CSCE 310J: Data Structures & Algorithms

Brute Force

Dr. Steve Goddard
goddard@cse.unl.edu

http://www.cse.unl.edu/~goddard/Courses/CSCE310J

CSCE 310J: Data Structures & Algorithms

- **Giving credit where credit is due:**
  - **Most of the lecture notes are based on the slides from the Textbook's companion website**
    - **http://www.aw.com/cssuport/**
  - **I have modified them and added new slides**

Brute Force

A straightforward approach usually based on problem statement and definitions

Examples:

1. **Computing a^n ($a > 0$, n a nonnegative integer)**
2. **Computing n!**
3. **Multiply two n by n matrices**
4. **Selection sort**
5. **Sequential search**

String matching

- ***pattern*: a string of *m* characters to search for**
- ***text*: a (long) string of *n* characters to search in**
- ***Brute force algorithm:***
  1. **Align pattern at beginning of text**
  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**
  3. **while pattern is not found and the text is not yet exhausted, realign pattern one position to the right and repeat step 2.**

Brute force string matching – Examples:

1. **Pattern: 001011**
   Text: 10010101101001100101111010
2. **Pattern:** `happy`
   Text: `It is never too late to have a happy childhood.`

Number of comparisons:

Efficiency:

Brute force polynomial evaluation

- **Problem: Find the value of polynomial**
  $p(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x^1 + a_0$
  at a point $x = x_0$
- **Algorithm:**

```
p := 0.0
for i := n down to 0 do
    power := 1
    for j := 1 to i do
        power := power * x
    p := p + a[i] * power
return p
```

- **Efficiency:**

Polynomial evaluation: improvement

- **We can do better by evaluating from right to left:**
- **Algorithm:**

```
p := a[0]
power := 1
for i := 1 to n do
        power := power * x
        p := p + a[i] * power
return p
```

- **Efficiency:**

More brute force algorithm examples:

- **Closest pair**
  - **Problem: find closest among *n* points in *k*-dimensional space**
  - **Algorithm: Compute distance between each pair of points**
  - **Efficiency:**
- **Convex hull**
  - **Problem: find smallest convex polygon enclosing *n* points on the plane**
  - **Algorithm: For each pair of points p_1 and p_2 determine whether all other points lie to the same side of the straight line through p_1 and p_2**
  - **Efficiency:**

Brute force strengths and weaknesses

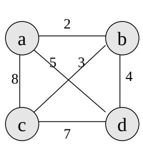
- **Strengths:**
  - **wide applicability**
  - **simplicity**
  - **yields reasonable algorithms for some important problems**
    - **searching**
    - **string matching**
    - **matrix multiplication**
  - **yields standard algorithms for simple computational tasks**
    - **sum/product of *n* numbers**
    - **finding max/min in a list**
- **Weaknesses:**
  - **rarely yields efficient algorithms**
  - **some brute force algorithms unacceptably slow**
  - **not as constructive/creative as some other design techniques**

Exhaustive search

- **A brute force solution to a problem involving search for an element with a special property, usually among combinatorial objects such a permutations, combinations, or subsets of a set.**
- **Method:**
  - **construct a way of listing all potential solutions to the problem in a systematic manner**
    - **all solutions are eventually listed**
    - **no solution is repeated**
  - **Evaluate solutions one by one, perhaps disqualifying infeasible ones and keeping track of the best one found so far**
  - **When search ends, announce the winner**

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:**



Traveling salesman by exhaustive search

Tour	Cost
a→b→c→d→a	2+3+7+5 = 17
a→b→d→c→a	2+4+7+8 = 21
a→c→b→d→a	8+3+4+5 = 20
a→c→d→b→a	8+7+4+2 = 21
a→d→b→c→a	5+4+3+8 = 20
a→d→c→b→a	5+7+3+2 = 17

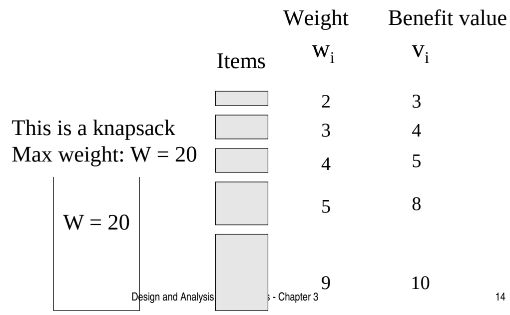
- **Efficiency:**

0-1 Knapsack problem

- Given a knapsack with maximum capacity W, and a set S consisting of n items
- Each item i has some weight w_i and benefit value v_i (all w_i, v_i and W are integer values)
- <u>Problem</u>: How to pack the knapsack to achieve maximum total value of packed items?

Design and Analysis of Algorithms - Chapter 3 13

0-1 Knapsack problem: a picture



Design and Analysis of Algorithms - Chapter 3 14

0-1 Knapsack problem

- Problem, in other words, is to find

$$\max \sum_{i \in T} v_i \text{ subject to } \sum_{i \in T} w_i \le W$$

- The problem is called a *"0-1"* problem, because each item must be entirely accepted or rejected.
- In the *"Fractional Knapsack Problem,"* we can take fractions of items.

Design and Analysis of Algorithms - Chapter 3 15

0-1 Knapsack problem: brute-force approach

Let's first solve this problem with a straightforward algorithm

- We go through all combinations and find the one with maximum value and with total weight less or equal to W

Design and Analysis of Algorithms - Chapter 3 16

Example 2: Knapsack Problem

Given n items:

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

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

<u>Example:</u>

item	*weight*	*value*
1	2	$20
2	5	$30
3	10	$50
4	5	$10

Knapsack capacity W=16

Design and Analysis of Algorithms - Chapter 3 17

Knapsack by exhaustive search

Subset	*Total weight*	*Total value*
{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:

Design and Analysis of Algorithms - Chapter 3 18

0-1 Knapsack problem: brute-force approach

- **Algorithm:**
  - **We go through all combinations and find the one with maximum value and with total weight less or equal to W**
- **Efficiency:**
  - **Since there are n items, there are 2^n possible combinations of items.**
  - **Thus, the running time will be $O(2^n)$**

Final comments:

- **Exhaustive search algorithms run in a realistic amount of time *only on very small instances***
- **In many cases there are much better alternatives!**
  - **Euler circuits**
  - **shortest paths**
  - **minimum spanning tree**
  - **assignment problem**
- **In some cases exhaustive search (or variation) is the only known solution**