Module – 2
Combinatorial Optimization
Day-2: 20-10-2020
Afternoon session

- Backtracking
  - N Queen problem
  - Hamiltonian Circuit
  - Subset Sum
- Greedy Techniques
  - Huffman trees

Design of Algorithmic Techniques

- Algorithm types include:
  - Simple recursive algorithms
  - Backtracking algorithms
  - Divide and conquer algorithms
  - Dynamic programming algorithms
  - Greedy algorithms
  - Branch and bound algorithms
  - Brute force algorithms
  - Randomized algorithms

Backtracking

The principal idea is to construct solutions one component at a time and evaluate such partially constructed candidates as follows.

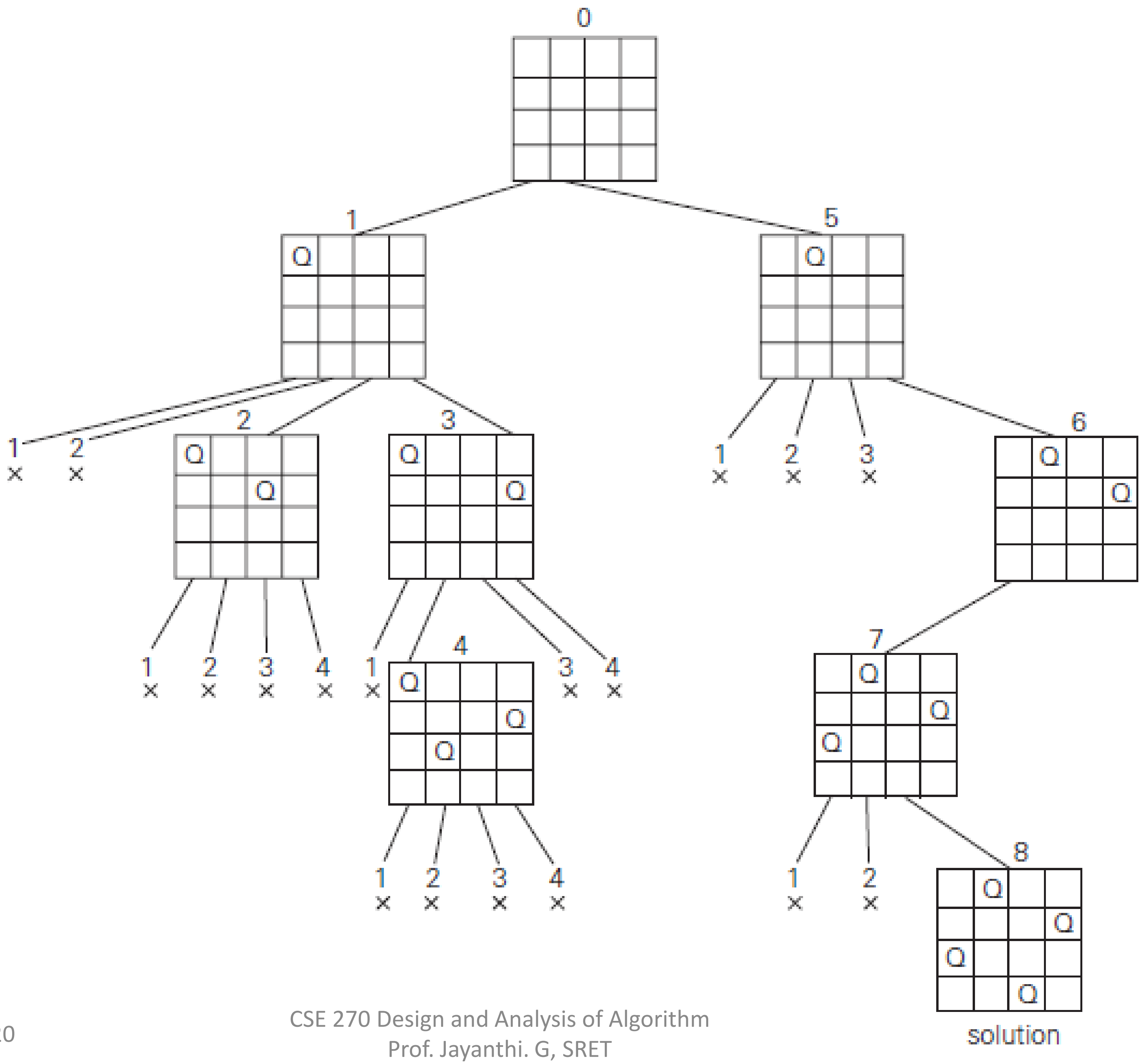
- If a partially constructed solution can be developed further without violating the problem's constraints, it is done by taking the ***first remaining legitimate option*** for the next component.
- If there is no legitimate option for the next component, no alternatives for any remaining component need to be considered.
- In this case, ***the algorithm backtracks to replace the last component*** of the partially constructed solution with its next option.
- It is convenient to implement this kind of processing ***by constructing a tree of choices being made, called the state-space tree.***
- Its root represents an initial state before the search for a solution begins. The nodes of the first level in the tree represent the choices made for the first component of a solution, the nodes of the second level represent the choices for the second component, and so on.

N Queen Problem

- A node in a state-space tree is said to be promising if it corresponds to a partially constructed solution that may still lead to a complete solution; otherwise,it is called nonpromising.
- Leaves represent either nonpromising dead ends or complete solutions found by the algorithm. ***In the majority of cases, a statespace tree for a backtracking algorithm is constructed in the manner of depthfirst search.***
- If the current node is promising, its child is generated by adding the first remaining legitimate option for the next component of a solution, and the processing moves to this child.
- If the current node turns out to be nonpromising, the algorithm backtracks to the node's parent to consider the next possible option for its last component; if there is no such option, it backtracks one more level up the tree, and so on. Finally, if the algorithm reaches a complete solution to the problem, it either stops (if just one solution is required) or continues searching for other possible solutions.

Steps

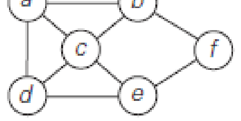
- 1) Start in the leftmost column
- 2) If all queens are placed return true
- 3) Try all rows in the current column. Do following for every tried row.
  - a) If the queen can be placed safely in this row then mark this [row, column] as part of the solution and recursively check if placing queen here leads to a solution.
  - b) If placing the queen in [row, column] leads to a solution then return true.
  - c) If placing queen doesn't lead to a solution then unmark this [row, column] (Backtrack) and go to step (a) to try other rows.
  - 3) If all rows have been tried and nothing worked, return false to trigger backtracking.


0
1
5
2
3
6
4
7
8
Q
×
solution

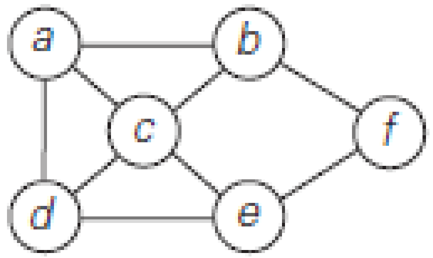
Complexity

- The worst case “brute force” solution for the N-queens puzzle has an O(n^n) time complexity.
- This means it will look through every position on an NxN board, N times, for N queens.

Hamiltonian Circuit

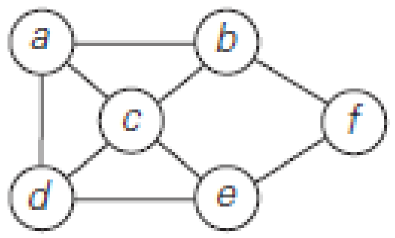


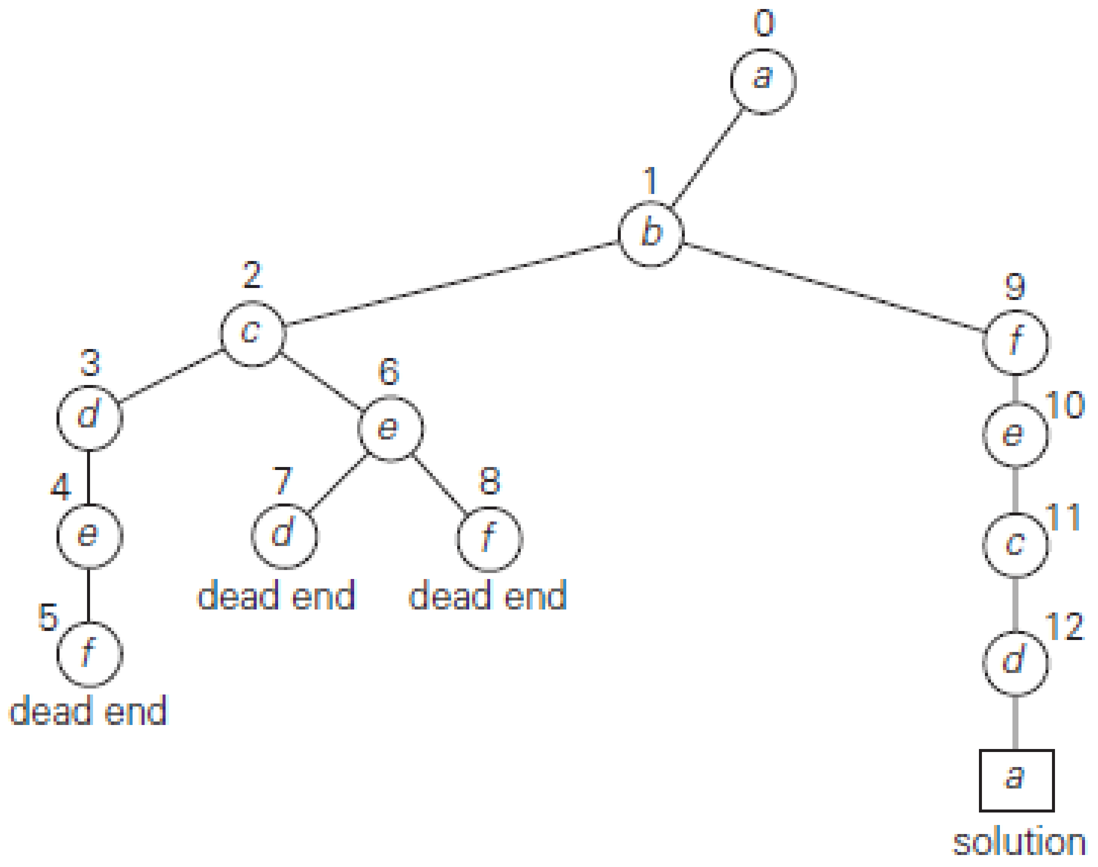
- Vertex a the root of the state-space tree. The first component of our future solution, if it exists, is a first intermediate vertex of a Hamiltonian circuit to be constructed.
- Using the alphabet order to break the three-way tie among the vertices adjacent to a, we select vertex b.
- From b, the algorithm proceeds to c, then to d, then to e, and finally to f, which proves to be a dead end.
- So the algorithm backtracks from f to e, then to d, and then to c, which provides the first alternative for the algorithm to pursue.
- Going from c to e eventually proves useless, and the algorithm has to backtrack from e to c and then to b.
- From there, it goes to the vertices f , e, c, and d, from which it can legitimately return to a, yielding the Hamiltonian circuit a, b, f , e, c, d, a.
- If we wanted to find another Hamiltonian circuit, we could continue this process by backtracking from the leaf of the solution found.



What are the nodes?
a,b,c,d,e,f are nodes or vertices of the graph.
Computer network as a graph. Each node is a host.
Edges of graphs is the physical connection

Circuit from the graph : closed path or loop or circuit
A-B-F-E-C-D-A – CLOSED PATH OR CIRCUIT
A-B-F-E-D-C-A – SAME
A-C-D-E-F-B-A


a
b
c
d
e
f


0 a
1 b
2 c
3 d
4 e
5 f
dead end
6 e
7 d
dead end
8 f
dead end
9 f
10 e
11 c
12 d
a
solution

Subset Sum

consider the subset-sum problem: find a subset of a given

set A = {a1, . . . , an} of n positive integers whose sum is equal to a given positive integer d. For example, for **A = {1, 2, 5, 6, 8}** and d = 9,

there are two solutions:**{1, 2, 6} and {1, 8}.**

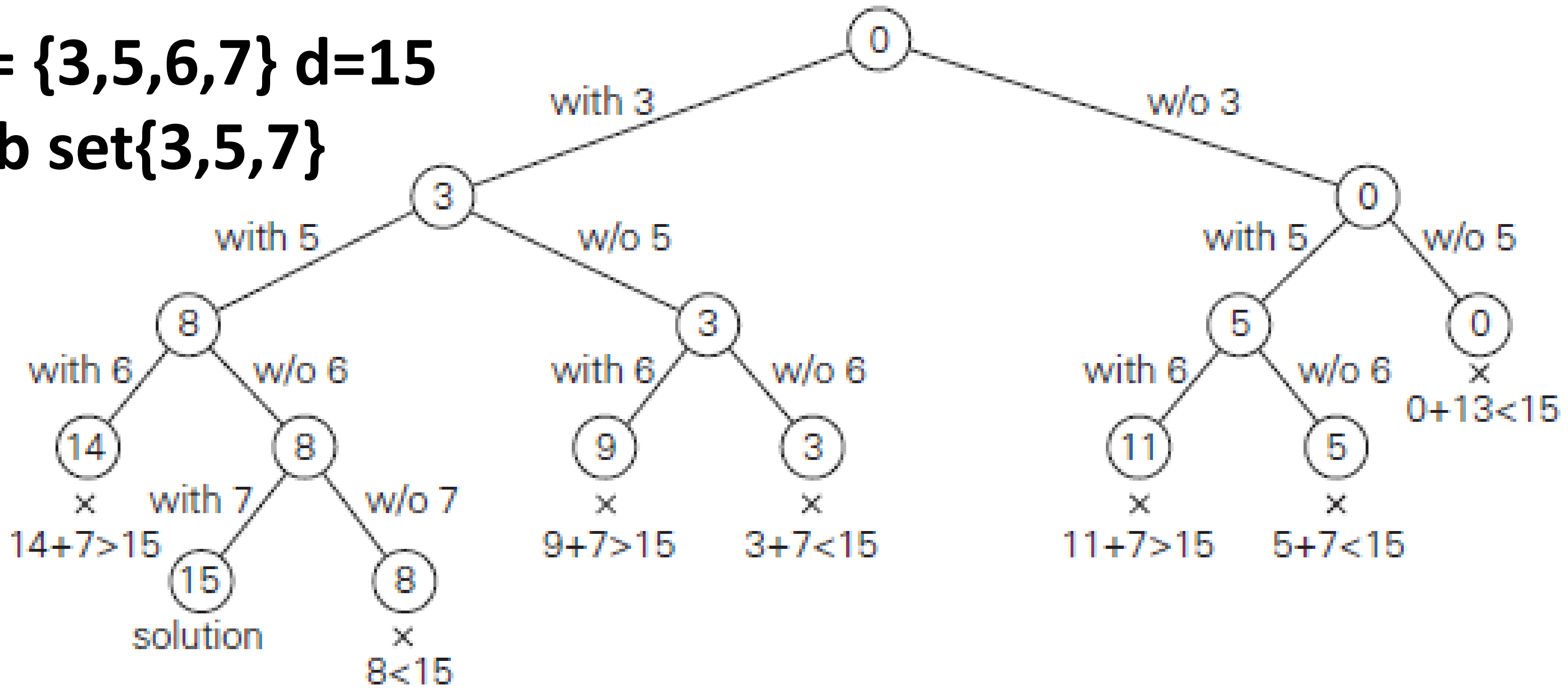
Of course, some instances of this problem may have no solutions.

It is convenient to sort the set's elements in increasing order. So, we will assume that

a1< a2 < . . . < an.

State space search tree

A = {3,5,6,7} d=15
Sub set{3,5,7}



Greedy Technique

Huffman coding

There are mainly two major parts in Huffman Coding

1) Build a Huffman Tree from input characters.

2) Traverse the Huffman Tree and assign codes to characters.

Steps to build Huffman Tree

Input is an array of unique characters along with their frequency of occurrences and output is Huffman Tree.

1. Create a leaf node for each unique character and build a min heap of all leaf nodes
(Min Heap is used as a priority queue.
The value of frequency field is used to compare two nodes in min heap.
Initially, the least frequent character is at root)

2. Extract two nodes with the minimum frequency from the min heap.

3. Create a new internal node with a frequency equal to the sum of the two nodes frequencies.
Make the first extracted node as its left child and the other extracted node as its right child. Add this node to the min heap.

4. Repeat steps#2 and #3 until the heap contains only one node. The remaining node is the root node and the tree is complete.

Example

character	**Frequency**
a	**5**
b	**9**
c	**12**
d	**13**
e	**16**
f	**45**

Step 1. Build a min heap that contains 6 nodes where each node represents root of a tree with single node.

Step 2 Extract two minimum frequency nodes from min heap. Add a new internal node with frequency 5 + 9 = 14.

character	**Frequency**
c	**12**
d	**13**
Internal Node	**14**
e	**16**
f	**45**

Step 3: Extract two minimum frequency nodes from heap. Add a new internal node with frequency 12 + 13 = 25

character	**Frequency**
Internal Node	**14**
e	**16**
Internal Node	**25**
f	**45**

Step 4: Extract two minimum frequency nodes. Add a new internal node with frequency 14 + 16 = 30

character	**Frequency**
Internal Node	**25**
Internal Node	**30**
f	**45**

Step 5: Extract two minimum frequency nodes. Add a new internal node with frequency 25 + 30 = 55
Now min heap contains 2 nodes.

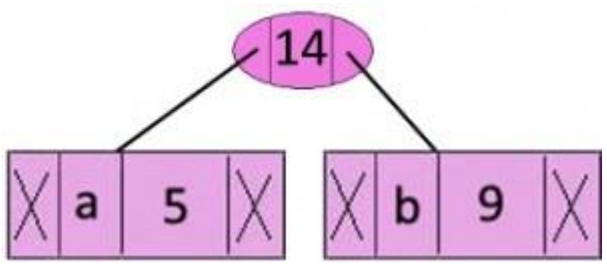
character	**Frequency**
f	**45**
Internal Node	**55**

Step 6: Extract two minimum frequency nodes. Add a new internal node with frequency 45 + 55 = 100
Now min heap contains only one node.

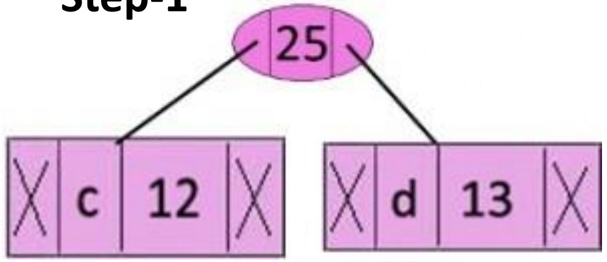
character	**Frequency**
Internal Node	**100**

Since the heap contains only one node, the algorithm stops here.
Steps to print codes from Huffman Tree:
Traverse the tree formed starting from the root. Maintain an auxiliary array.
While moving to the left child, write 0 to the array.
While moving to the right child, write 1 to the array. Print the array when a leaf node is encountered.
The codes are as follows:

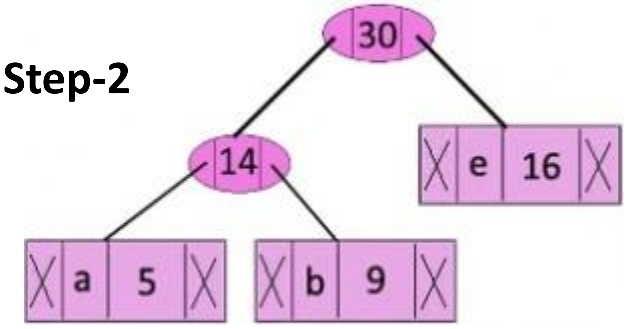
character	code-word
f	0
c	100
d	101
a	1100
b	1101
e	111



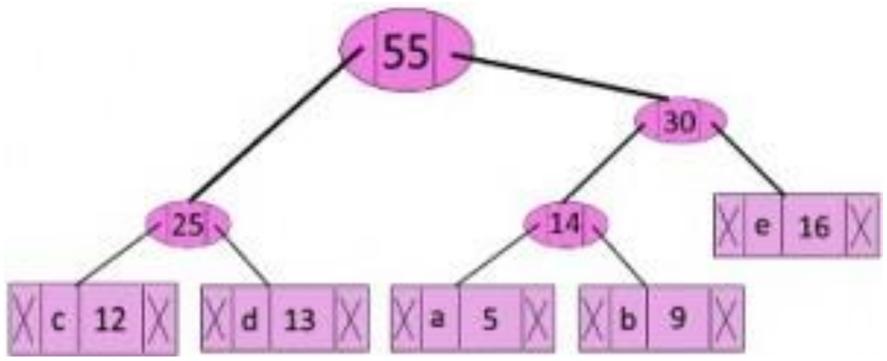
Step-1



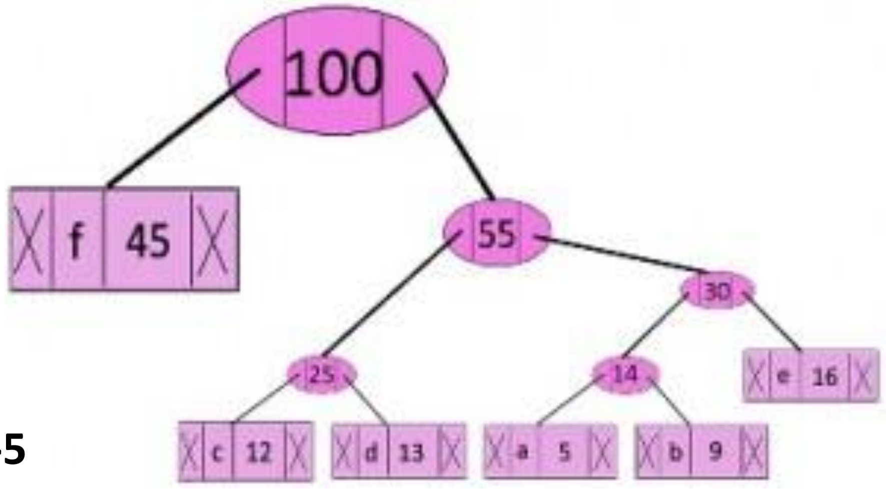
Step-2



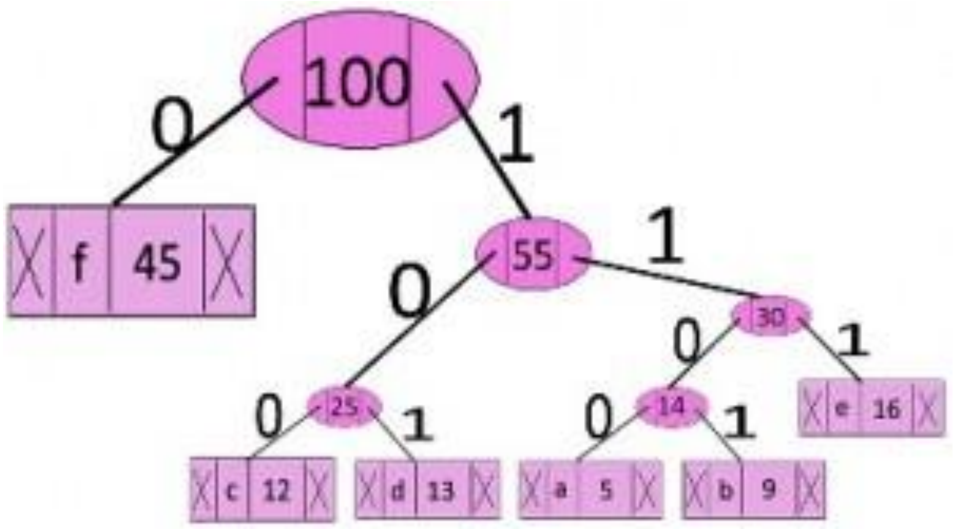
Step-3



Step-4



Step-5



character	code-word
f	0
c	100
d	101
a	1100
b	1101
e	111

Data Compression

- Files are stored as binary code in a computer and each character of the file is assigned a binary character code and normally, these character codes are of fixed length for different characters.
- For example, if we assign 'a' as 000 and 'b' as 001, the length of the codeword for both the characters are fixed i.e., both 'a' and 'b' are taking 3 bits.
- ***Huffman code doesn't use fixed length codeword for each character and assigns codewords according to the frequency of the character appearing in the file.***
- Huffman code assigns a shorter length codeword for a character which is used more number of time (or has a high frequency) and a longer length codeword for a character which is used less number of times (or has a less frequency).