼에도 불구하고 힙 데이터 구조 자체는 엄청나게 사용됩니다. 다음은 Heapsort이외의 다른 용도입니다.

우선 순위 대기 열: 우선 순위 대기 열은 삽입(), 삭제()및 extractmax(), decreaseKey()작업을 O(logn)시간으로 지원하므로 이진 힙을 사용하여 효율적으로 구현할 수 있습니다. Binomoial힙과 피보나치 힙은 이진 힙의 변형입니다. 이러한 차이는 이진 힙의 O(logn)시간(n)단위로 결합을 수행 시간은 O(n)작업입니다. 힙 임시 우선 순위 대기 열은 Prim의 알고리즘 및 Dijkstra알고리즘과 같은 그래프 알고리즘에 사용됩니다.

7. Form a binary max-heap and a min-heap from the following sequence of data:

50, 40, 35, 25, 20, 27, 33.

Max heap: 

Min heap: 

8. Heaps are excellent data structures to implement priority queues. Justify this statement.

A priority queue must at least support the following operations:

is_empty: check whether the queue has no elements.

insert_with_priority: add an to the with an associated priority.

pull_highest_priority_element: remove the element from the queue that has the highest priority, and return it.

This is also known as "pop_element(Off)", "get_maximum_element" or "get_front(most)_element".

Some conventions reverse the order of priorities, considering lower values to be higher priority, so

this may also be known as "get_minimum_element", and is often referred to as "get-min" in the literature.

This may instead be specified as separate "peek_at_highest_priority_element" and "delete_element" functions, which can be combined to produce "pull_highest_priority_element".

In addition, (in this context often called find-max or find-min), which returns the highest-priority element but does not modify the queue, is very frequently implemented, and nearly always executes in time. This operation and its O(1) performance is crucial to many applications of priority queues.

More advanced implementations may support more complicated operations, such as pull_lowest_priority_element, inspecting the first few highest- or lowest-priority elements, clearing the queue, clearing subsets of the queue, performing a batch insert, merging two or more queues into one, incrementing priority of any element, etc.

One can imagine a priority queue as a modified , but when one would get the next element off the queue, the highest-priority element is retrieved first.

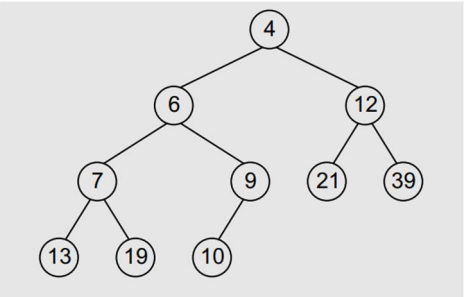
Stacks and queues may be modeled as particular kinds of priority queues. As a reminder, here is how stacks and queues behave:

stack – elements are pulled in -order (e.g., a stack of papers)

queue – elements are pulled in -order (e.g., a line in a cafeteria)

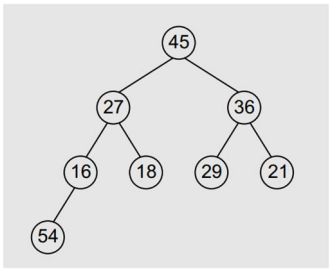
In a stack, the priority of each inserted element is monotonically increasing; thus, the last element inserted is always the first retrieved. In a queue, the priority of each inserted element is monotonically decreasing; thus, the first element inserted is always the first retrieved.

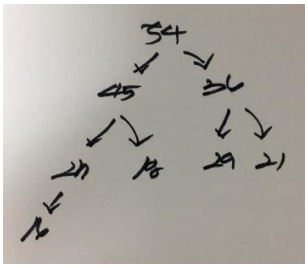
21. Consider the figure given below and state whether it is a heap or not.



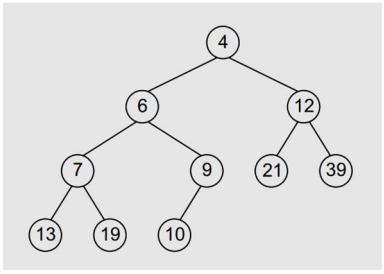
Min heap

22. Reheap the following structure to make it a heap.



to convert


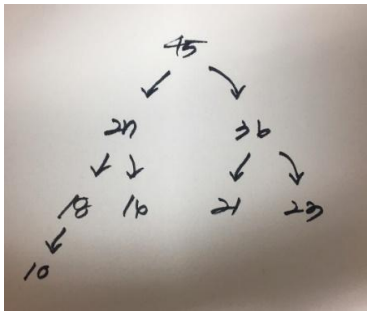
23. Show the array implementation of the following heap.



-	4	6	12	7	9	21	39	13	19	10	-

24. Given the following array structure, draw the heap.

45	27	36	18	16	21	23	10



Also, find out

(a) the parent of nodes 10, 21, and 23, and :

21, 23의 상위 부모 : 36

10의 상위 노드 : 18

(b) index of left and right child of node 23. : none.

Multiple-choice Questions: 1. (b) 2. (a) 3. (c) 4. (d)

True or False: 1. True 2. False 3. True 4. True 5. False

Fill in the Blanks: 1. 2i+1 2. Priority queues 3. Partially ordered trees 4. Min heap 5. Root

Chapter 13 Review Questions

1. Explain the relationship between a linked list structure and a digraph.

Directed Graph : 방향성이 있는 그래프이다.

Adjacency List를 통해서 Adjacency Matrix의 메모리 낭비 문제를 없앨 수 있다.

Adjacency List의 구현 한가지 방법으로, Linked List의 배열을 만드는 것이다. 즉, N개의 Linked List를 만드는 것이다. 여기서, N은 노드의 갯수이다. 배열 한칸을 하나의 노드라고 보면된다.

Linked List안에 있는 원소들은 해당 노드에서 연결된 Edge를 표현한다.

예를 들어, 3개의 노드가 존재하고 1번과 3번 노드 사이에 Edge가 존재한다.

0 : 3

1 :

2 :

0, 1, 2는 각각의 노드이고, 인덱스 때문에 1이 감소된 것을 볼 수 있다.

3을 통해 (1,3)사이에 Edge가 존재함을 알 수 있다.

2. What is a graph? Explain its key terms.

Define a graph G = (V, E) by defining a pair of sets:

- ✓ V = a set of vertices
- ✓ E = a set of edges

Edges:

- ✓ Each edge is defined by a pair of vertices
- ✓ An edge connects the vertices that define it
- ✓ In some cases, the vertices can be the same

Vertices:

- ✓ Vertices also called nodes
- ✓ Denote vertices with labels

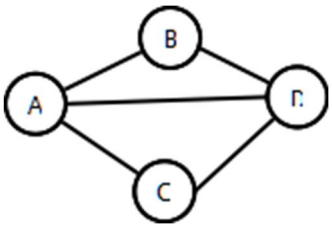
Representation:

- ✓ Represent vertices with circles, perhaps containing a label

- ✓ Represent edges with lines between circles

Example:

- ✓ V = {A,B,C,D}
- ✓ E = {(A,B),(A,C),(A,D),(B,D),(C,D)}



3. How are graphs represented inside a computer's memory? Which method do you prefer and why?

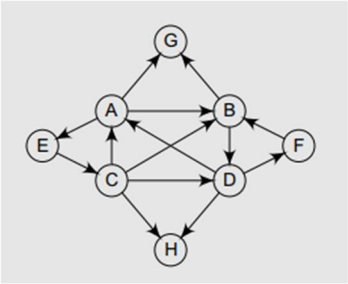
Graph is a data structure that consists of following two components:

1. A finite set of vertices also called as nodes.

2. A finite set of ordered pair of the form (u, v) called as edge. The pair is ordered because (u, v) is not same as (v, u) in case of a directed graph(di-graph). The pair of the form (u, v) indicates that there is an edge from vertex u to vertex v. The edges may contain weight/value/cost.

Graphs are used to represent many real-life applications: Graphs are used to represent networks. The networks may include paths in a or telephone network or circuit network. Graphs are also used in social networks like linked In, Facebook. For example, in Facebook, each person is represented with a vertex

4. Consider the graph given below.



(a) Write the adjacency matrix of G: none.

(b) Write the path matrix of G:

AG, BG, CAG, CDFBG, CDAG, CDABG, CBG, DFBG, DAG, DABG, ECAG, ECDFBG, ECDAG, ECDABG, ECBG, FBG, FBDAG, FBDABG

(c) Is the graph biconnected? No.

(d) Is the graph complete? Yes.

(e) Find the shortest path matrix using Warshall's algorithm.

	A	B	C	D	E	F	G	H
A	0	1	1	1	1	1	1	1
B	1	0	1	1	1	1	1	1
C	1	1	0	1	1	1	1	1
D	1	1	1	0	1	1	1	1
E	1	1	1	1	0	1	1	1
F	1	1	1	1	1	0	1	1
G	0	0	0	0	0	0	0	1
H	0	0	0	0	0	0	0	0

5. Explain the graph traversal algorithms in detail with example.

Depth-first search: A depth-first search (DFS) is an algorithm for traversing a finite graph. DFS visits the child vertices before visiting the sibling vertices; that is, it traverses the depth of any particular path before exploring its breadth. A stack (often the program's via) is generally used when implementing the algorithm.

The algorithm begins with a chosen "root" vertex; it then iteratively transitions from the current vertex to an adjacent, unvisited vertex, until it can no longer find an unexplored vertex to transition to from its current location. The algorithm then along previously visited vertices, until it finds a vertex connected to yet more uncharted territory. It will then proceed down the new path as it had before, backtracking as it encounters dead-ends, and ending only when the algorithm has backtracked past the original "root" vertex from the very first step.

DFS is the basis for many graph-related algorithms, including and.

Pseudocode

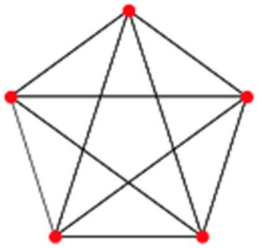
- ✓ Input: A graph G and a vertex v of G.
- ✓ Output: A labeling of the edges in the connected component of v as discovery edges and back edges.

Breadth-first search: A breadth-first search (BFS) is another technique for traversing a finite graph. BFS visits the sibling vertices before visiting the child vertices, and a is used in the search process. This algorithm is often used to find the shortest path from one vertex to another.

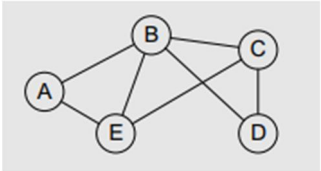
Pseudocode

- ✓ Input: A graph G and a vertex v of G.
- ✓ Output: The closest vertex to v satisfying some conditions, or null if no such vertex exists.

6. Draw a complete undirected graph having five nodes.

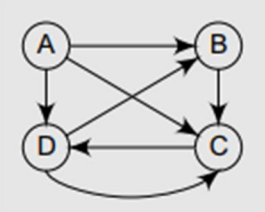


7. Consider the graph given below and find out the degree of each node.



DEG(A): 2, DEG(B): 4, DEG(C): 3, DEG(D): 2, DEG(E): 3

8. Consider the graph given below. State all the simple paths from A to D, B to D, and C to D. Also, find out the in-degree and out-degree of each node. Is there any source or sink in the graph?



A -> D: AD, ACD, ABCD

B -> D: BCD

C -> D: CD

A is out-degree: 3, in-degree: 0

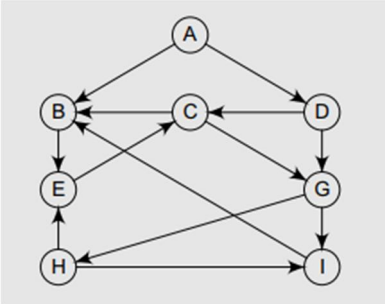
B is out-degree: 1, in-degree: 2

C is out-degree: 1, in-degree: 3

D is out-degree: 2, in-degree: 2

Sink not exists.

9. Consider the graph given below. Find out its depth-first and breadth-first traversal scheme.



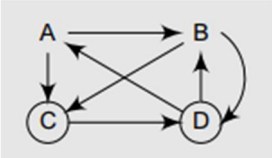
DFS: A E B C D I H G

BFS: A B D C E G H I

10. Differentiate between depth-first search and breadth-first search traversal of a graph.

	BFS	**DFS**
1	BFS stands for Breadth First Search.	DFS stands for Depth First Search.
2	BFS(Breadth First Search) uses Queue data structure for finding the shortest path.	DFS(Depth First Search) uses Stack data structure.
3	BFS can be used to find single source shortest path in an unweighted graph, because in BFS, we reach a vertex with minimum number of edges from a source vertex.	In DFS, we might traverse through more edges to reach a destination vertex from a source.
4	BFS is more suitable for searching verteces which are closer to the given source.	DFS is more suitable when there are solutions away from source.
5	BFS considers all neighbors first and therefore not suitable for decision making trees used in games or puzzles.	DFS is more suitable for game or puzzle problems. We make a decision, then explore all paths through this decision. And if this decision leads to win situation, we stop.
6	The Time complexity of BFS is O(V + E), where V stands for vertices and E stands for edges.	The Time complexity of DFS is also O(V + E), where V stands for vertices and E stands for edges.

26. Consider the graph given below and show its adjacency list in the memory.



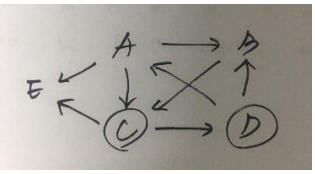
A -> B -> C

B -> C -> D

C -> C -> D

D -> A -> B -> D

27. Consider the graph given in Question 26 and show the changes in the graph as well as its adjacency list when node E and edges (A, E) and (C, E) are added to it. Also, delete edge (B, D) from the graph.



A -> B -> C -> E

B -> C

C -> C -> D -> E

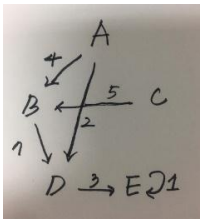
D -> A -> B -> D

B -> none.

28. Given the following adjacency matrix, draw the weighted graph.

$$\begin{pmatrix} 0 & 4 & 0 & 2 & 0 \\ 0 & 0 & 0 & 7 & 0 \\ 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

to convert



Multiple-choice Questions: 1. (b) 2. (c) 3. (b) 4. (a) 5. (b)

True or False: 1. False 2. False 3. True 4. False

Fill in the Blanks: 1. An isolated node 2. Terminate 3. Bit matrix or Boolean matrix 4. Cycle 5. Multi-graph