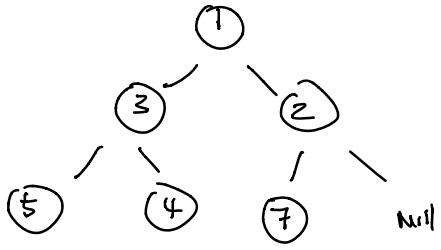
min / max heap ( Priority Queue ).




| index | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| | 1 | 3 | 2 | 5 | 4 | 7 |

binary heap:

# 1. complete binary tree

# 2. Every nodes < its descendent.

if root is MIN → MIN HEAP.

root is MAX → MAX HEAP.

implemented as an unsorted array.

# 1. index of left child = $i \times 2 + 1$

# 2. index of right child = $i \times 2 + 2$

# 3. index of parent = $(i - 1) / 2$.

Operations:

# 1. insert: Time: $O(\log n)$. - percolate Up.

#2. update: keep heap after insertion. Time ( $\log n$ ).

# 3. get / top: Time: $O(1)$.

# 4. pop: Remove top. Time: ( $\log n$ ). - percolate down

# 5. heapify. turn unsorted arr into a heap. Time: $O(n)$.

Q1: Find K smallest from unsorted arr. of size n.

Soln 0: sort → $O(n \log n)$.

Soln 1:

#1. heapify: $O(n)$

#2. Pop the 1st k elements: $O(k \cdot \log n)$.

Total time: $O(n + k \cdot \log n)$.

Soln 2: Online algorithms.

#1. heapify the 1st k elements → MAX Heap of size k.

#2. Iterate over the remaining $(n-k)$.

Compare the new elements w/ current top.

a. new >= top — ignore.

b. new < top — update (top → new).

Time: $(n-k) \log k$.

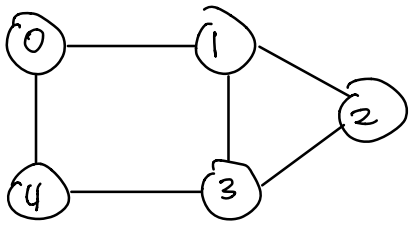
Total Time: $O(k + (n-k) \log k)$.

| Compare | Soln 1 $(n + k \log n)$ | Soln 2 $(k + (n-k) \log k)$ |
|---|---|---|
| $k \ll n$ | $O(\text{Constant} \cdot n)$ | $O(n \log k)$. |
| $k \sim n$. | $O(n \log n)$ | $O(n \log n)$ |

Hard to say

Soln 3: Quick Select (Omitted).

Graph:




1. Node / Vertex (State)
2. Edge / action
3. Directed vs. Undirected
4. Weighted & no cost
5. Dense vs. Sparse.

## 6. Representation:

### 6.1 Adjacency Matrix.

- pros: easy to implement. O(1) removal. Queries. O(1)
- cons: more space $O(n^2)$. if sparse, waste of space.

### 6.2. Adjacency List.

- pros. Space: O(V+E). adding is easier
- cons: Time complexity: O(V) to check whether there is an edge from a node to the other

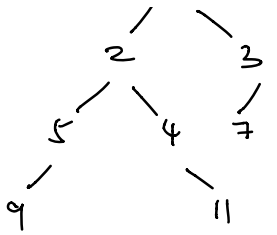
### 6.3. HashMap. <key – node. value – set of node>

```
GraphNode
    int value;
    String name;
    List<GraphNode> neighbors;
```

Algorithms related to Graph.

# BFS. Queue.

1 expand (1) → generate (2) & (3).




Order: BFS - 1 2 3 5 4 7 9 11

Data Structure: Queue.

- initial state: put root -> Q
- Expand a node s.
- Generate its neighbors.
- Termination: Q is empty.
- Optionally deduplicates ( in graph not trees)
  - each node is expanded & generated once.

Q1. Print a binary Tree by level.

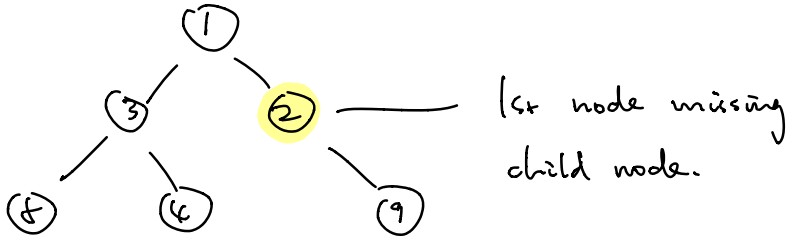
need to add new line

```
Queue<Node> q = new ArrayDeque<Node>();
q.offer(root);
while (!q.isEmpty())
    int size = q.size();
    for (int i=0; i< size ; i++)
        Node n = q.poll();
        if has left. offer left.
        if has right. offer right
        Print current. node + space.
    Print Newline
```

Q2. Bipartie: whether a graph can be divided into 2 groups. Such that nodes from the same group cannot be connected by direct edge.

Assign colors & do BFS / DFS.

Q3. If a tree is a complete tree?




1st node missing child node.

For case of node 2:

#1. if node 2 miss Left → return false

#2. if node 2 miss Right only

When we detected a node that has missed one child, set a found_node_missing_child flag.

After that, if we found any other node that can

generate other nodes. return false.

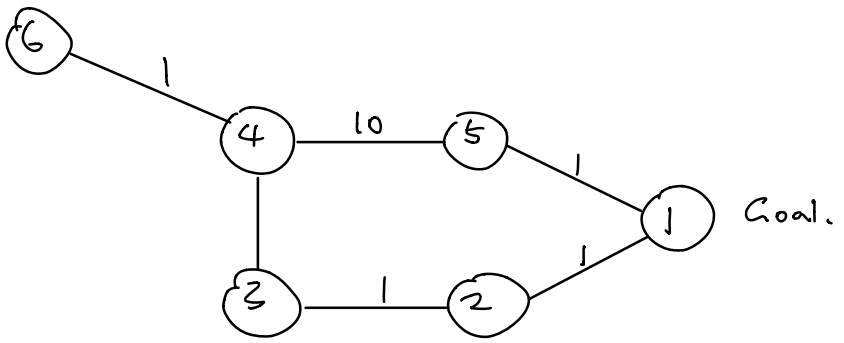
## ✡ BFS-2 ( Best first Search ).

Dijkstra's Algorithm. ( runtime efficiency improvement : A* algorithm ).

# 1. usage : shortest path cost . to any other nodes.

# 2. Example : from Beijing to other cities' shortest path

# 3. Data Structure : priority Queue.




Algorithm :

initialize PQ . add <4 , 0>.

For each step ( PQ not empty ).

Expand:

dequeue the node x (smallest cost) — O( log n ).

dedup : if popped before . STOP generating.

Generate:

O(E) —— enqueue all neighbors < neighbor . Total cost cost(x) + cost(nei) >

if total cost >= cost(nei) . skip enqueue.

else enqueue or update O(logn).

Termination: PQ Empty.

Properties:

1. one node can be expanded once only
2. one node can be generated once or more
3. all cost of nodes that are expanded are monotonically non-decreasing. (cost can only be higher).
4. Time Complexity: $O(E \log V)$
5. when a node is popped out for expansion, its cost is fixed which is equal to the shortest distance from the start node.

Variants:

Termination (Shortest Path): target node is expanded

☆. k-th-smallest in a sorted matrix (ascending order).
Discover the iteration pattern. and info needed.