Greedy Algorithms
Coin Change Problem

(Class 23)

Greedy Algorithm

- An optimization problem is one in which you want to find, not just a solution, but the best solution.
- Search techniques look at many possible solutions. E.g., dynamic programming or backtrack search.
- A “greedy algorithm” sometimes works well for optimization problems.

- A greedy algorithm works in phases.
- At each phase:
  - You take the best you can get right now, without regard for future consequences.
  - You hope that by choosing a local optimum at each step, you will end up at a global optimum.

- For some problems, greedy approach always gets optimum.
- For others, greedy finds good, but not always best.
- If so, it is called a greedy heuristic, or approximation.
- For still others, greedy approach can do very poorly.

Example: Counting Money

- Suppose you want to count out a certain amount of money, using the fewest possible bills (notes) and coins.
- A greedy algorithm to do this would be:
- At each step, take the largest possible note or coin that does not overshoot.

```
while (N > 0)
{
    give largest denomination coin ≤ N   // N is the total money we want to give
    reduce N by value of that coin
}
```

- Consider the currency in U.S.A.
- There are paper notes for one dollar, five dollars, ten dollars, twenty dollars, fifty dollars and hundred dollars.
- The notes are also called “bills”.

- The coins are one cent, five cents (called a “nickel”), ten cents (called a “dime”) and twenty-five cents (a “quarter”).
- In Pakistan, the currency notes are ten rupees, fifty rupees, hundred rupees, five hundred rupees and thousand rupees.
- The coins are one rupee and five rupees.

- Suppose you are asked to give change of $6.39 (six dollars and thirty-nine cents), you can choose:
  - a $5 note
  - a $1 note to make $6
  - a 25 cents coin (quarter), to make $6.25
  - a 10 cents coin (dime), to make $6.35
  - four 1 cent coins, to make $6.39

- Notice how we started with the highest note, $5, before moving to the next lower denomination.
- Formally, the Coin Change problem is stated as:
  - Given k denominations $d_1, d_2, \ldots, d_k$ and given N, find a way of writing:

$$N = i_1 d_1 + i_2 d_2 + \cdots + i_k d_k$$

  - such that

$$i_1 + i_2 + \cdots + i_k \; is \; minimized$$

- The "size" of problem is k (we have total k types of coins).
- The greedy strategy works for the coin change problem but not always.

- Here is an example where it fails.
- Suppose we have 1 rupee, 7 rupees, and 10 rupees coins.
- Using a greedy algorithm to count out 15 rupees, you would get:
  - A 10 rupees piece
  - Five 1-rupee coins, for a total of 15 rupees.
- This requires total six coins.

- A better solution, however, would be to use two 7 rupees coins and one 1-rupee coin.
- This only requires three coins.
- In this case, greedy algorithm results in a solution, but not in an optimal solution.

When Greedy Strategy works for Coin Change Problem?

- The greedy approach gives us an optimal solution when the coins are all powers of a fixed denomination.

$$N = i_0 D^0 + i_1 D^1 + i_2 D^2 + + i_k D^k$$

- Note that this is N represented in based D.

- U.S.A coins are multiples of 5:
  - 5 cents – nickel – (5^1)
  - 10 cents – dime – (multiple of 5; 5×2)
  - 25 cents – quarter – (5^2)
- Coins should be multiple or power of a fixed base then the greedy algorithm will work optimally.
- If denominations not multiple or power of a fixed base then we have to use dynamic programming approach.

Making Change: Dynamic Programming Solution

- The general coin change problem can also be solved using Dynamic Programming.
- Set up a Table, $C[1..k, 0..N]$.
- k are the denominations.
- N is the total amount we have to return in the form of coins.

- Where the rows denote available denominations, d_i where $1 \leq i \leq k$.
- And columns denote the amount from $0 \ldots N$ units where $0 \leq j \leq N$.
- $C[i, j]$ denotes the minimum number of coins, required to pay an amount j using only coins of denominations 1 to i.
- $C[k, N]$ is the solution required.

- To pay an amount j units, using coins of denominations 1 to i, we have two choices:
  1. either chose NOT to use any coins of denomination i,
  2. or chose at least one coin of denomination i, and also pay the remaining amount $(j - d_i)$.

- To pay $(j - d_i)$ units it takes $C[i, j - d_i]$ coins. Thus,

$$C[i, j] = 1 + C[i, j - d_i]$$

- Since we want to minimize the number of coins used,

$$C[i, j] = min(C[i - 1, j], 1 + C[i, j - d_i])$$

- This approach is exactly like knapsack problem.

- Here is the dynamic programming-based algorithm for the coin change problem.

```
COINS(N)
1  d[1..k] ← {1, 4, 6}   // (coinage, for example)
2  for i ← 1 to k
3    do c[i, 0] ← 0     // if return amount is 0 then no coin wll be returned
4  for i ← 1 to k
5    for j ← 1 to N
6      if (i ← 1 & j < d[i])
7          then c[i, j] ← ∞
8      else if (i ← 1)
9          then c[i, j] ← 1 + c[1, j – d[1]]
10     else if (j < d[i])
11         then c[i, j] ← c[i – 1, j]
12     else c[i, j] ← min (c[i – 1, j], 1 + c[i, j – d[i]])
13 return c[k,N]
```

Complexity of Coin Change Algorithm

- Greedy algorithm (non-optimal) takes $O(k)$ time.
- While dynamic programming approach takes $O(k.N)$ time.

Greedy Algorithm: Huffman Encoding

(From Book's Page Number 431)

- The Huffman codes provide a method of encoding data efficiently.
- Normally, when characters are coded using standard codes like ASCII.
- Each character is represented by a fixed-length codeword of bits, e.g., 8 bits per character.

- Fixed-length codes are popular because it is very easy to break up a string into its individual characters,
- And to access individual characters and substrings by direct indexing.
- However, fixed-length codes may not be the most efficient from the perspective of minimizing the total quantity of data.

- Consider the string “ abacdaacac”.
- If the string is coded with ASCII codes, the message length would be:

$$10 \times 8 = 80\ bits$$

- We will see that the same string encoded with a variable length Huffman encoding scheme will produce a shorter message.