Brute Force Algorithm

Algorithm

2014 Fall Semester

SUNG KYUN KWAN UNIVERSITY

Brute Force

- A straightforward approach, usually based directly on the problem's statement and definitions of the concepts involved

- Examples:
  - Computing a^n ($a > 0$, n a nonnegative integer)
  - Computing $n!$
  - Multiplying two matrices
  - Searching for a key of a given value in a list

Concepts of Brute Force Algorithm

- A brute-force algorithm solves a problem in the most **simple**, direct or obvious way. As a result, such an algorithm can end up doing far **more work** to solve a given problem than a more clever or sophisticated algorithm might do. On the other hand, a brute-force algorithm is often **easier to implement** than a more sophisticated one and, because of this simplicity, sometimes it can be more efficient.
- Typically, a brute-force algorithm solves such a problem by **exhaustively enumerating all the possibilities**. I.e., for every decision we consider each possible outcome.

The Simplest Approach

- Brute Force - the simplest of the design strategies
  - Is a straightforward approach to solving a problem, usually directly based on the problem's statement and definitions of the concepts involved.
  - Just do it - the brute-force strategy is easiest to apply.
  - Results in an algorithm that can be improved with a modest amount of time.
- Brute force is important due to its wide applicability and simplicity.

Examples of Brute Force Algorithm

- Selection Sort
- Bubble Sort
- String Matching
- Sequential Search
- Traveling Salesman Problem (Hamilton Circuits)
- Knapsack Problem
- Job Assignment Problem

Selection Sort

- Find its smallest element by scanning the list
- Exchange it with the first element making the smallest element in its final position in the sorted list.
- Scan the list, starting with the second element to find the smallest among the last n-1.
- The second element will be put in its final position.

Selection Sort – how to work

| 89 , 45 , 68 , 90 , 29 , 34 , **17**

17 | 45 , 68 , 90 , **29** , 34 , 89

17 , 29 | 68 , 90 , 45 , **34** , 89

17 , 29 , 34 | 90 , **45** , 68 , 89

17 , 29 , 34 , 45 | 90 , **68** , 89

17 , 29 , 34 , 45 , 68 | 90 , **89**

17 , 29 , 34 , 45 , 68 , 89 | 90

Bubble Sort

- Compare adjacent elements of the list.
- Exchange them if they are out of order.
- By doing it repeatedly, we end up “Bubbling up” the largest element to the last position on the list
- The next pass bubbles up the second largest element, and so on until, after n-1 passes, the list is sorted.

Bubble Sort – how to work

89 45 68 90 29 34 17
89 ↔ 45 68 90 29 34 17
45 89 ↔ 68 90 29 34 17
45 68 89 ↔ 90 ↔ 29 34 17
45 68 89 29 90 ↔ 34 17
45 68 89 29 34 90 ↔ 17
45 68 89 29 34 17 | **90**

45 ↔ 68 ↔ 89 ↔ 29 34 17 | **90**
45 68 29 89 ↔ 34 17 | **90**
45 68 29 34 89 ↔ 17 | **90**
45 68 29 34 17 | **89** **90**

etc..

String Matching

- Given a string of n characters called the **text** and a string of m characters (m<=n) called the **pattern,** find a substring of the text that matches the pattern.

- A brute force algorithm for string matching:
  - Align the pattern against the first m characters of the text.
  - Start matching the corresponding pair of characters from left to right until either all the m pairs are match.
  - Or if the missing pair is found, the pattern is shifted one position to the right and character comparisons are resumed, starting again from the 1^{st} character.

String Matching – how to work

N O B O D Y _ N O T I C E D _ H I M   Text

N O T   Pattern

N

N

……

N O T

SUNG KYUN KWAN UNIVERSITY

Sequential Search

- Given an array A.1,...,A.N and target value V.
- Search routine should return an index of V in A, if V is present in the array, and N+1 otherwise.

Given array – 12 24 35 42 46 49 50 82 99

Target value – 52

Return value : 10

Given array – 12 24 35 42 46 49 50 82 99

Target value – 42

Return value : 4

Exhaustive Search

- Exhaustive search is a brute-force approach to combinatorial problems.
  - It suggests generating each and every combinatorial object of the problem, selecting those of them of that satisfy the problem's constraints and then finding a desired object.
  - Impractical for all but applicable to very small instances of problems.
- Examples:
  - Traveling salesman problem
    - Finding the shortest tour through a given set of *n* cities that visits each city exactly once before returning to the city where it started.
  - Knapsack problem
    - Finding the most valuable list of out-of *n* items that fit into the knapscak.
  - Job Assignment problem
    - Finding an assignment of n people to execute *n* jobs with the smallest total cost.

Traveling Salesman Problem

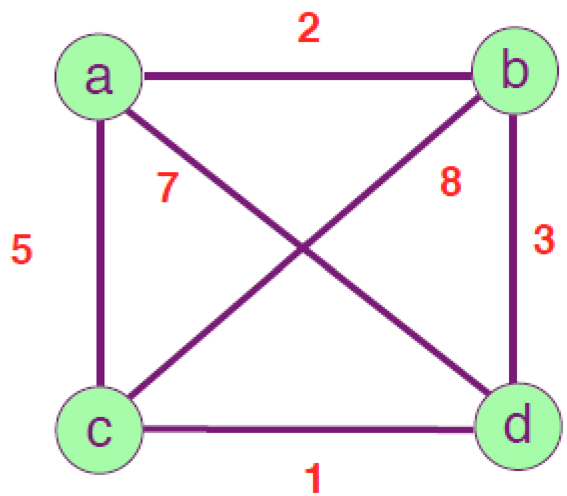
- Given : A complete, weighted graph
- Find : A Hamilton circuit of minimum weight

※ A circuit that starts at a vertex of a graph, passes through every vertex exactly once, and returns to the starting vertex is called a HAMILTON CIRCUIT.

- Algorithm
  - List all Hamilton circuits
  - Compute the sum of weights (total weight) for each circuit
  - Choose the one with the smallest total weight

TSP – how to work

Tour	Length
a b c d a	2+8+1+7 = 18
a b d c a	2+3+1+5 = **11**
a c b d a	23
a c d b a	**11**
a d b c a	23
a d c b a	18



More than one optimal solutions.

Hamilton Circuits

Step 1: Choose a starting point.

Step 2: List all the Hamilton circuits with that starting point.

Step 3: Find the total weight of each circuit.

Step 4: Choose a Hamilton circuit with the smallest total weight.

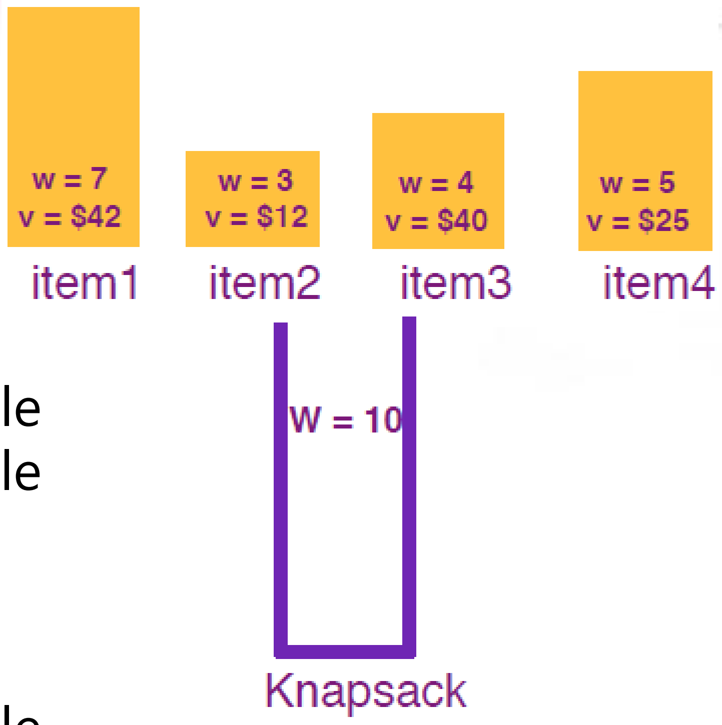
Knapsack Problem

- Given ***n*** items of known weights $w_1, \ldots, w_n$ and values $v_1, \ldots, v_n$ and a knapsack of capacity ***W***, find the most valuable subset of the items that fit into the knapsack.

- Consider all the subsets of the set of ***n*** items given,
  - Computing the total weight of each subset in order to identify feasible subsets (the ones with the total not exceeding the knapsack's capacity).
  - Finding a subset of the largest value among them.

Knapsack – how to work

Subset	Total weight	Total value
Null	0	$0
{1}	7	$42
{2}	3	$12
{3}	4	$40
{4}	5	$25
{1,2}	10	$36
{1,3}	11	not feasible
{1,4}	12	not feasible
{2,3}	7	$52
{2,4}	8	$37
{3,4}	9	$65
{1,2,3}	14	not feasible
{1,2,4}	15	not feasible
{1,3,4}	16	not feasible
{2,3,4}	12	not feasible
{1,2,3,4}	19	not feasible



Job Assignment Problem

- There are n people who need to be assigned to n jobs, one person per job. The cost of assigning person i to job j is $C[i,j]$. Find an assignment that minimizes the total cost.
- Select one element in each row so that all selected elements are in different columns and the total sum of the selected elements is the smallest possible.

Job Assignment Problem

	Job 1	Job 2	Job 3	Job 4
Person 1	**9**	**2**	**7**	**8**
Person 2	**6**	**4**	**3**	**7**
Person 3	**5**	**8**	**1**	**8**
Person 4	**7**	**6**	**9**	**4**

$$C = \begin{bmatrix} 9 & 2 & 7 & 8 \\ 6 & 4 & 3 & 7 \\ 5 & 8 & 1 & 8 \\ 7 & 6 & 9 & 4 \end{bmatrix}$$

Assignment (col.#s)	Total Cost
1, 2, 3, 4	9+4+1+4=18
1, 2, 4, 3	9+4+8+9=30
1, 3, 2, 4	9+3+8+4=24
1, 3, 4, 2	9+3+8+6=26
1, 4, 2, 3	9+7+8+9=33
1, 4, 3, 2	9+7+1+6=23

etc.

Strengths and Weaknesses

- <u>Strengths</u>
  - wide applicability
  - simplicity
  - yields reasonable algorithms for some important problems
    (e.g., matrix multiplication, sorting, searching, string matching)

- <u>Weaknesses</u>
  - rarely yields efficient algorithms
  - some brute-force algorithms are unacceptably slow
  - not as constructive/creative as some other design techniques

Q & A