EECS 215
Lecture 6

Brian Demsky

bdemsky@uci.edu

The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine
Lecture 6: Overview

- Greedy Algorithms (Chapter 16)
Activity Selection Problem

- Set $S$ of $n$ activities
- Each activity has
  - $s_i = \text{start time of activity } i$
  - $f_i = \text{finish time of activity } i$
  - $0 \leq s_i < f_i < \infty$
- Activities $i$ and $j$ are compatible
  - if $f_i \leq s_j$ or $f_j \leq s_i$ (no overlap)
- Problem: select the maximum size subset of mutually compatible activities
Structure of the Optimal Solution

- $S_{ij} = \{a_k \in S: f_i \leq s_k < f_k \leq s_j\}$
- Let $A$ = optimal set for $S$
- $A_{i,j}$ = optimal for $S_{i,j}$
- if $a_k \in A_{i,j}$ then
  - $A_{i,k}$ must be optimal solution to $S_{i,k}$
  - Proof: assume there exist $A'_{i,k}$ with more activities than our $a_{i,k}$ (that is $|A'_{i,k}| > |A_{i,k}|$)
  => we can construct a longer $A'_{i,j}$ by using $A'_{i,k}$ prefix. Contradiction.
Solution Algorithm

- Let $c[i,j]=\text{max} \ # \ of \ compatible \ activities \ in \ S_{i,j}$
- Want to maximize $C[0,n+1]$
- $c[i,j]=0 \ if \ S_{i,j}=\{}$
- $=\text{max}\{c[i,k]+c[k,j]+1\} \ for \ i<k<j \ is \ S_{i,j}\neq\{}$
- Can do better with greedy choice
- Solution:
  Pick next compatible task with earliest finishing time
Knapsack Problems

- 0-1 Knapsack Problem
  - many items
  - ith item is worth $v_i$ dollars and weighs $w_i$
  - If the knapsack can hold a maximum weight $W$, how do we choose items

- Fractional Knapsack Problem
  - Can load fractions of an item
  - Strategy - fill with most valuable items (per unit weight), then next...
Optimal Structure

- Both problems have optimal structure
- If we remove $w_j$ the remain load must be the most value load weighing $W-w_j$ that can be taken from the items excluding $j$
- Greedy strategy works for fractional problem but not 0-1 problem
Huffman Codes

- Want to change representation of characters to compress file
- Normal representation uses the same number of bits to represent each character
- IDEA: Use variable bit representations.
- Use shortest bit string for the most commonly used characters
Huffman Code

- Goal: minimize $\sum(\text{frequency}(c) \times \text{bitlength}(c))$ for all $c$ in character set

Algorithm:
1. Place all characters in priority queue
2. Extract two items with lowest frequency
3. Combine the two items under one parent node
4. Reinsert this combined branch into set with frequency = sum of two items
5. Repeat step 2 until we’ve constructed a complete tree
Why Optimal

- Imagine another optimal tree
- Can swap least frequently two items to bottom most node (doesn’t make tree worse)
- Suppose we have a tree T’ for an alphabet C’, and a second alphabet C that differs only in that it contains characters x and y instead of character z and f[x]+f[y]=f[z]. Then we can create a tree T for C that replaces z with a sub-tree containing just x and y.