Algorithms and Data Structures (CSci 115)

California State University Fresno
College of Science and Mathematics
Department of Computer Science
H. Cecotti

Learning outcomes

- Greedy programming
  - Definitions and principles
  - Examples

- Rationale
  - Many graph walk algorithms use greedy approaches

Introduction

- Optimization problems
  - For most optimization problems, you desire to find ideally the **best** solution
    - Policy:
      - Sequence of steps leading to the solution
        - You may not be able to isolate each step from the others (dynamic programming)
        - You may be able to pick what s the best right now (#yolo)
- A **greedy algorithm** sometimes works well for optimization problems
  - Multiple steps:
    1. Take the best you can get **right now**,
       - Without regard for future consequences.
    2. Hope that by choosing a local optimum at each step, you will end up at a global optimum.

Example: counting coins

- Problem
  - To count out a certain amount of money, using the **fewest possible bills and coins**
- Greedy algorithm:
  1. At each step, take the **largest** possible bill or coin that does not overshoot
  - Example: To make $16.39, you can choose:
    - A $10 bill
    - A $5 bill
    - A $1 bill
    - A 25¢ coin
    - A 10¢ coin
    - Four 1¢ coins
  - The greedy algorithm always gives the optimum solution with US bills and coins

Example: counting coins

- Same problem
  - Monetary system: “klop” come in 1 klop, 7 klops, and 10 klop coins
- With the greedy algorithm to count out 15 klops
  - A 10 klop piece
  - Five 1 klop pieces, for a total of 15 kops
  - → 6 coins
- Best solution
  - Two 7 klop pieces and one 1 klop piece
  - → 3 coins
- Conclusion
  - The greedy algorithm provides a solution, but not in an optimal solution

Example: scheduling

- Problem
  - Example with CPU scheduling
  - You have to run 9 jobs
    - running times of 3, 5, 6, 10, 11, 14, 15, 18, and 20 minutes.
  - You have 3 processors that can be used to run these jobs.
- Strategy
  - Do the **longest**-running jobs first, on whatever processor is available.
  - Do the **shortest**-running jobs first, on whatever processor is available

Strategy

- A greedy algorithm obtains an optimal solution to a problem
  - By making **a sequence of choices**.
  - At each decision, the algorithm makes choice that seems best at the moment.
- Warning:
  - This heuristic strategy does not always produce an optimal solution
    - **Heuristic**:
      - Technique designed for solving a problem more quickly, or for finding an approximate solution when regular methods fail to find any exact solution, or are too slow.
  - **but** sometimes it does.
- Steps:
  1. Determine the optimal substructure of the problem.
  2. Develop a **recursive** solution.
  3. Show that if we make the greedy choice, then only 1 sub-problem remains.
  4. Prove that it is always safe to make the greedy choice.
     - Steps 3 and 4 can occur in either order.
  5. Develop a recursive algorithm that implements the greedy strategy.
  6. Convert the recursive algorithm to an iterative algorithm.

Design

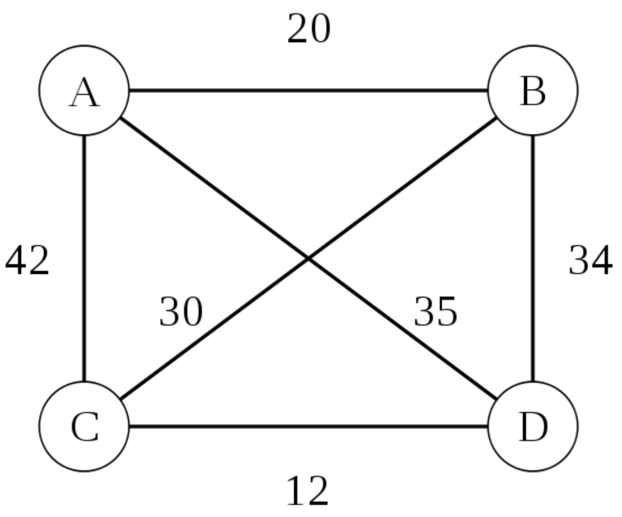
- Steps
  1. Cast the optimization problem as one in which we make a choice and are left with 1 subproblem to solve.
  2. Prove that there is **always** an optimal solution to the original problem
     - Which makes the greedy choice → the greedy choice is always **safe**.
  3. Demonstrate optimal substructure by showing:
     1. We made the greedy choice, rest is:
        - A subproblem with the property that if we combine an optimal solution to the subproblem with the greedy choice we have made,
        - We arrive at an optimal solution to the original problem.

Greedy choice property

- Goal
  - ➢ Assemble a globally optimal solution by making locally optimal (greedy) choices
    - When considering which choice to make
    - → make the choice looking best in the current problem
      - **Without** considering results from subproblems.
- The choice made by a greedy algorithm may depend on choices so far
  - ➢ **But** it cannot depend on any future choices or on the solutions to subproblems
  - ➢ Progress in a **top-down** fashion
    - Reduce each given problem instance to a smaller one.
- A greedy algorithm makes its first choice before solving any subproblems
- Comparison with dynamic programming (DP)
  - ➢ A choice at each step **but** the choice usually depends on the solutions to subproblems.
  - ➢ → DP solution in a **bottom-up** manner
    - Progressing from smaller subproblems to larger subproblems.
  - ➢ DP: solves the subproblems **before** making the first choice

Greedy

- Advantage
  - ➢ Used to get an approximation for Hard optimization problems
    - o Example: Traveling Salesman Problem (NP Hard problem)
      - Given a list of cities and the distances between each pair of cities
      - What is the shortest possible route that visits each city and returns to the origin city?



Huffman coding compression algorithm

- Huffman Coding (Huffman Encoding)
  - An algorithm for doing data compression
    - → the basic idea behind file compression.
- Fixed length
  - Every character is stored with a sequence of 0 and 1 using 8 bits
- Variable length encoding
  - It is to design an algorithm that can represent the same piece of text using lesser number of bits.
  - We assign variable number of bits to characters depending on their frequency in the given text. → some character might end up taking 1 bit, some might end up taking 2 bits, …
  - The problem with variable length encoding lies in its decoding.

Huffman coding

- Variable Length codes
  - Suppose the frequency distribution of the characters is:

A	B	C
999,000	500	500

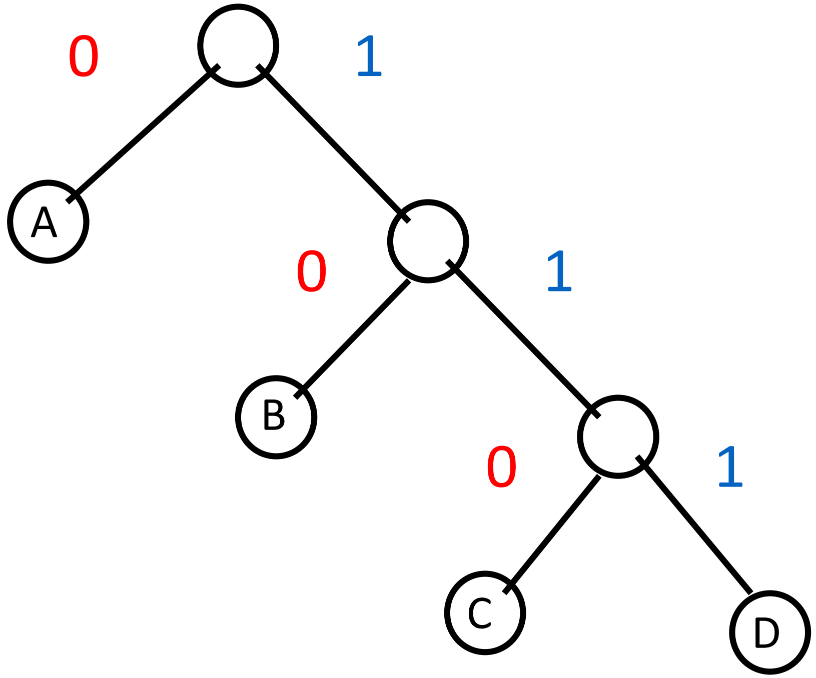
  - Encode:

A	B	C
0	10	11

  - The code of A: length 1, and the codes for B and C: length 2
    - Fixed code: 1,000,000 x 2 = 2,000,000
    - Variable code: 999,000 x 1 + 500 x 2 + 500 x 2 = 1,001,000

Huffman coding

- Decoding
  - In the variable length code: **Prefix code**
    - Where no code is a prefix of another.
  - Example: A = 0, B = 10, C = 11
    - None of the above codes is a prefix of another
  - Example
    - Input: AAABBCCCBCBAACC
    - Encoding: 0001010111111101110 0 01111
  - Binary tree:
    - 0: left child
    - 1: right child



Huffman coding

- **Pseudo code**
  - Consider all pairs: <frequency, symbol>.
  - Choose the 2 lowest frequencies:
    1. Create a node, set these pairs as children
    2. The node gets the combined frequency.
  - Iterate

Huffman coding

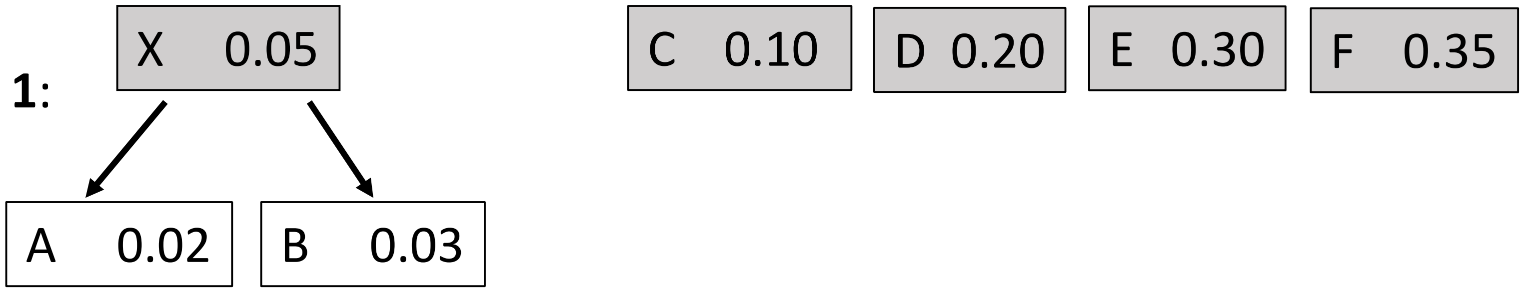
- Example:
  - Alphabet: A,B,C,D,E,F
  - Frequency table:

A	B	C	D	E	F
0.02	0.03	0.10	0.20	0.30	0.35

  - Current values (key,frequency)
    - **Step 0**:

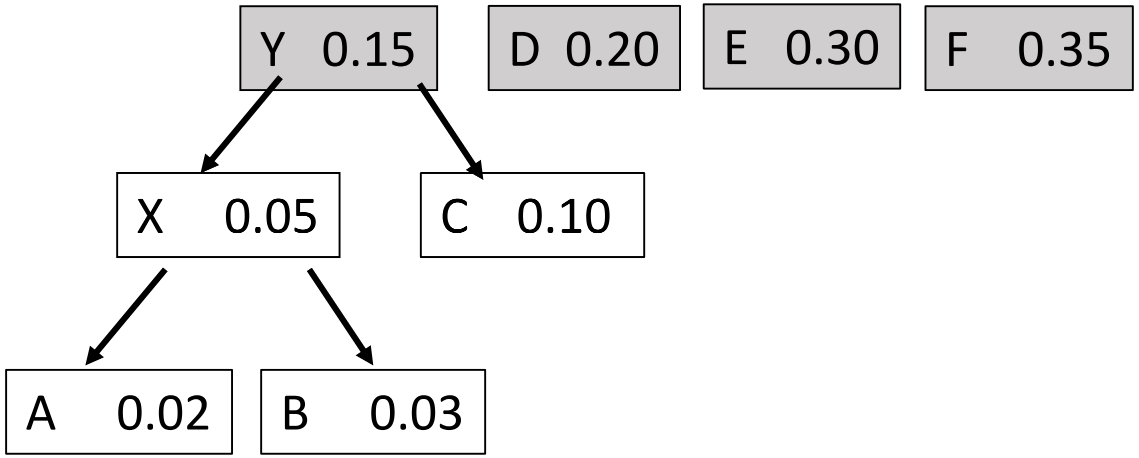
A 0.02	B 0.03	C 0.10	D 0.20	E 0.30	F 0.35

    - **Step 1**:

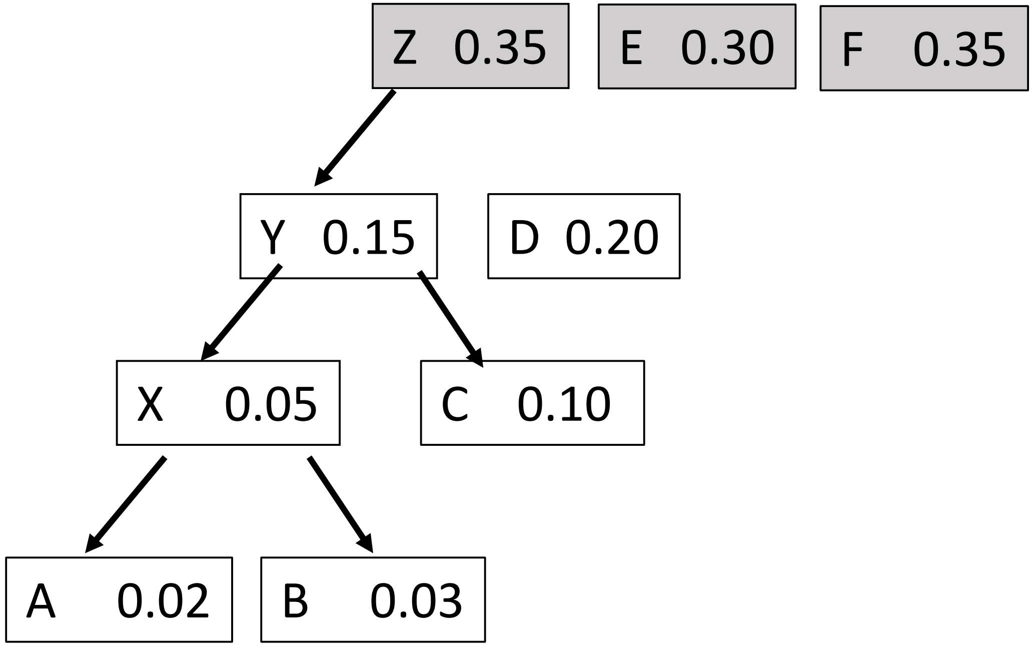


Huffman coding

- Example
  - Step 2

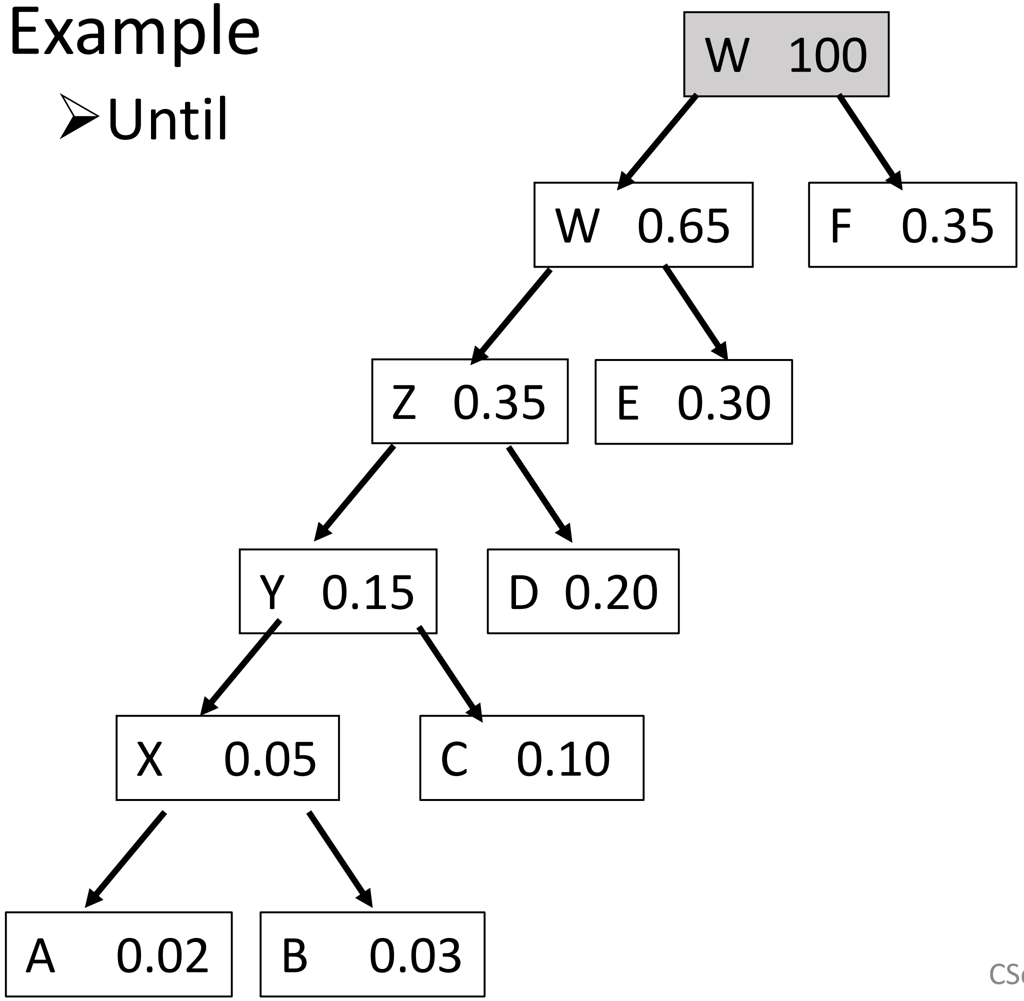


Step 3



Huffman coding

- Example
  - Until



Each priority queue operation (e.g. heap): O(log n)
In each iteration: 1 less subtree.
Initially: n subtrees.
Total: O(n log n) time.

Huffman coding

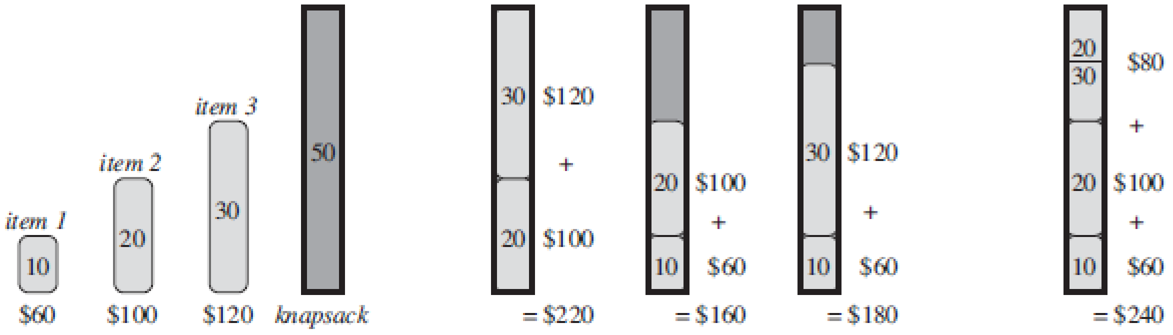
- Correctness
  - Greedy Choice Property:
    - There exists a minimum cost prefix tree where the 2 smallest frequency characters are indeed siblings with the longest path from root.
    - → the greedy choice does not "hurt" finding the optimum
  - Optimal Substructure Property:
    - An optimal solution to the problem once we choose the 2 least frequent elements
    - and combine them to produce a smaller problem,
      - is a solution to the problem when the 2 elements are added.

Applications

- **Kruskal's Minimum Spanning Tree (MST)**
  - Create a MST by picking edges one by one.
  - **Greedy Choice**: pick the **smallest weight edge that doesn't cause a cycle in the MST constructed so far**.
- **Prim's Minimum Spanning Tree**
  - Create a MST by picking edges one by one.
  - Maintain 2 sets: set of the vertices already included in MST and the set of the vertices not yet included.
  - **Greedy Choice**: pick the **smallest weight edge that connects the 2 sets**.
- **Dijkstra's Shortest Path** (similar to Prim's algorithm)
  - The shortest path tree is built up, edge by edge.
  - Maintain 2 sets: set of the vertices already included in the tree and the set of the vertices not yet included.
  - **Greedy Choice**: pick the edge that connects the 2 sets, and is on the smallest weight path from source to the set that contains not yet included vertices.
- **Huffman Coding**
  - lossless compression technique.
  - It assigns variable length bit codes to different characters.
  - **Greedy Choice**: assign **least bit length code** to the **most frequent character**.

Conclusion

- Greedy
  - Short term policy: makes the choice that looks best at the moment.
    - Once it is done, it is done and we are close to the final solution
  - They do not always yield optimal solutions,
    - **but** for many problems they do



Fractional 0-1

Questions ?

- Reading
  - ➢Canvas Csci 115 book – Chapter 9
  - ➢Introduction to Algorithms, Chapter 16.