COMP 6651

Algorithm Design Techniques

Week 2

Divide and Conquer. Master Theorem.
Greedy Algorithms

(some material is taken from web or other various sources with permission)

Recursion

- Natural and Elegant for many problems
  - F(n) = (n==1) ? 1 : n*F(n-1)
- Inefficient if we are not careful

$$p(x,n) = \begin{cases} 1 & \text{if } n = 0 \\ x \cdot p(x, n-1) & \text{else} \end{cases}$$

$$p(x,n) = \begin{cases} 1 & \text{if } x = 0 \\ x \cdot p(x,(n-1)/2)^2 & \text{if } x > 0 \text{ is odd} \\ p(x, n/2)^2 & \text{if } x > 0 \text{ is even} \end{cases}$$

Recursion

- Natural and Elegant for many problems
  - F(n) = (n==1) ? 1 : n*F(n-1)
- Inefficient if we are not careful
- Difficult to analyze?
  - How many times it is called?
- Better to avoid recursive calls

Recursion

- Structure
  - Base case(s)
  - Recursive calls that are guaranteed to reach a base case!!!!!

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \quad \text{for all integers } n, k : 1 \leq k \leq n-1,$$

Recursion Pipeline

1. Given a problem, create the recursive algorithm
   - F(n) = (n==1) ? 1 : n*F(n-1)
2. Analyze?
   - Easy with Master theorem
3. Make a non-recursive version that has the same complexity

Master Theorem

1. What is the structure of a recursive algorithm:

```
recursive_algorithm(input_of_size_n){

    // non recursive part (includes the base case
analysis)
    execute_some_instructions(n)

    // recursive parts called a times
    recursive_algorithm(input_of_size_n/b);
    recursive_algorithm(input_of_size_n/b);
    ...

    recursive_algorithm(input_of_size_n/b);

}
```

Example - binary search

```
/* n input size, n>0 */
/* v - sorted array of integers of size n */
/* x - integer number */
/* return an index k whose value v[k]==x OR -1 otherwise */
int findElement(v, x){
    return findElementI(v, 0, n-1, x);
}
int findElementI(v, int i1, int i2, x){
    if(i1==i2){
        if(v[i1]==x){
            return i1;
        }
        return -1;
    }
    int middle = (i1+i2)/2;
    if(x<=v[middle]){
      return findElementI(v, i1, middle, x);
    } else {
      return findElementI(v, middle + 1, i2, x)
   }
```

Example - Merge-Sort

```
MERGE-SORT(A, p, r)
  if p < r                          // check for base case
      q = ⌊(p + r)/2⌋               // divide
      MERGE-SORT(A, p, q)           // conquer
      MERGE-SORT(A, q + 1, r)       // conquer
      MERGE(A, p, q, r)             // combine
```

Example

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \quad \text{for all integers } n, k : 1 \le k \le n-1,$$

```
/* compute n choose k */
int n_choose_k(int n, int k){

      if(k==0)
            return 1;

       if(n==k)
            return 1;

    int a = n_choose_k(n-1, k-1);
    int b = n_choose_k(n-1, k);

    return a + b;
}
```

Master Theorem

What is the structure of a recursive algorithm:

```
recursive_algorithm(input_of_size_n){

    // non recursive part (includes the base case analysis)
    execute_some_instructions(n)

    // recursive parts called a times
    recursive_algorithm(input_of_size_n/b);
    recursive_algorithm(input_of_size_n/b);
    ...

    recursive_algorithm(input_of_size_n/b);

}
```

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Master Theorem

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Three cases:

1. If $f(n) = O\left(n^{log_b(a)-\varepsilon}\right)$ for some constant ε then $T(n) = \theta(n^{log_b(a)})$
2. If $f(n) = \theta\left(n^{log_b(a)}\right)$ then $T(n) = \theta(n^{log_b(a)}\lg(n))$
3. If $f(n) = \Omega\left(n^{log_b(a)+\varepsilon}\right)$ for some constant ε and if $\mathrm{a} \cdot f\left(\frac{n}{b}\right) \le c \cdot f(n)$ for some constant $c < 1$ and all sufficiently large n then $T(n) = \theta(f(n))$

Master Theorem

Theoretical examples:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

$$T(n) = 9 \cdot T\left(\frac{n}{3}\right) + n$$

$$T(n) = T\left(\frac{2n}{3}\right) + 1$$

$$T(n) = 3 \cdot T\left(\frac{n}{4}\right) + nlgn$$

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + nlgn$$

Recursion

- F(n) = (n==1) ? 1 : n*F(n-1)

$$p(x,n) = \begin{cases} 1 & \text{if } n = 0 \\ x \cdot p(x, n-1) & \text{else} \end{cases}$$

$$p(x,n) = \begin{cases} 1 & \text{if } x = 0 \\ x \cdot p(x, (n-1)/2)^2 & \text{if } x > 0 \text{ is odd} \\ p(x, n/2)^2 & \text{if } x > 0 \text{ is even} \end{cases}$$

Example - binary search

```
/* n input size, n>0 */
/* v - sorted array of integers of size n */
/* x - integer number */
/* return an index k whose value v[k]==x OR -1 otherwise */
int findElement(v, x){
    return findElementI(v, 0, n-1, x);
}
int findElementI(v, int i1, int i2, x){
    if(i1==i2){
        if(v[i1]==x){
            return i1;
        }
        return -1;
    }
    int middle = (i1+i2)/2;
    if(x<=v[middle]){
      return findElementI(v, i1, middle, x);
    } else {
      return findElementI(v, middle + 1, i2, x)
   }
```

Example - Merge-Sort

```
MERGE-SORT(A, p, r)
  if p < r                                // check for base case
      q = ⌊(p + r)/2⌋                     // divide
      MERGE-SORT(A, p, q)                 // conquer
      MERGE-SORT(A, q + 1, r)             // conquer
      MERGE(A, p, q, r)                   // combine
```

Example

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \quad \text{for all integers } n, k : 1 \leq k \leq n-1,$$

```
/* compute n choose k */
int n_choose_k(int n, int k){

      if(k==0)
            return 1;

       if(n==k)
            return 1;

    int a = n_choose_k(n-1, k-1);
    int b = n_choose_k(n-1, k);

    return a + b;
}
```

Proof of Master Theorem

- Simplified proof assuming that $log_b n \in \mathbb{N}$
- In other words n is an integer power of b

Greedy Algorithms

- ***Compression problem***
- ***Huffman codes***

Definitions

- **<u>Alphabet</u>**: Finite set containing at least one element: $\mathbf{A} = \{a, b, c, d, e\}$
- **<u>Symbol</u>**: Alphabet element: $s \in \mathbf{A}$
- **<u>String (over alphabet)</u>**: Sequence of symbols: ccdabdcaad…
- **<u>Codeword</u>**: Sequence of bits representing coded symbol or string:
  1101010011010101100…
- p_i: Occurrence probability of symbol s_i in input string

$$\sum_{\forall i \in A} p_i = 1$$

Code types

- **Fixed-length codes** - all codewords have same length (number of bits)

  A – 000, B – 001, C – 010, D – 011, E – 100, F – 101

- **Variable-length codes** - may give different lengths to codewords

  A – 0, B – 00, C – 110, D – 111, E – 1000, F – 1011

Code types (cont.)

- **Prefix code** - No codeword is prefix of any other codeword

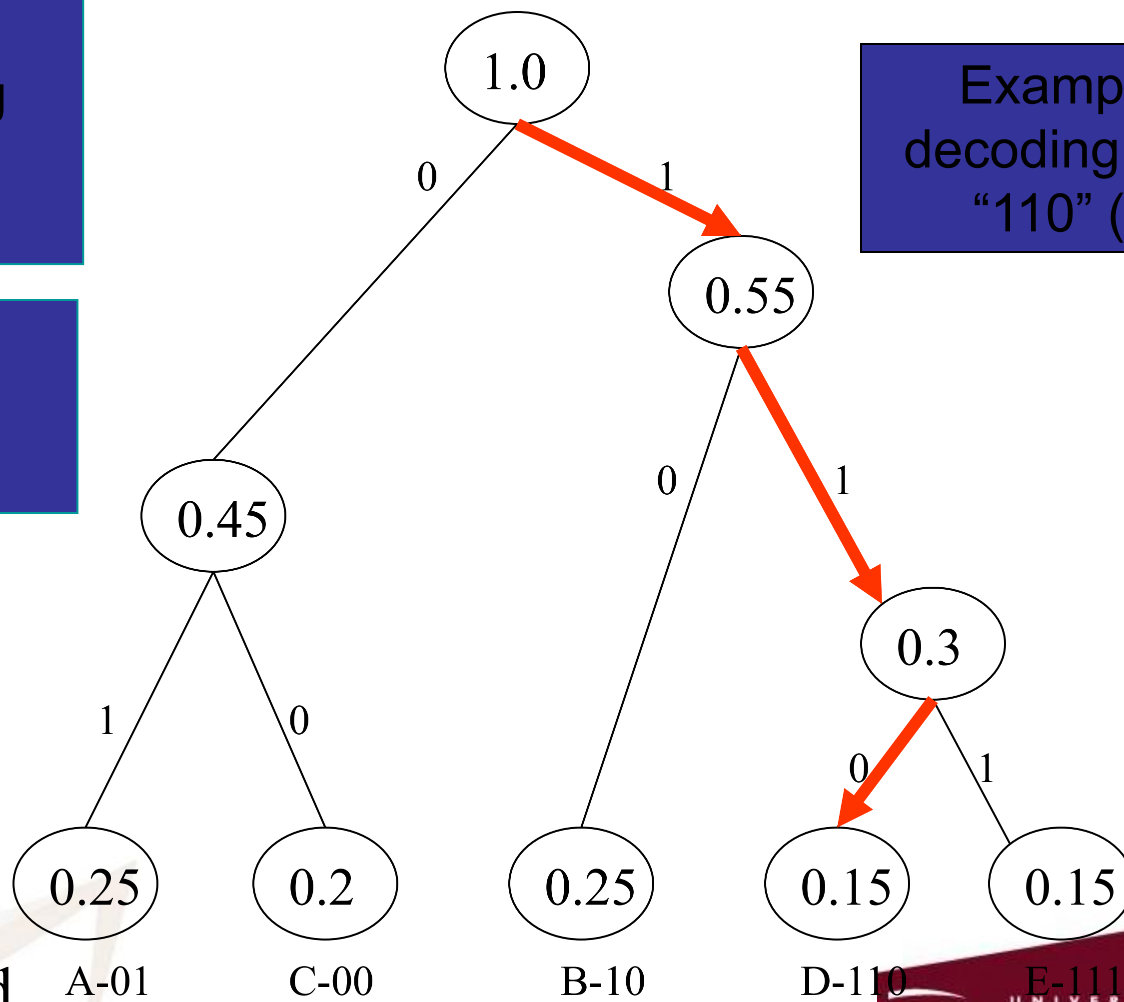
  A = 0;   B = 10;   C = 110; D = 111
- **Uniquely decodable code** - Has only one possible source string producing it
  - Unambigously decoded
  - Examples:
    - Prefix code - end of codeword immediately recognized (without ambiguity) : 010011001110 → 0 | 10 | 0 | 110 | 0 | 111 | 0
    - Fixed-length code

Huffman tree example

Each codeword determined according to path from root to symbol

When decoding tree traverse tree from root.

Probabilities



Example: decoding input “110” (D)

codewords:

Greedy Algorithms

- ***Algorithm where the optimal local solution is taken at each time step***
- ***Does it lead to a global solution?***
- ***… sometimes …***
- ***How can you tell if they do?***
- ***Theory….***