Review of Algorithms

Young-Rae Cho, Ph.D.
Professor
Division of Software / Division of Digital Healthcare
Yonsei University – Mirae Campus

Algorithms

- Definition
  - A process that performs a sequence of operations
  - A series of well-defined instructions to perform a specific task

- How to express algorithms?
  - Natural Language
  - Flow Chart or Diagram
  - Programming Language
  - Pseudocode

- How to evaluate algorithms?
  - Correctness
  - Efficiency
    - Must run in a reasonable time
    - Big-O notation is used

Overview

- [x] Exhaustive Search
- [ ] Divide-and-Conquer Algorithm
- [ ] Dynamic Programming
- [ ] Greedy Algorithm
- [ ] Randomized Algorithm

Exhaustive Search

- Process
  - Examine all possible cases to find a solution
  - Also, called brute force search
- Features
  - Simple
  - Sometimes, very inefficient because of combinatorial explosion
- Example
  - Selection sort
- Alternatives
  - Random Sampling
  - Branch and bound algorithm
  - Anti-monotonic property

Selection Sort

- Algorithm
  - Iteratively search the smallest one

$$
\begin{aligned}
&\textsc{SelectionSort}(a, n)\\
&\quad for\ \ i \leftarrow 1\ \ to\ \ n-1\\
&\qquad a_j \leftarrow smallest\ \ one\ \ between\ \ a_i\ \ and\ \ a_n\\
&\qquad swap\ \ a_i\ \ and\ \ a_j\\
&\quad return\ \ a
\end{aligned}
$$

- Runtime?

Anti-monotonic Property

- Definition
  - If a case satisfies a condition, then more general cases always satisfy it
  - If a case violates a condition, then more specific cases always violate it

- Example
  - Find maximal sized sets of genes that occur together in at least two functions

Function ID	Genes
10	A, B, C
20	C, D, F
30	A, C, E
40	A, B, C, E

Overview

- ❑ Exhaustive Search
- ✓ Divide-and-Conquer Algorithm
- ❑ Dynamic Programming
- ❑ Greedy Algorithm
- ❑ Randomized Algorithm

Divide-and-Conquer Algorithm

- Process
  - (1) Recursively splitting the problem into smaller sub-problems
  - (2) Solve the smallest sub-problem independently
  - (3) Recursively merging the solutions of sub-problems until having a solution of the original problem

- Features
  - Improve efficiency

- Example
  - Merge sort
  - Quick sort

Merge Sort

- Algorithm
  - (1) Recursively divide the array
  - (2) Recursively combine two arrays in a sorted order

$$
\begin{array}{l}
\textsc{MergeSort}(A[1..n]) \\
\quad if \;\; (n > 1) \\
\qquad m \leftarrow \lfloor n/2 \rfloor \\
\qquad \textsc{MergeSort}(A[1..m]) \\
\qquad \textsc{MergeSort}(A[m+1..n]) \\
\qquad \textsc{Merge}(A[1..n], m)
\end{array}
$$

$$
\begin{array}{l}
\textsc{Merge}(A[1..n], m) \\
\quad i \leftarrow 1; \;\; j \leftarrow m+1 \\
\quad for \;\; k \leftarrow 1 \;\; to \;\; n \\
\qquad if \;\; i > m \\
\qquad\quad B[k] \leftarrow A[j]; \;\; j \leftarrow j+1 \\
\qquad else \;\; if \;\; j > n \\
\qquad\quad B[k] \leftarrow A[i]; \;\; i \leftarrow i+1 \\
\qquad else \;\; if \;\; A[i] > A[j] \\
\qquad\quad B[k] \leftarrow A[j]; \;\; j \leftarrow j+1 \\
\qquad else \;\; if \;\; A[i] < A[j] \\
\qquad\quad B[k] \leftarrow A[i]; \;\; i \leftarrow i+1 \\
\quad for \;\; k \leftarrow 1 \;\; to \;\; n \\
\qquad A[k] \leftarrow B[k]
\end{array}
$$

- Runtime?

Quick Sort

- Algorithm
  - (1) Recursively divide the array based on the pivot
  - (2) Recursively combine two arrays

```
QuickSort(A[1..n])
    if (n > 1)
        k ← Partition(A, p)
        QuickSort(A[1..k − 1])
        QuickSort(A[k + 1..n])
```

```
Partition(A[1..n], p)
    if (p ≠ n) swap A[p] and A[n]
    i ← 0; j ← n
    while(i < j)
        repeat i ← i + 1
            until i = j or A[i] ≥ A[n]
        repeat j ← j − 1
            until i = j or A[j] ≤ A[n]
        if(i < j)
            swap A[i] and A[j]
    if (i ≠ n) swap A[i] and A[n]
    return i
```

- Runtime?

Overview

- ❑ Exhaustive Search
- ❑ Divide-and-Conquer Algorithm
- ☑ Dynamic Programming
- ❑ Greedy Algorithm
- ❑ Randomized Algorithm

Dynamic Programming

- **Process**
  1) Formulate the problem recursively by breaking it down into sub-problems
  2) Build solutions in a linear fashion

     (Repeatedly use the result of a sub-problem to solve the next sub-problem)

- **Features**
  - Optimization (finding an optimal solution)
  - Memoization (storing results of intermediate sub-problems)

- **Examples**
  - Sequence alignment
  - Binary search tree

Dynamic Programming Example (1)

- **Rule**
  - 2 piles of rocks
  - A player may take either
    1 rock (from either pile)
    or 2 rocks (1 from each)
  - The player who takes
    the last rock wins

	0	1	2	3	4	5	6	7	8	9	10
0		W	L	W	L	W	L	W	L	W	L
1	W	W	W	W	W	W	W	W	W	W	W
2	L	W	L	W	L	W	L	W	L	W	L
3	W	W	W	W	W	W	W	W	W	W	W
4	L	W	L	W	L	W	L	W	L	W	L
5	W	W	W	W	W	W	W	W	W	W	W
6	L	W	L	W	L	W	L	W	L	W	L
7	W	W	W	W	W	W	W	W	W	W	W
8	L	W	L	W	L	W	L	W	L	W	L
9	W	W	W	W	W	W	W	W	W	W	W
10	L	W	L	W	L	W	L	W	L	W	L

- **Winning / Losing**

$$
\begin{cases}
R_{1,0} \leftarrow W \\
R_{0,1} \leftarrow W \\
R_{1,1} \leftarrow W \\
R_{i,j} \leftarrow L \quad if \;\; R_{i-1,j} = W \;\; (where\ i \geq 1) \;\; and \\
\qquad\qquad\qquad R_{i,j-1} = W \;\; (where\ j \geq 1) \;\; and \\
\qquad\qquad\qquad R_{i-1,j-1} = W \;\; (where\ i \geq 1\ and\ j \geq 1) \\
R_{i,j} \leftarrow W \quad Otherwise
\end{cases}
$$

Dynamic Programming Example (2)

Rule

- A user chooses any positive integer n
- Finds the minimum number of operations to transform n to 1, out of the followings
  - If n is a multiple of 3, then divide n by 3
  - If n is a multiple of 2, then divide n by 2
  - n - 1

Solution

- Recursive formula

Overview

- ❑ Exhaustive Search
- ❑ Divide-and-Conquer Algorithm
- ❑ Dynamic Programming
- ☑ Greedy Algorithm
- ❑ Randomized Algorithm

Greedy Algorithm

Process

1) Determine the optimal structure of a problem
2) Find the local optimal solution at each step

Features

- Local optimization

Examples

- Huffman codes
- Minimum spanning tree

Greedy Algorithm Examples

- **Money Counting**
  - Count a certain amount of money using the fewest bills and coins
  - Local optimum solution: Take the largest bill or coin at each step
  - Example: 1620 won

- **Scheduling**
  - Assign m jobs into n processors to finish all the jobs as early as possible (m > n)
  - Local Optimum Solution ?
  - Example: 9 jobs on 3 processors

    (runtimes of 9 jobs are 3, 5, 6, 10, 11, 14, 15, 18, and 20 min.)

Overview

- ❑ Exhaustive Search
- ❑ Divide-and-Conquer Algorithm
- ❑ Dynamic Programming
- ❑ Greedy Algorithm
- ✓ Randomized Algorithm

Randomized Algorithm

❑ Process

- Examine random samples to find a solution

❑ Features

- Simple
- Probabilistic
- Sometimes, non-deterministic

• **Deterministic algorithm**:

always produce the same solution given a particular input

• **Non-deterministic algorithm**:

allows multiple solutions based on an input or random choices

Bolts and Nuts

- **Problem**
  - Among n nuts, find the nut that matches a given bolt

- **Expected Number of Comparison ?**
  - $T(n)$: number of comparison to find a match for a single bolt out of n nuts
  - $$E[T(n)] = \sum_{k=1}^{n-1} k \cdot Pr[T(n) = k]$$
  - $$Pr[T(n) = k] = \begin{cases} 1/n & if \;\; k < n-1 \\ 2/n & if \;\; k = n-1 \end{cases}$$
  - $$E[T(n)] = \frac{n+1}{2} - \frac{1}{n}$$

Questions?

- Lecture Slides on the Course Website, “https://ads.yonsei.ac.kr/faculty/biocomputing”