Greedy Algorithms

TOPICS

- Greedy Strategy
- Activity Selection
- Minimum Spanning Tree
- Shortest Paths
- Huffman Codes
- Fractional Knapsack

Chapter 5

Algorithm Design ***Kleinberg and Tardos***

The Greedy Principle

- **The problem:** We are required to find a feasible solution that either maximizes or minimizes a given objective solution**.**
- It is easy to determine a feasible solution but not necessarily an optimal solution.
- The greedy method solves this problem in stages, at each stage, a decision is made considering inputs in an order determined by the selection procedure which may be based on an optimization measure.
- The greedy algorithm always makes the choice that looks best at the moment.
  - For each decision point in the greedy algorithm, the choice that seems best at the moment is chosen
- It makes a local optimal choice that may lead to a global optimal choice.

Activity Selection Problem

- Scheduling a resource among several competing activities.
- S = {1,2, 3, …, *n*} is the set of *n* proposed activities
- The activities share a resource, which can be used by only one activity at a time -a Tennis Court, a Lecture Hall etc.,
- Each activity *i* has a start time, s_i and a finish time f_i, where $s_i \le f_i$.
- When selected, the activity takes place during time (s_i, f_i)
- Activities *i* and *j* are compatible if $s_i \ge f_j$ or $s_j \ge f_i$
- The activity-selection problem selects the maximum-size set of mutually compatible activities
- The input activities are in order by increasing finishing times.
- $f_1 \le f_2 \le f_3 \ldots \le f_n$; Can be sorted in O (*n log n*) time

Procedure for activity selection (from CLRS)

Procedure GREEDY_ACTIVITY_SELECTOR(s, f)

$n \leftarrow$ length [S]; in order of increasing finishing times;

$A \leftarrow \{1\}$; first job to finish

$j \leftarrow 1$;

for $i \leftarrow 2$ to n

    do if $s_i \geq f_j$

        then $A \leftarrow A \cup \{i\}$;

            $j \leftarrow i$;

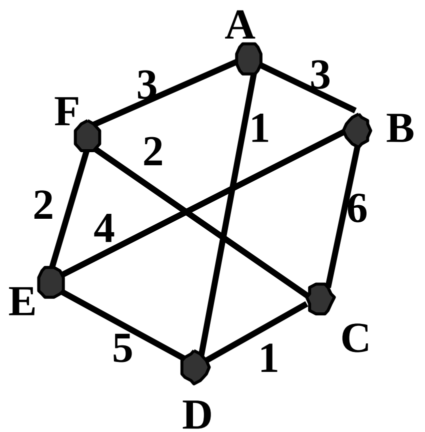
i	s_i	f_i
1	1	4
2	3	5
3	0	6
4	5	7
5	3	8
6	5	9
7	6	10
8	8	11
9	8	12
10	2	13
11	12	14

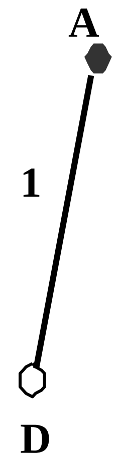
- Initially we choose activity 1 as it has the least finish time.
- Then, activities 2 and 3 are not compatible as $s_2 < f_1$ and $s_3 < f_1$.
- We choose activity 4, $s_4 > f_1$, and add activity 4 to the set A.
- A = {1, 4}
- Activities 5, 6, and 7 are incompatible and activity 8 is chosen
- A = {1,4,8}
- Finally activity 10 is incompatible and activity 11 is chosen
- A {1,4,8,11}
- The algorithm can schedule a set of n activities in Θ (n) time.

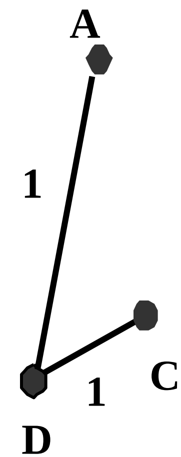
Greedy Algorithms

- Minimum Cost Spanning Tree
  - Kruskal's algorithm
  - Prim's Algorithm
- Single Source Shortest Path
- Huffman Codes

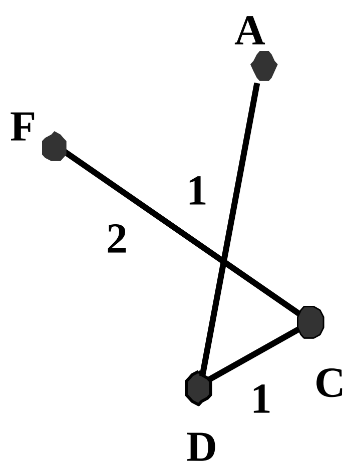
Prim’s Algorithm

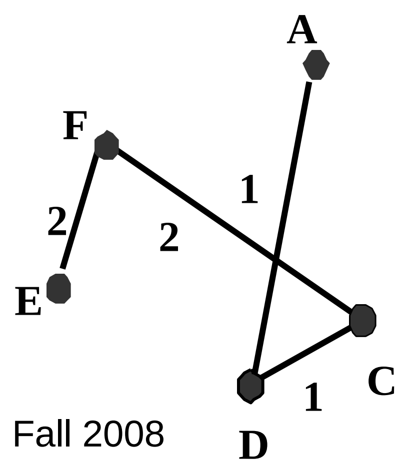


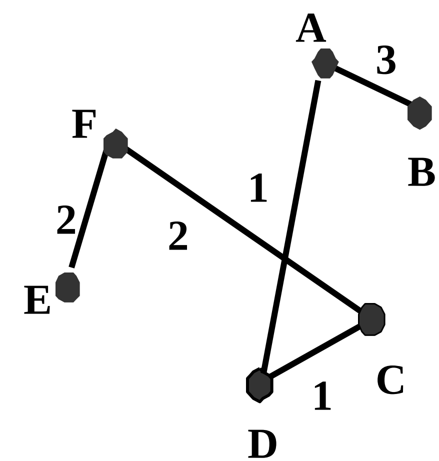




The equivalent Graph and the MCST







Huffman codes

Huffman codes are used to compress data. We will study Huffman's greedy algorithm for encoding compressed data.

Data Compression

- **A given file can be considered as a string of characters.**
- **The work involved in compressing and uncompressing should justify the savings in terms of storage area and/or communication costs.**
- **In ASCII all characters are represented by bit strings of size 7.**
- **For example if we had 100000 characters in a file then we need 700000 bits to store the file using ASCII.**

Example

The file consists of only 6 characters as shown in the table below.
Using the fixed-length binary code, the whole file can be encoded in 300,000 bits.
However using the variable-length code , the file can be encoded in 224,000 bits.

	a	b	c	d	e	f
Frequency (in thousands)	**45**	**13**	**12**	**16**	**9**	**5**
Fixed-length codeword	**000**	**001**	**010**	**011**	**100**	**101**
Variable-length codeword	**0**	**101**	**100**	**111**	**1101**	**1100**

A variable length coding scheme assigns frequent characters, short code words and infrequent characters, long code words.
In the above variable-length code, 1-bit string represents the most frequent character a, and a 4-bit string represents the most infrequent character *f*.

Let us denote the characters by C_1, C_2, …, C_n and denote their frequencies by f_1, f_2, ,,,, f_n. Suppose there is an encoding E in which a bit string S_i of length s_i represents C_i, the length of the file compressed by using encoding *E* is

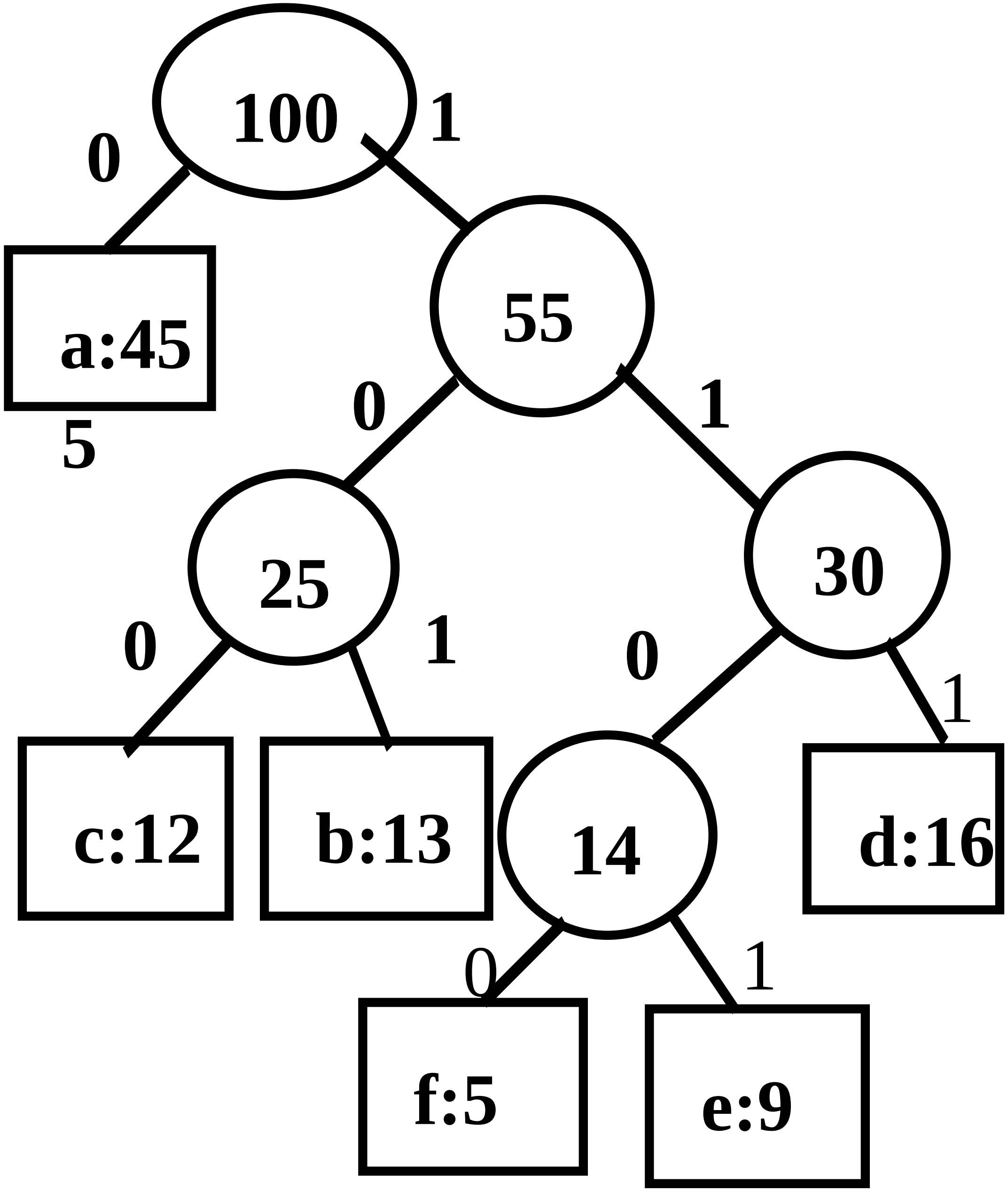
$$L(E,F) = \sum_{i=1}^{n} s_i \cdot f_i$$

Prefix Codes

- The prefixes of an encoding of one character must not be equal to a complete encoding of another character.
  - 1100 and 11001 are not valid codes
  - because 1100 is a prefix of 11001
- This constraint is called the prefix constraint.
- Codes in which no codeword is also a prefix of some other code word are called prefix codes.
- Shortening the encoding of one character may lengthen the encodings of others.
- To find an encoding *E* that satisfies the prefix constraint and minimizes L(*E*,*F*).

0 101 100 111 1101 1100

The prefix code for file can be represented by a binary tree in which every non leaf node has two children. Consider the variable-length code of the table above, a tree corresponding to the variable-length code of the table is shown below.



Note that the length of the code for a character is equal to the depth of the character in the tree shown.

Greedy Algorithm for Constructing a Huffman Code

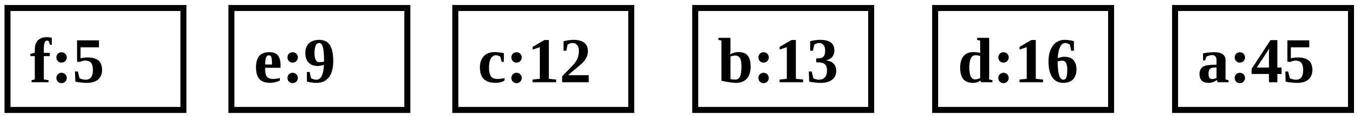
The algorithm builds the tree corresponding to the optimal code in a bottom-up manner.
The algorithm begins with a set of $|C|$ leaves and performs a sequence of 'merging' operations to create the tree.
***C* is the set of characters in the alphabet.**

Procedure Huffman_Encoding(*S*,*f*);

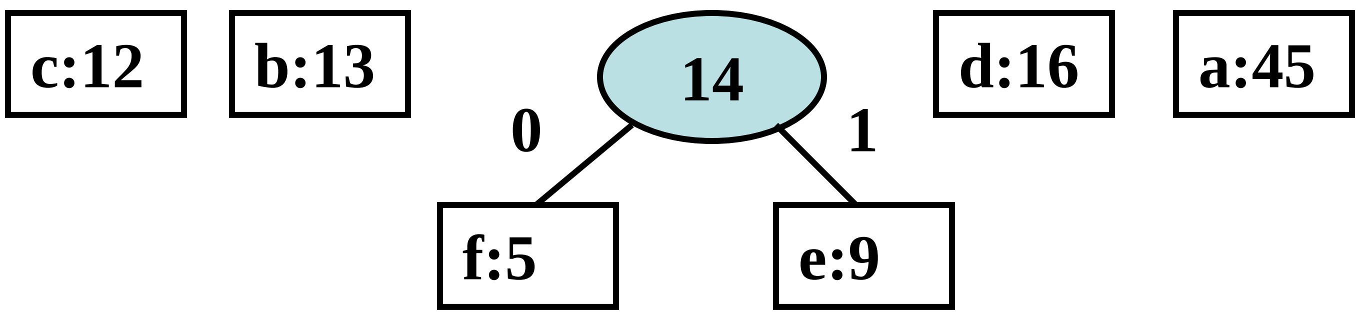
Input : *S* (a string of characters) and *f* (an array of frequencies).

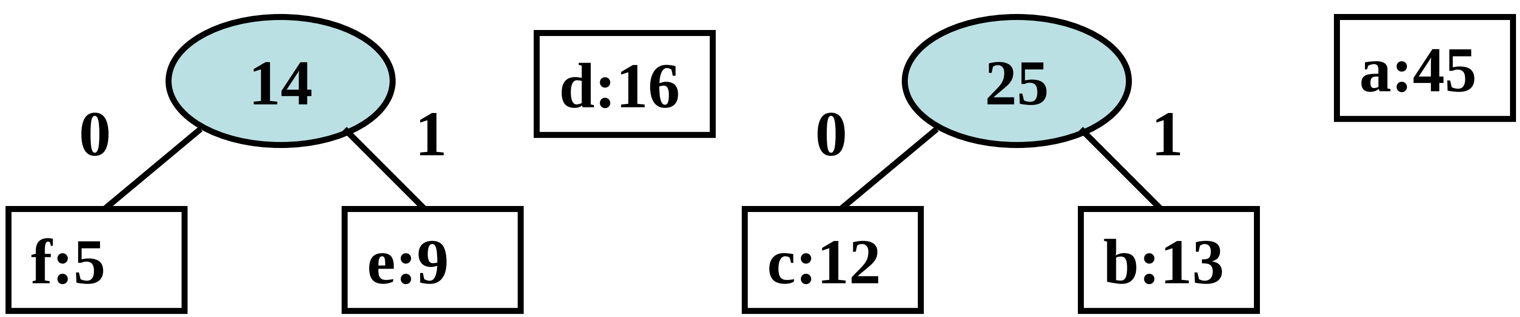
Output : *T* (the Huffman tree for *S*)

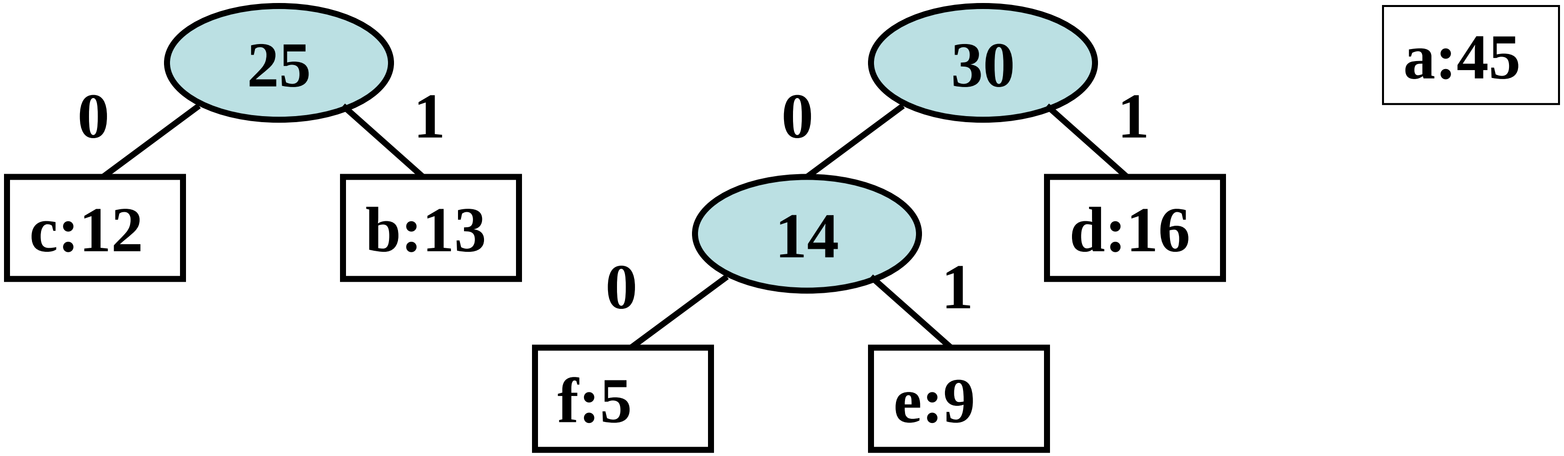
1. insert all characters into a heap H according to their frequencies;
2. **while** H is not empty **do**
3. **if** H contains only one character x **then**
4. $x \leftarrow$ root (T);
5. **else**
6. $z \leftarrow$ ALLOCATE_NODE();
7. $x \leftarrow$ left[T,z] $\leftarrow$ EXTRACT_MIN(H);
8. $y \leftarrow$ right[T,z] $\leftarrow$ EXTRACT_MIN(H);
9. $f_z \leftarrow f_x + f_y$;
10. INSERT(H,z);



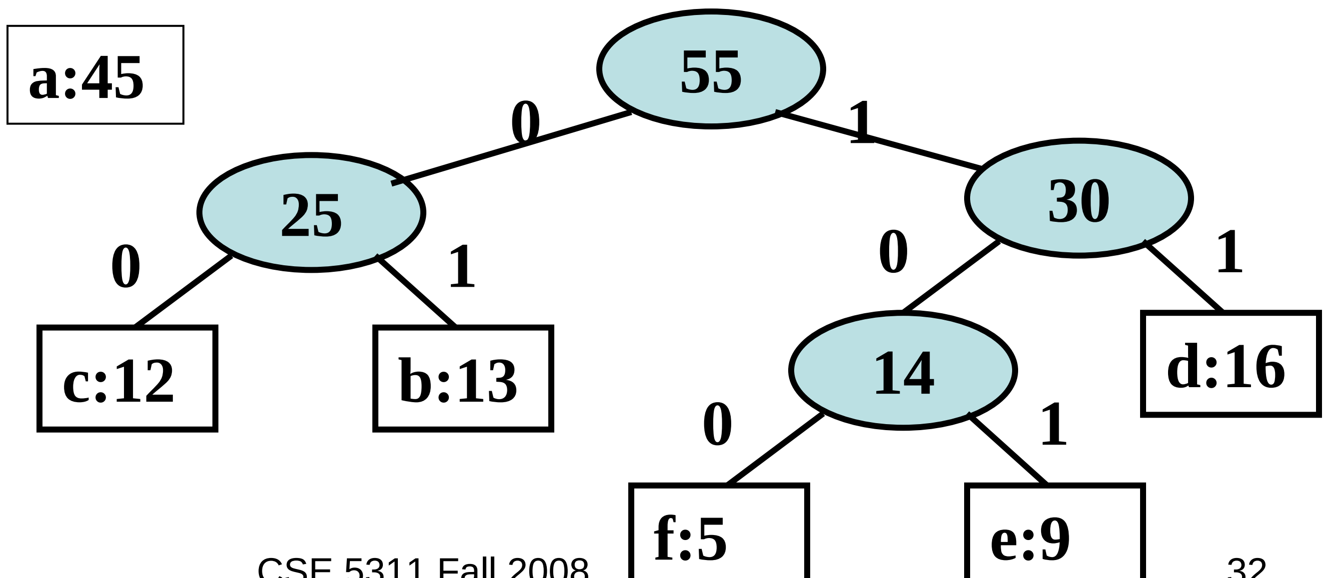
- The algorithm is based on a reduction of a problem with *n* characters to a problem with *n*-1 characters.
- A new character replaces two existing ones.

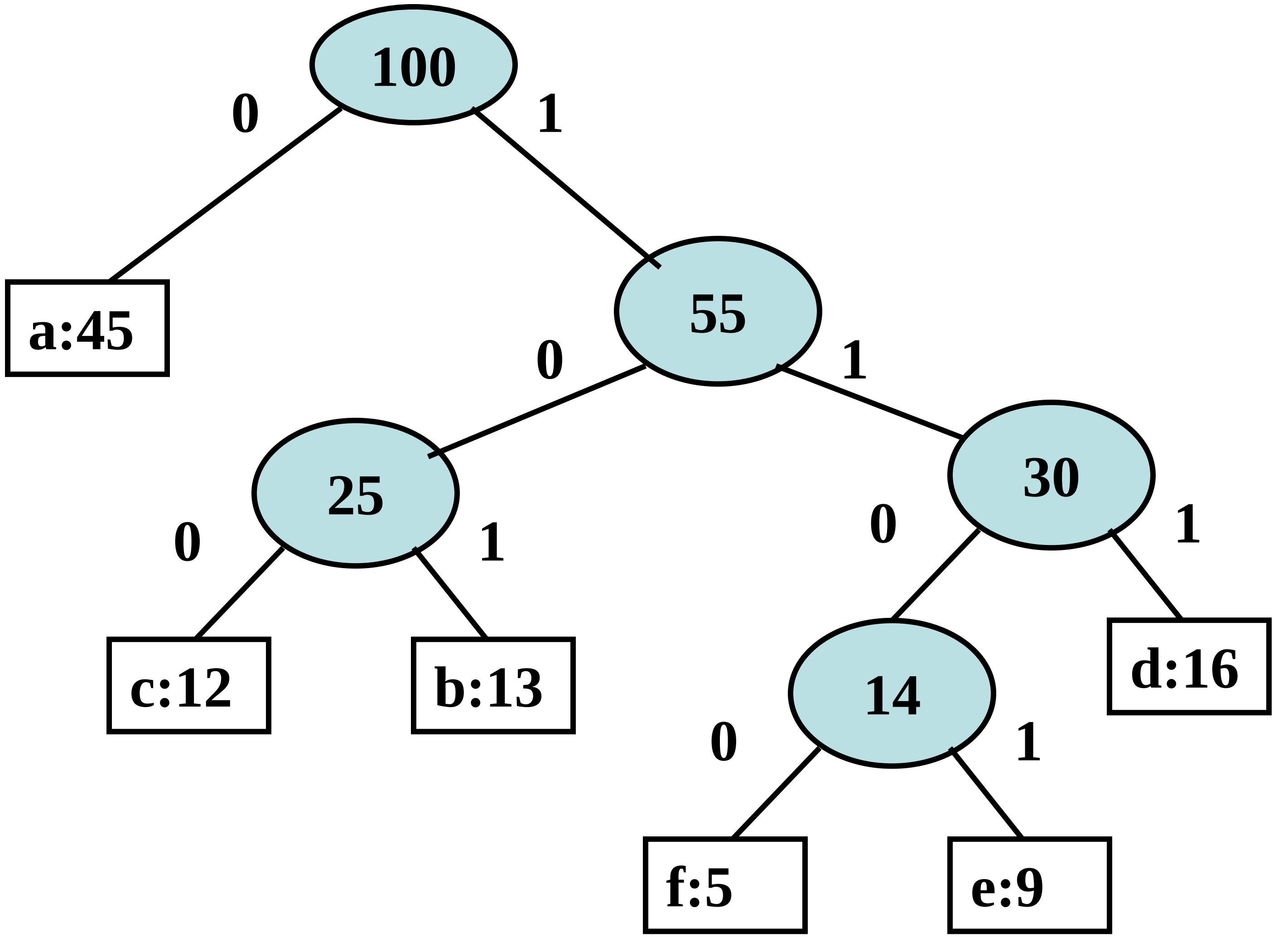






Suppose C_i and C_j are two characters with minimal frequency, there exists a tree that minimizes $L(E,F)$ in which these characters correspond to leaves with the maximal distance from the root.





0 101 100 111 1101 1100

Complexity of the algorithm

Building a heap in step 1 takes O(*n*) time
Insertions (steps 7 and 8) and
deletions (step 10) on H
take O (*log n*) time each
Therefore Steps 2 through 10 take O(*n logn*) time

Thus the overall complexity of the algorithm is
O(*n logn*).

- The fractional knapsack problem
    - Limited supply of each item
    - Each item has a size and a value per unit (e.g., Pound)
  - greedy strategy
    - Compute value per Pound for each item
    - Arrange these in non-increasing order
    - Fill sack with the item of greatest value per pound until either the item is exhausted or the sack is full
    - If sack is not full, fill the remainder with the next item in the list
    - Repeat until sack is full

How about a 0-1 Knapsack?? Can we use Greedy strategy?

Problems

1. Suppose that we have a set of k activities to schedule among n number of lecture halls; activity i starts at time si and terminates at time fi $1 \leq i \leq k$. We wish to schedule all activities using as few lecture halls as possible. Give an efficient greedy algorithm to determine which activity should use which lecture hall.
2. You are required to purchase n different types of items. Currently each item costs \$$D$. However, the items will become more expensive according to exponential growth curves. In particular the cost of item j increases by a factor $r_j > 1$ each month, where r_j is a given parameter. This means that if item j is purchased t months from now, it will cost $D \times r_j^t$. Assume that the growth rates are distinct, that is $r_i = r_j$ *for items* $i \neq j$. Given that you can buy only one item each month, design an algorithm that takes n rates of growth $r_1, r_2, \ldots, r_n$, *and computes an order in which to buy the items so that the total amount spent is minimized.*