Chapter 10 Application of Classical Trees

Yonghui Wu

School of Computer Science, Fudan University

yhwu@fudan.edu.cn

Wechat: 13817360465

Chapter 10 Application of Classical Trees

- 10.1 Binary Search Trees
- 10.2 Binary Heap
- 10.3 Huffman Trees

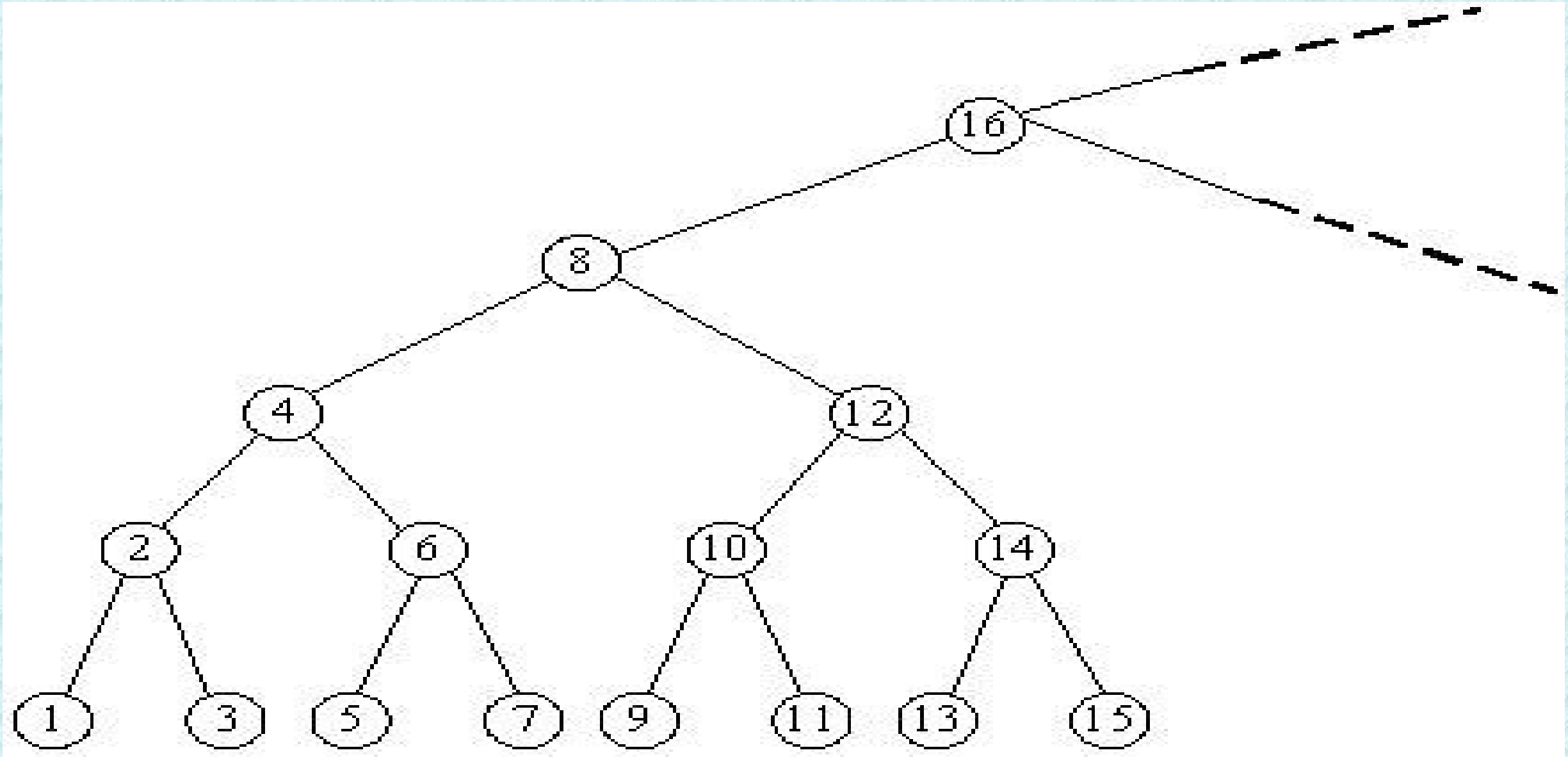
10.1 Binary Search Trees

- Binary search trees are binary trees which have the following properties.
  - Every node has a key.
  - The key values in the left subtree of the root are smaller than the key value in the root. The key values in the right subtree of the root are larger than the key value in the root.
  - The left and right subtree of the root are also binary search trees.
  - Inorder traversal for a binary search tree produces an increasing sequence.

10.1.1 BST

- **Source: POJ Monthly, Minkerui**
- **IDs for Online Judge: POJ 2309**

- Consider an infinite full binary search tree, the numbers in the nodes are 1, 2, 3, ... In a subtree whose root node is *X*, we can get the minimum number in this subtree by repeating going down the left node until the last level, and we can also find the maximum number by going down the right node. Now you are given some queries as "What are the minimum and maximum numbers in the subtree whose root node is *X*?" Please try to find answers for these queries.


16
8
4
12
2
6
10
14
1
3
5
7
9
11
13
15

- **Input**
- In the input, the first line contains an integer N, which represents the number of queries. In the next N lines, each contains a number representing a subtree with root number X ($1 \leq X \leq 2^{31}-1$).
- **Output**
- There are N lines in total, the i-th of which contains the answer for the i-th query.

Analysis

- Based on the problem description, if a root number X is odd, the root must be a leaf. If a root number X is even, and X divided by 2^k is odd, that is, $X \% 2^k=0$ and $X \% 2^{k+1}\neq 0$; the height of the subtree with root number X is k, and there are 2^{k+1}-1 nodes in the subtree.

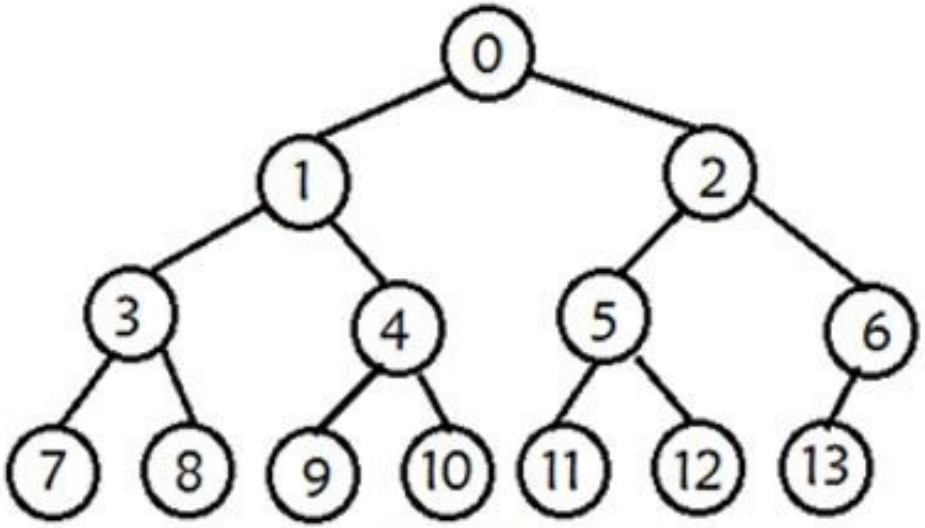
- Based on properties of the infinite full binary search tree and the rule of nodes' numbers, for a subtree with root number X (X % 2^k=0 and X % 2^{k+1}≠0), the following properties hold:
  - The minimum number *min* and maximum number *max* in the subtree must be odd.
  - There are 2^k-1 nodes in its left and right subtree respectively.
  - The interval of nodes' numbers in the left subtree is [*min*, X-1], and the interval of nodes' numbers in the right subtree is [X+1, *max*]; where *min*=X-2^k+1 and *max*=X+2^k-1.

10.2 Binary Heaps

- A binary heap is a complete binary tree which the value in each node is greater (less) than those in its children (if any). If the value in each node is greater than those in its children (if any), we call it a max heap. If the value in each node is less than those in its children (if any), we call it a min heap.
- Obviously, in a max heap the value in the root is maximal; and in a min heap the value in the root is minimal.

- A binary heap can be used to store a priority queue.
- In a priority queue, each element has a priority.
- In a min priority queue the search operation finds the element with minimum priority, while the deletion operation deletes this element.
- In a max priority queue the search operation finds the element with maximum priority, while the deletion operation deletes this element.
- If a linear list is used to represent a priority queue, the time complexity finding an element with minimum (or maximum) priority is O(n). Using a binary heap can improve the efficiency.

A binary heap is a complete binary tree. At the bottom level, nodes are filled from left to right. Therefore elements in a binary heap are stored in an array *heap*[0..*n*-1], where the parent of *heap*[*i*] is *heap*[$\left\lfloor \frac{i-1}{2} \right\rfloor$], and its left child and right child are *heap*[2**i*+1] and *heap*[2**i*+2] respectively; $1 \le i \le \left\lfloor \frac{n-1}{2} \right\rfloor$. If $i > \left\lfloor \frac{n-1}{2} \right\rfloor$, node *i* is a leaf.



The index for a heap

Insertion operation in a min heap

- The element is inserted at the end of the queue (rear).
- The position of the element is adjusted.
  - The inserted element's value is compared with its parent's value. If its value is less than its parent's value, then the two values' positions are exchanged.
- The process repeats until the inserted value isn't less than its parent's value or the inserted element becomes the root.
- Obviously, such a complete binary tree is a min heap.

- int *k* = ++*top*; // insert the node at rear
- *heap*[*k*] = inserted element;
- while (*k* > 0) { // adjust the position
- int *t* = (*k*-1) / 2; // parent's position *t*
- if (*heap*[*t*]>*heap*[*k*]) {
- *heap*[*t*] and *heap*[*k*] exchange;
- *k* = *t*;
- } else
- break;
- }

Deletion operation in a min heap

- When an element is to be deleted from a min heap, the root of the heap is deleted, and the heap is adjusted.
  - That is, the value in *heap*[0] is deleted, *heap*[0]=*heap*[*n*-1], and then array *heap*[0…*n*-2] is adjusted to become a min heap. The process is as follow. Initially *k*=0.
  - Suppose *heap*[*t*] is the minimal child for heap[*k*];
  - If *heap*[*k*]>*heap*[*t*], then *heap*[*k*] and *heap*[*t*] exchange, *k*=*t*, and continue to downward adjust from *k*; else the complete binary tree is a min heap.

```
if (top) {       // the heap isn't empty
      return the root of the heap heap[0];
       int k =0;
       heap[k] = heap[top--];  // the element at the rear is put the root of the heap, the length -1
       while ((k * 2+1) <= top) {    // Adjust the heap
              int t = k * 2+1;          // the minimal child t
              if (t < top && (heap[t+1] < heap[t])
                     ++t;
              if (heap[k]>heap[t]) {    // Adjust
                     heap[k] and heap[t] exchange;
                     k = t;
              } else
                     break;
       }
  } else
      output “the heap is empty”;
```

10.2.1 Windows Message Queue

- **Source: Zhejiang University Local Contest 2006, Preliminary**
- **IDs for Online Judge: ZOJ 2724**

- Message queue is the basic fundamental of windows system. For each process, the system maintains a message queue. If something happens to this process, such as mouse click, text change, the system will add a message to the queue. Meanwhile, the process will do a loop for getting message from the queue according to the priority value if it is not empty. Note that the less priority value means the higher priority. In this problem, you are asked to simulate the message queue for putting messages to and getting message from the message queue.

- **Input**
- There's only one test case in the input. Each line is a command, "GET" or "PUT", which means getting message or putting message. If the command is "PUT", there're one string means the message name and two integer means the parameter and priority followed by. There will be at most 60000 command. Note that one message can appear twice or more and if two messages have the same priority, the one comes first will be processed first. (i.e., FIFO for the same priority.) Process to the end-of-file.
- **Output**
- For each "GET" command, output the command getting from the message queue with the name and parameter in one line. If there's no message in the queue, output "EMPTY QUEUE!". There's no output for "PUT" command.

Analysis

- A priority queue is used to store a windows message queue.
  - If the priority queue is not empty, the message with the highest priority is gotten.
  - If two messages have the same highest priority, the one comes first will be gotten first.
- Because the number of messages will be at most 60000, a min heap can be used to store the priority queue.

- For each command:
  - If the current command is “GET”, the message at the top of the heap is moved out, the message at the rear is move to the top of the heap, the length of the heap decreases 1, and then the heap property is maintained.
  - If the current command is “PUT”, the message is inserted into the heap: the new message is added at the rear, the length of the heap increases 1, and then the heap property is maintained.

10.3 Huffman Trees

In a binary tree there are n leaves with weights, where the weight of the k-th leaf is w_k, and the length of the path from the root to the kth leaf is p_k. Then $w_k * p_k$ is the length of weighted path for the k-th leaf, $1 \le k \le n$. The binary tree whose the sum of lengths of weighted paths is minimal is called a Huffman tree. That is, if $L = \sum_{k=1}^{n} w_k p_k$ is minimal, the binary tree is a Huffman tree.

- Given n nodes whose weights are $w_1, w_2, \ldots, w_n$ respectively, the process constructing a Huffman tree is as follow.
- First n nodes constitute a set of n binary trees $F=\{T_1, T_2, \ldots\ldots, T_n\}$, where T_i has only one node whose weight is w_i, $1 \le i \le n$.
- while (F is not a tree)
- { Replace the rooted trees T and T' of least weights from F with a tree having a new root that has T as its left subtree and T' as its right subtree;
- The weight of the new tree = the weight of T + the weight of T';
- }

- A Huffman tree is a complete binary tree.
- In a Huffman tree, if there are n leaves, there are $2n$-1 nodes. Constructing a Huffman tree is a greedy method, each time two trees with least weights are selected.
- A min heap is used to store roots of trees when a Huffman tree is constructed.

10.3.1 Fence Repair

- **Source: USACO 2006 November Gold**
- **IDs for Online Judge: POJ 3253**

- Farmer John wants to repair a small length of the fence around the pasture. He measures the fence and finds that he needs N ($1 \leq N \leq 20,000$) planks of wood, each having some integer length L_i ($1 \leq L_i \leq 50,000$) units. He then purchases a single long board just long enough to saw into the N planks (i.e., whose length is the sum of the lengths L_i). FJ is ignoring the "kerf", the extra length lost to sawdust when a sawcut is made; you should ignore it, too.
- FJ sadly realizes that he doesn't own a saw with which to cut the wood, so he mosies over to Farmer Don's Farm with this long board and politely asks if he may borrow a saw.

- Farmer Don, a closet capitalist, doesn't lend FJ a saw but instead offers to charge Farmer John for each of the *N*-1 cuts in the plank. The charge to cut a piece of wood is exactly equal to its length. Cutting a plank of length 21 costs 21 cents.
- Farmer Don then lets Farmer John decide the order and locations to cut the plank. Help Farmer John determine the minimum amount of money he can spend to create the *N* planks. FJ knows that he can cut the board in various different orders which will result in different charges since the resulting intermediate planks are of different lengths.

- **Input**
- Line 1: One integer *N*, the number of planks;
- Lines 2..*N*+1: Each line contains a single integer describing the length of a needed plank.
- **Output**
- Line 1: One integer: the minimum amount of money he must spend to make *N*-1 cuts.

- **Hint**
- He wants to cut a board of length 21 into pieces of lengths 8, 5, and 8.
- The original board measures 8+5+8=21. The first cut will cost 21, and should be used to cut the board into pieces measuring 13 and 8. The second cut will cost 13, and should be used to cut the 13 into 8 and 5. This would cost 21+13=34. If the 21 was cut into 16 and 5 instead, the second cut would cost 16 for a total of 37 (which is more than 34).

Analysis

Because each cut produces two planks of wood, the process cutting planks can be represented as a binary tree. The initial single long board is as the root, and the length of the board is its weight; N planks are as N leaves, where the weight of the i-th leaf is the length of the i-th plank w_i, and the length from the root to leaf p_i is the number of cuts producing the i-th plank. Based on the problem description, the cost for cutting the i-th plank is $p_i * w_i$, $1 \le i \le n$. Obviously the total cost is $\sum_{k=1}^{n} w_k p_k$. Therefore calculating the minimal charge to cut a piece of wood is to calculate a Huffman tree.

- The process is as follow.
- A min heap is constructed based on lengths of N planks. Each times roots of the heap are deleted twice. The two deleted roots' weights are a and b respectively. A new node whose weight is $(a+b)$ is inserted into the min heap. And the cost *ans* increases $(a+b)$. Repeat the process until there is only one node in the min heap. At that time *ans* is the minimal cost.