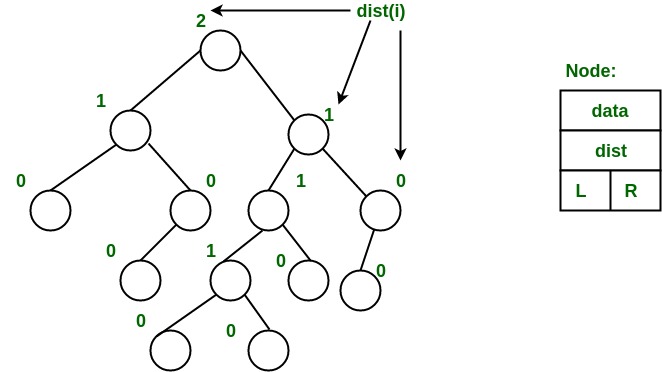
heap

value of a node must be larger or smaller than the 2 children.

make: insert: delete min / max:

leftist heap

A type of heap that can be merged in time.

Every node has key[u] like a heap and dist[u], which stores the shortest distance from u to a descendant of u with less than 2 children.

Important property of leftist heap: dist[u.right] <= dist[u.left]

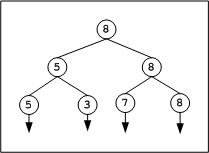
Every subtree is also a leftist heap.

Since dist[u.right] <= dist[u.left], dist[u] = dist[u.right] + 1

Therefore, the path from root to the rightmost leaf is the shortest path from root to a leaf. Moreover, the length of that path is for a leftist tree with nodes.

How to merge:

1. if (key[root(a)] > key[root(b)]) swap(a, b)
2. a.right = merge(a.right, b)
3. if (dist[a.right] > dist[a.left]) swap(a.right, a.left)
4. if (a.right == null) dist[a] = 0 else dist[a] = dist[a.right] + 1

How to insert:

1. Merge the tree and the single node

How to delete min / max:

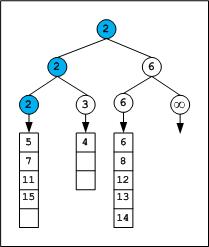
1. Remove the min / max node
2. Merge the 2 subtrees

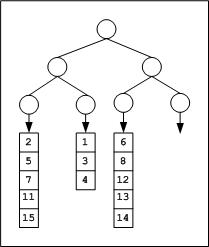
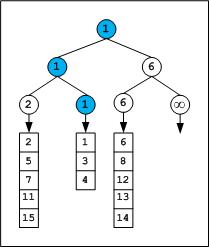
tournament tree

The leafs represent players, and the root node represents the winner. Can construct the tournament tree with comparisons.

To find the second best, just check the players who lost to the best player. Since the height of the tree is , you just need comparisons.

Total: can find second best player with comparisons

Find the first k smallest element of m sorted arrays

Tournament the first element of each array. Then send the next element from the array of winning element. Repeat

time complexity:

Some problems about heap

Given an array where each element is at most k away from its target position, sort the array efficiently.

Use a heap of size k, then remove the root and add nodes repeatedly.

Time complexity:

Find the lexicographically smallest / largest derangement of an array.

Use a min / max heap to store all elements, than check if the root element can be put in ith position, if not then put the second best one in. Also check if need to swap last two elements. Time complexity:

std::priority\_queue<int> pq;

for (int i = 0; i < N; i++)

    pq.push(seq[i]);

for (int i = 0; i < N; i++) {

    int d = pq.top();

    pq.pop();

    if (d != seq[i] || i == N - 1) {

        res[i] = d;

    } else {

        res[i] = pq.top();

        pq.pop();

        pq.push(d);

}

}

if (res[N - 1] == seq[N - 1]) {

    res[N - 1] = res[N - 2];

    res[N - 2] = seq[N - 1];

Find the k maximum a­i + bj from 2 arrays a and b of size n

Sort a and b. Then, initialize a heap with tuple with 1 element: an-1 + bn-1, n – 1, n – 1 (sort by first element)

Repeat for k times: output the max (a­i + bj, i, j) and remove max from heap, then insert (ai + bj-1, i, j – 1) and (ai-1 + bj, i – 1, j) into the heap. check for repeat

Time complexity:

Find the kth largest continuous subarray sum of an array a with size n

Make prefix sum array. Then, push continuous subarray sums into a min heap until it has k elements. When inserting a new element, if it is larger than the current minimum, pop and insert, else skip to the next element. The final minimum is the kth largest.

Time complexity:

backup (APIO 2017)

<https://www.iarcs.org.in/inoi/online-study-material/problems/backup.php>

Given n points on the x axis and k lines, find a way to pair 2k points together such that the total distance between pairs are minimized. A point can only be at most in 1 pair, or the 2k chosen point must be distinct.

Obviously, you must connect adjacent points. Therefore, the problem becomes: given n – 1 distances, pick k of them such that the sum is minimized and no two adjacent distances are chosen.

Greedily choose the minimum distance. However, what if the optimal solution takes the adjacent distances?

To solve this problem, after choosing the minimum (dm), we can create a virtual edge with distance dm-1 + dm+1 – dm (or something else if at the ends) so we can later choose the other option if it is more optimal. The virtual edge simply means to undo choosing d­m and choosing dm-1 and dm+1.

Solution: maintain a minimum heap / priority queue of tuples and push all (ai+1 - ai, i, i + 1) into it. Then, pop the one with minimum distance, add it to the answer, and insert (appropriate distance, i – 1, j + 1) into the heap. Also maintain a Boolean array used[n] and update used[i] and used[j] to true after adding it to the answer. Don’t add if used[i] or used[j] are already true.

Time complexity:

Problems

<https://codeforces.com/problemset/problem/722/D>