Greedy Algorithms

Optimization Problems

- Optimization problems
  - Problems that involve searching through set of configurations
  - Maximize or minimize objective function given some set of constraints
- Greedy Solution
  - Choose best possible or well understood configuration
  - Proceed with best available configuration at each step
  - Usually local optimum is chosen

Greedy Choice Property

- Global optimal solution can be reached by a series of locally optimal choices
  - Choices available currently
  - If greedy choice property satisfied, greedy strategy is optimal and correct
- Example of Greedy Algorithms
  - Path Finding
  - Coin changing problem
  - Scheduling problem
  - Knapsack problem

Fractional Knapsack

- Determine the amount of each item to take so that it fits the sack, and total utility is maximized

The optimization problem

- Given set S of n items, where each item *i* has
  - Benefit b_i >= 0 and Weight w_i
- We can take a fraction x_i of each item *i*
- We have knapsack that can carry weight atmost W
- Goal – Maximize total benefit s.t

$$\sum_{i \in S} x_i \leq W$$

- Objective function to maximize

$$\sum_{i \in S} b_i \left(x_i / w_i \right)$$

The Greedy Solution

- For each item i in S, give a value $v_i = b_i/w_i$
- Let current weight of knapsack be w
  - Remove from S an item with highest value v_i
  - Let a be min{w_i, W-w}
      - Check if the chosen object can fit the knapsack
  - set x_i to a, and increment w by a
  - Stop when w >= W or no more items in sack to add

Exercise

- Suppose you have a knapsack with capacity 10 and there are 4 items:
  - item 1 has weight 6 and benefit 9
  - item 2 has weight 3 and benefit 6
  - item 3 has weight 2 and benefit 8
  - item 4 has weight 2 and benefit 2
- What is the best set of items for the fractional knapsack problem?

Proof of Correctness

- Assume there are two objects *i* and *j*
  - Value of *i* less than value of *j*
  - $x_i < w_i$ and $x_j > 0$
  - Let $y = \min\{w_i - x_i, x_j\}$
- Can replace amount y of item j with equal amount of i
  - Increase total benefit without increasing weight
  - Can correctly compute optimal amounts for items by greedily choosing items with largest value index

Exercise

- Let S = {*a,b,c,d,e,f,g*} be a collection of objects with benefit-weight values as follows
  - a:(12,4), b:(10,6), c:(8,5), d:(11,7), e:(14,3), f:(7,1), g:(9,6)
- What is an optimal solution to the fractional knapsack problem for S assuming we have a sack that can hold objects with total weight 18

Task Scheduling

- Given a set of n tasks T
  - Each task *i* has start time s_i and end time e_i
  - $s_i < e_i$ and task is guaranteed to finish by e_i
  - Each machine can execute only one task at a time
- Goal :
  - Schedule all tasks in T on the fewest machines possible in a way that is non conflicting
- Similar to scheduling meetings, classes in limited number of rooms

Greedy Strategy

- Take the task *i* with smallest start time
  - If it does not conflict with task on machine *j* (initially 0) schedule on machine *j*
    - Else add a new machine and schedule task *i*
- Repeat this greedy process till all tasks are scheduled
- Running time O(nlogn)
- Proof – by contradiction
  - Show the tasks cannot be scheduled in k-1 machines in a non conflicting way

Exercise

- Solve the task scheduling problem for the following set of tasks
  - The tasks are specified as pairs of start times and end times
  - T = {(1,2), (1,3), (1,4), (2,5),(3,7),(4,9),(5,6),(6,8),(7,9)}

Activity Selection Problem

- Schedule several competing activities that require exclusive use of a common resource, with a goal of selecting a maximum-size set of mutually compatible activities
  - have a set S = {a_1, a_2,...,a_n} of n proposed activities
    - activity a_i has a start time s_i and a finish time f_i, where $0 <= s_i < f_i < \infty$
    - Activities a_i and a_j are compatible if the intervals $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap.
- Select a maximum-size subset of mutually compatible activities

Sample problem

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

Greedy Solution

- Order the tasks by finish time
  - Choose the task with earliest finishing time and schedule it.
  - The next task is the one which does not start earlier that the current task
- Theorem
  - Consider any nonempty subproblem S_k, and let a_m be an activity in S_k with the earliest finish time. Then a_m is included in some maximum-size subset of mutually compatible activities of S_k.

Huffman Coding

- It is an encoding algorithm for encoding text
  - Characters are stored with their probabilities or frequency of occurrence
- Codes can be of variable or fixed size (variable more efficient)
  - Number of bits of the coded characters is based on frequency
  - Shortest code is assigned to most frequently occurring character.
- Can use a greedy algorithm to find an efficient code for a text file

Huffman Coding problem

- Goal : Minimize total number of bits needed to encode a file
- For Huffman coding, a binary tree is constructed
  - Leaves are characters to be encoded
  - Nodes contain occurrence probabilities of the characters belonging to the subtree.
  - 0 and 1 are assigned to the branches of the tree arbitrarily
    - different Huffman codes possible for the same data.
- The number of bits $B(T)$ to encode a file T is

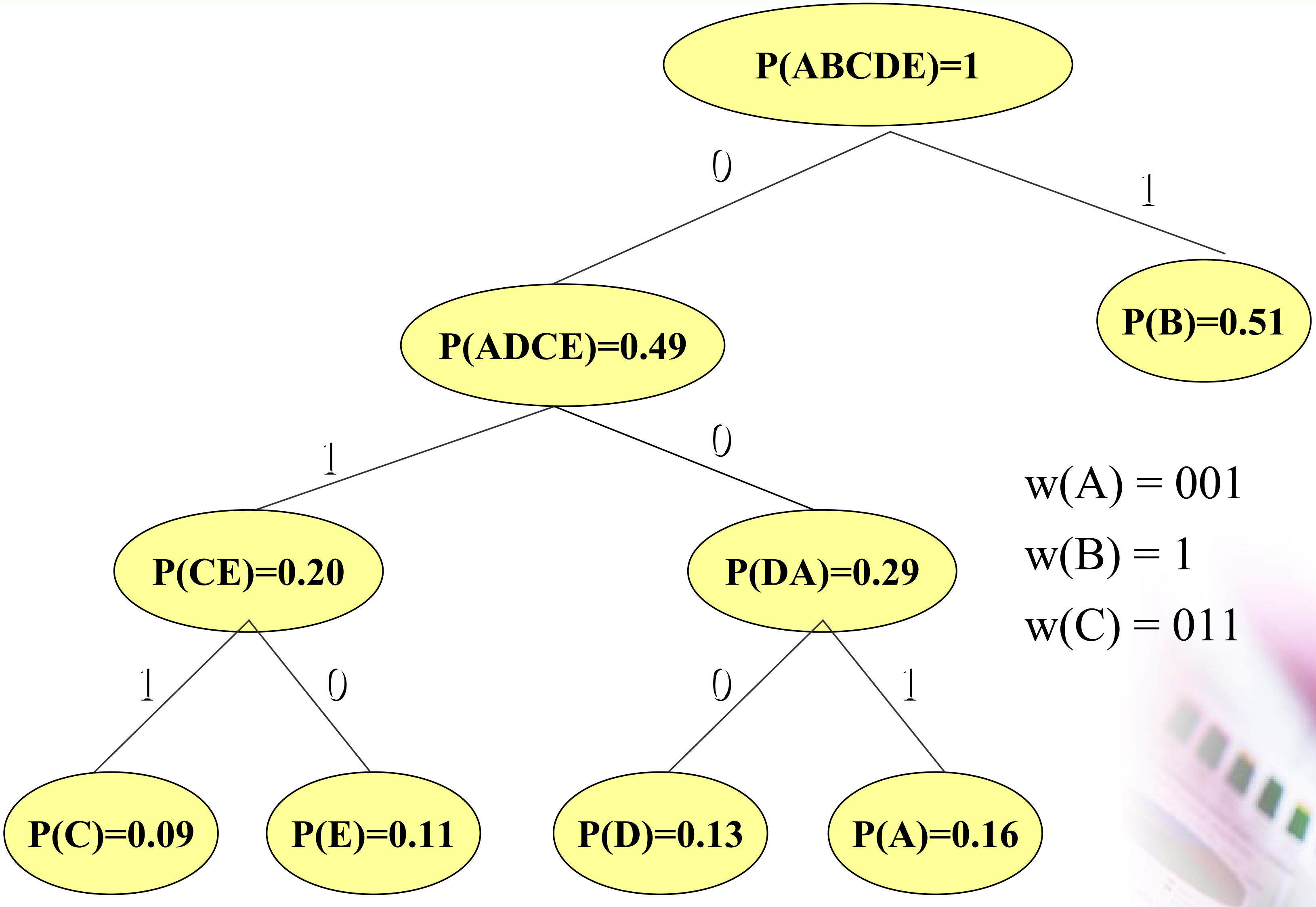
$$B(T) = \sum_{c \in C} freq(c) \, . \, d_T(c)$$

$d_T(c)$ is the length of the codeword for character c

Huffman Coding

- Each character is assigned a probability
  - Characters form leaves
- Characters with lowest probabilities combined in first binary tree
  - Combined probability is probability at parent node
- Each edge to the parent is assigned a bit, either 1 or 0 arbitrary
  - This is repeated (lowest two nodes combined) till all are combined at root
- e.g P(A) = 0.16, P(B) = 0.51, P(C) = 0.09, P(D)=0.13, P(E) = 0.11

Sample



w(A) = 001

w(B) = 1

w(C) = 011

Exercise

Consider the following alphabet A = {a, c, e, h, r, s, t, y}. Let the occurence probability be – P(a) = 0.2, P(c) = 0.12, P(e) = 0.3, P(h) = 0.04, P(r) = 0.1, P(s) = 0.08, P(t)= 0.11, P(y) = 0.05.

- Determine the Huffman codes for A, and encode the word "heart"