

# Brute Force

Dr. Yingwu Zhu

# Brute Force

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

Examples:


1. Computing  $a^n$  ( $a > 0$ ,  $n$  a nonnegative integer)
2. Computing  $n!$
3. Multiplying two matrices
4. Searching for a key of a given value in a list

# Outline

- Sorting
  - Selection sort
  - Bubble sort
- String matching
- Close-pair problem
- Exhaustive search to combinatorial problems
  - Travelling salesman problem (TSP)
  - Knapsack problem
  - Job assignment problem

# Brute-Force Sorting Algorithm

Selection Sort Scan the array to find its smallest element and swap it with the first element. Then, starting with the second element, scan the elements to the right of it to find the smallest among them and swap it with the second element. Generally, on pass  $i$  ( $0 \leq i \leq n-2$ ), find the smallest element in  $A[i..n-1]$  and swap it with  $A[i]$ :

$A[0] \leq \dots \leq A[i-1] \mid A[i], \dots, A[\min], \dots, A[n-1]$   
in their final positions                      

Example: 7 3 2 5

# Analysis of Selection Sort

**ALGORITHM** *SelectionSort*( $A[0..n - 1]$ )  
//Sorts a given array by selection sort  
//Input: An array  $A[0..n - 1]$  of orderable elements  
//Output: Array  $A[0..n - 1]$  sorted in ascending order  
**for**  $i \leftarrow 0$  **to**  $n - 2$  **do**  
     $min \leftarrow i$   
    **for**  $j \leftarrow i + 1$  **to**  $n - 1$  **do**  
        **if**  $A[j] < A[min]$   $min \leftarrow j$   
        swap  $A[i]$  and  $A[min]$

$T(n) = ?$

# Bubble Sort

- Exchange sort, makes  $n-1$  scans over the list. On each pass, swap two neighbor data items only if they are out of order. Each pass  $i$  will bubbling up the  $(i+1)$ -th largest item to the right position. Pass  $i$  ( $0 \leq i \leq n-2$ ) can be represented as follows:

$A[0], \dots, A[j] \quad A[j+1], \dots, A[n-i-1] \mid A[n-i] \leq \dots \leq A[n-1]$   
in their final positions

# Bubble Sort

Algorithm BubbleSort(A[0..n-1])

for  $i \leftarrow 0$  to  $n-2$  do

for  $j \leftarrow 0$  to  $n-2$  do //or  $n-2-i$ , why ?

if  $A[j+1] < A[j]$  swap  $A[j]$  and  $A[j+1]$

$T(n) = ?$

# Brute-Force String Matching

- pattern: a string of  $m$  characters to search for
- text: a (longer) string of  $n$  characters to search in
- problem: find a substring in the text that matches the pattern
- Examples:
  1. Pattern: 001011  
Text: 10010101101001100101111010
  2. Pattern: happy  
Text: It is never too late to have a happy childhood.



# Examples of Brute-Force String Matching

1. Pattern: 001011

Text: 10010101101001100101111010

2. Pattern: happy

Text: It is never too late to have a  
happy childhood.

# Brute-Force String Matching

## Brute-force algorithm

Step 1 Align pattern at beginning of text

Step 2 Moving from left to right, compare each character of pattern to the corresponding character in text until

- all characters are found to match (successful search); or
- a mismatch is detected

Step 3 While pattern is not found and the text is not yet exhausted, realign pattern one position to the right and repeat Step 2

# Pseudocode and Efficiency

**ALGORITHM** *BruteForceStringMatch*( $T[0..n - 1]$ ,  $P[0..m - 1]$ )

//Implements brute-force string matching  
//Input: An array  $T[0..n - 1]$  of  $n$  characters representing a text and  
// an array  $P[0..m - 1]$  of  $m$  characters representing a pattern  
//Output: The index of the first character in the text that starts a  
// matching substring or  $-1$  if the search is unsuccessful

```
for  $i \leftarrow 0$  to  $n - m$  do  
     $j \leftarrow 0$   
    while  $j < m$  and  $P[j] = T[i + j]$  do  
         $j \leftarrow j + 1$   
    if  $j = m$  return  $i$   
return  $-1$ 
```

Efficiency:

# Closest-Pair Problem

Find the two closest points in a set of  $n$  points  
(in the two-dimensional Cartesian plane).

Brute-force algorithm

How?

# Closest-Pair Problem

Find the two closest points in a set of  $n$  points (in the two-dimensional Cartesian plane).

## Brute-force algorithm

Compute the distance between every pair of distinct points

and return the indexes of the points for which the distance is the smallest.

# Closest-Pair Brute-Force Algorithm (cont.)

**ALGORITHM** *BruteForceClosestPoints(P)*

//Input: A list  $P$  of  $n$  ( $n \geq 2$ ) points  $P_1 = (x_1, y_1), \dots, P_n = (x_n, y_n)$

//Output: Indices  $index1$  and  $index2$  of the closest pair of points

$dmin \leftarrow \infty$

**for**  $i \leftarrow 1$  **to**  $n - 1$  **do**

**for**  $j \leftarrow i + 1$  **to**  $n$  **do**

$d \leftarrow \text{sqrt}((x_i - x_j)^2 + (y_i - y_j)^2)$  //sqrt is the square root function

**if**  $d < dmin$

$dmin \leftarrow d; index1 \leftarrow i; index2 \leftarrow j$

**return**  $index1, index2$

**Efficiency:**

# Brute-Force Strengths and Weaknesses

- Strengths

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

- Weaknesses

- rarely yields efficient algorithms
- some brute-force algorithms are unacceptably slow
- not as constructive as some other design techniques

# Exhaustive Search

A brute force solution to combinatorial problems. It suggests generating each and every combinatorial object (e.g., permutations, combinations, or subsets of a set) of the problem, selecting those of them that satisfying all the constraints, and then finding a desired object.

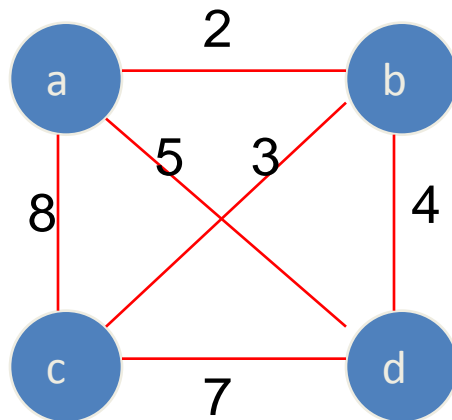
## Method:

- generate a list of all potential solutions to the problem in a systematic manner
- evaluate potential solutions one by one, disqualifying infeasible ones and, for an optimization problem, keeping track of the best one found so far
- when search ends, announce the solution(s) found



# Example 1: Traveling Salesman Problem

- Given  $n$  cities with known distances between each pair, find the shortest tour that passes through all the cities exactly once before returning to the starting city
- Alternatively: Find shortest *Hamiltonian circuit* in a weighted connected graph
- Example:



# TSP by Exhaustive Search

Problem:  $v_0, v_1, \dots, v_{n-1}, v_0 \rightarrow$  permutations of  $n$  vertices

Tour	Cost	
$a \rightarrow b \rightarrow c \rightarrow d \rightarrow a$	$2+3+7+5 = 17$	optimal
$a \rightarrow b \rightarrow d \rightarrow c \rightarrow a$	$2+4+7+8 = 21$	
$a \rightarrow c \rightarrow b \rightarrow d \rightarrow a$	$8+3+4+5 = 20$	
$a \rightarrow c \rightarrow d \rightarrow b \rightarrow a$	$8+7+4+2 = 21$	
$a \rightarrow d \rightarrow b \rightarrow c \rightarrow a$	$5+4+3+8 = 20$	
$a \rightarrow d \rightarrow c \rightarrow b \rightarrow a$	$5+7+3+2 = 17$	optimal

NP-hard problem!

# Example 2: Knapsack Problem

Given  $n$  items:

- weights:  $w_1 w_2 \dots w_n$
- values:  $v_1 v_2 \dots v_n$
- a knapsack of capacity  $W$

Find most valuable subset of the items that fit into the knapsack

Example: Knapsack capacity  $W=16$

<u>item</u>	<u>weight</u>	<u>value</u>
1	2	\$20
2	5	\$30
3	10	\$50
4	5	\$10

# Knapsack Problem by Exhaustive Search

<u>Subset</u>	<u>Total weight</u>	<u>Total value</u>
{1}	2	\$20
{2}	5	\$30
{3}	10	\$50
{4}	5	\$10
{1,2}	7	\$50
{1,3}	12	\$70
{1,4}	7	\$30
{2,3}	15	\$80
{2,4}	10	\$40
{3,4}	15	\$60
{1,2,3}	17	not feasible
{1,2,4}	12	\$60
{1,3,4}	17	not feasible
{2,3,4}	20	not feasible
{1,2,3,4}	22	not feasible

**Efficiency:**  $\Omega(2^n)$

## Example 3: The 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.

	Job 0	Job 1	Job 2	Job 3
Person 0	9	2	7	8
Person 1	6	4	3	7
Person 2	5	8	1	8
Person 3	7	6	9	4

Algorithmic Plan: Generate all legitimate assignments (permutations), compute their costs, and select the cheapest one.

How many assignments are there?

Pose the problem as the one about a cost matrix:

# Assignment Problem by Exhaustive Search

**C =**

9	2	7	8
6	4	3	7
5	8	1	8
7	6	9	4

<u>Assignment</u> (col.#s)	<u>Total Cost</u>
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.

(For this particular instance, the optimal assignment can be found by exploiting the specific features of the number given. It is: )

# Final Comments on Exhaustive Search

- Exhaustive-search algorithms run in a realistic amount of time only on very small instances
- In some cases, there are much better alternatives!
  - shortest paths
  - minimum spanning tree
  - assignment problem
- In many cases, exhaustive search or its variation is the only known way to get exact solution