Algorithm Design Techniques

Introduction

Brute Force

Algorithm Design

- When searching for a solution to an algorithmic problem, we may be interested in two types:
  - We are looking for the optimal (Exact) solution
  - We are interested in a solution which is *good enough*, where good enough is defined by a set of parameters

    (Approximately optimal solution)
- In general, it is hard to design optimal algorithms that are:
  - correct
  - efficient
  - implementable

Algorithm Design

- From now on, we are going to study some basic and general strategies in designing algorithms to solve some typical computing problems.
- We will analyze the efficiency of these algorithms using the tools we have seen in the past few lectures.
- We will learn how to design feasible or partial solutions with better efficiency:
    - A *feasible solution* is a solution which satisfies any given requirements
    - A *partial solution* is a solution which is not complete but could possibly be extended/improved.

Algorithm Design Techniques

- Brute Force
- Divide and Conquer
- Greedy Algorithms
- Dynamic Programming
- Transform and Conquer
- Backtracking
- Genetic Algorithms
- ……………………

Brute Force

“Et tu, Brute?”

“You too, Brutus?“

-Julius Caesar's last words

Brute Force Strategy for Algorithm Design

- Brute Force is a straightforward approach to solving a problem
- The strategy is directly based on the problem's statement and definitions.
- But ,all possible solutions have to be considered , this often takes too much time to run.
- In many cases, Brute Force does not provide us very efficient solutions.

Brute Force : Some Examples

Solutions are based on the problem's statement and definition :

- Exponentiation (standard algorithm)
- Calculating a polynomial (standard algorithm)
- Matrix multiplication (standard algorithm)
- Sequential search
- Sum of an array (standard algorithm)
- Max /min of an array (standard algorithm)
- Simple sorting methods
- Exhaustive search
  - TSP problem
  - Knapsack problem
  - Assignment problem
- Exhaustive search in graphs
  - BFS, DFS, …
- …

Example :The Game of Sudoku

Sudoku rules :

- We have a 9 x 9 table with 81 cells.
- We have to fill the cells such that each number must appear once in every row, column, and 3 × 3 outlined squares.
- We are given some initial numbers, and if they are chosen appropriately, there is a unique solution.

What is total number of possible fillings?

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

Brute Force Solution of Sudoku

Brute Force: Try every possible solution, and discard those which do not satisfy the conditions:

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

Runtime : There are 61 free cells to fill.

→Brute Force technique would require us to check

$9^{61} \approx 1.6\times10^{58}$ possible solutions!

This is impossible. (Remember the age of the universe!)

Brute Force Search and Sort

- Remember that , in simple search and sort algorithms the entire list have to be scanned.
- These algorithms are effectively brute force algorithms. Because,in the worst case we have to check every possibility.
- Worst case complexities of some algorithms in this category are :
  - Sequential Search O(n)
  - Selection Sort $O(n^2)$
  - Insertion Sort $O(n^2)$
  - Bubble Sort $O(n^2)$
  - …………………………

  First,we consider two simple problems:1.Exponentiation,2.Evaluation of polynomials.

Brute Force Exponentiation : Iterative solution

Compute nth power of positive a : $a^n = a \times a \times \ldots\ldots \times a$

```
// The computation requires one loop for multiplications
res = 1
for i=1 to n
        res=res*a
return res
  Or
res = 1
for i=n  to 1
        res=res*a
return res
```

Iterative Exponentiation : Analysis

The number of multiplications for both solutions:

$$T(n) = \sum_{i=1}^{n} 1 = n$$

$= \Theta(n)$

Recursive Exponentiation

```
  //Computes a^n recursively, n is a non-negative integer
exp(a, n)
      if n == 0
       return 1
   return a * exp (a, n- 1)
```

Complexity : We have to perform n multiplications :

$T(n)= \Theta(n)$

Brute force polynomial evaluation

Problem:

Find the value of a polynomial at a point $x = x_0$

$p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x^1 + a_0$

Example :

$p(x) = 4x^4 + 7x^3 - 2x^2 + 3x^1 + 6$

Brute force solution: Perform all multiplications in every term:

$p(x) = 4*x*x*x*x + 7*x*x*x - 2*x*x + 3*x + 6$

Brute force polynomial evaluation

//The a terms in p(x) are passed in parameter P.

Poly1(P , x_0)

```
x=x0
p= 0.0
for i = n down to 0 do
   power = 1
   for  j = 1 to i do
        power =  power * x     //compute x^i :  x^n , x^(n-1)   …
   p = p + a[i] * power
return p
```

Efficiency : Two nested for loops :→ $T(n)= \Theta(n^2)$

Can ve do better? Yes.

Polynomial evaluation: Improvement

We can do better by evaluating from right to left:

→Use x^{i-1} to compute x^i i.e. only multiply x^{i-1} by x

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x^1 + a_0$$

Poly2(P , x_0)

```
x = x0
p = a[0]
power = 1
for i =  1 to n do
   power =  power * x
   p = p + a[i] * power
return p
```

Efficiency : One for loop→ $T(n)= \Theta(n)$

→One order of magnitude improvement!

Brute Force : Selection Sort

- We have discussed Insertion Sort, now we consider Selection Sort as an example of brute force:
  - Scan the entire given list to find its smallest element and swap it with the first unsorted element.
  - Repeat for every next element.
  - This is a straightforward solution : Brute Force strategy

  Time efficiency : $\Theta(n^2)$

  Why ?

  Each time we have to find the minimum by brute force.

  Buble sort is also a Brute Force algorithm.

Brute Force: Selection Sort

```
SelectionSort(A,n)
  for i = 0 to n-2 do
      min ← i
      for j = i+1 to n-1 do
         if A[j] < A[min]
              min ← j
      swap (A[i] , A[min])
```

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

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

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

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

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

17 29 34 45 68 89 | 90

Selection Sort: Analysis

How many times the second loop is executed?

comparisons :

T1(n) = $\sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1$

= $\sum_{i=0}^{n-2} [(n-1) - (i+1) + 1]$

= $\frac{(n-1)n}{2}$

→T1(n) = $\Theta(n^2)$

Selection Sort : Analysis

of key swaps:

T2(n)=$\sum_{i=0}^{n-2} 1 = n - 1$

T2(n)= Θ(n)

#Assignments (Consider data assignments in swaps):

$$a(n) = \sum_{i=2}^{n} 3 = 3(n-1)$$ (1 swap = 3 assignments)

T3(n) =Θ(n)

So that the overall complexity is

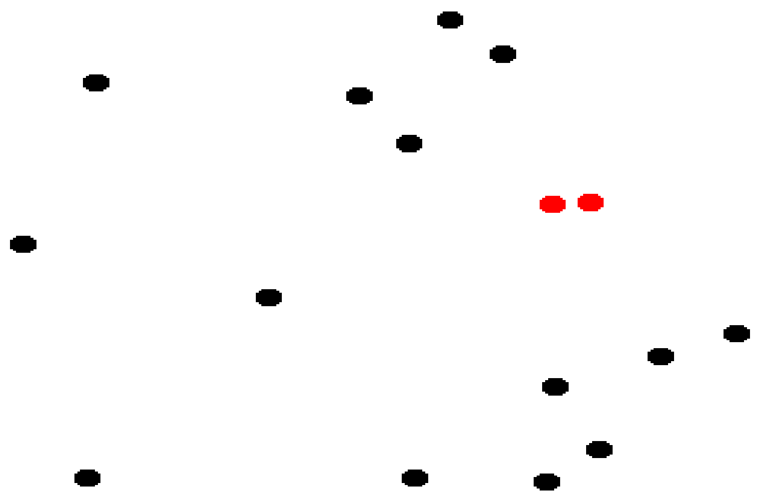
T(n)= T1(n)+T2(n)+ T3(n)

=$\Theta(n^2)$

Brute Force Closest-Pair of Points

- Find the two closest points in a set of n points on a plane.
- Points can be airplanes (most probable collision candidates), database records, DNA sequences,…

Example:



Closest-Pair by Brute-force

- For simplicity we consider 2-D case
- Euclidean distance :

$$d(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

- Brute-force: compute distance between each pair of disjoint points and find a pair with the smallest distance.Since

$$d(p_i, p_j) = d(p_j, p_i),$$

we consider only $d(p_i, p_j)$ for $i < j$

Closest-Pair by Brute-force

BruteForceClosestPair(P)

//Input: A list P of n (n≥2) points $p_1(x_1,y_1)$, …, $p_n(x_n,y_n)$

//Output: distance between closest pair of points

d ← ∞ //Initially, later it is minimized

for i ← 1 **to** n-1 **do**

for j ← i+1 **to** n **do**

d ← min(d, sqrt($(x_i - x_j)^2 + (y_i - y_j)^2$))

return d

Closest-Pair : Analysis

Input size: n

- Basic operation: Computing square root → Costly
- Actually , computing the square root is not needed . The result will be the same if we consider only the squares:
- We will find the same pair of points in both cases. So,the basic operation can be taken as squaring two numbers:

  How many times?

  T(n) = $\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2 = 2 \sum_{i=1}^{n-1} (n-i)$

  = 2[(n-1)+(n-2)+…+1]

  =$2\sum_{i=1}^{n-1} i$ = 2 [n(n+1)/2 –n]

  = (n-1)n

  = $\Theta(n^2)$

Exhaustive Search

State-space search:

Given an initial state, a goal state and

a set of operations,

Find a sequence of operations that transforms the

initial state to the goal state.

The solution process can be represented as a tree

Exhaustive Search

- A brute-force approach to combinatorial problems:
  - Generate each and every element of the problem's domain
  - Then compare and select the desirable element that satisfies the constraints
  - Use combinatorial objects such as permutations, combinations, and subsets of a given set.
  - Find the solution that optimizes some objective function
  - The time efficiency is usually bad – the complexity grows exponentially with the input size.

Exhaustive Search

- Examples: Brute force solution of
  - Traveling salesman problem
  - Knapsack problem
  - Assignment problem
  - Cripotograpy

We are going to consider some introductory examples.

(Better solutions will be considered later)

Exhaustive Search:
Traveling Salesperson Problem (TSP)

- Find the shortest tour through a given set of n cities that visits each city exactly once before returning to the city where it started.
- Can be conveniently modeled by a weighted graph; vertices are cities and edge weights are distances
- Same as finding a “Hamiltonian Circuit” in a graph:

→A circuit is a path with no repeating edges that begins and ends at the same vertex.

Hamiltonian circuits and TSP

- Hamiltonian path: A path that uses each vertex of a graph exactly once.
- Hamiltonian Circuit: If the path ends at the starting vertex, it is called a **Hamiltonian circuit.**

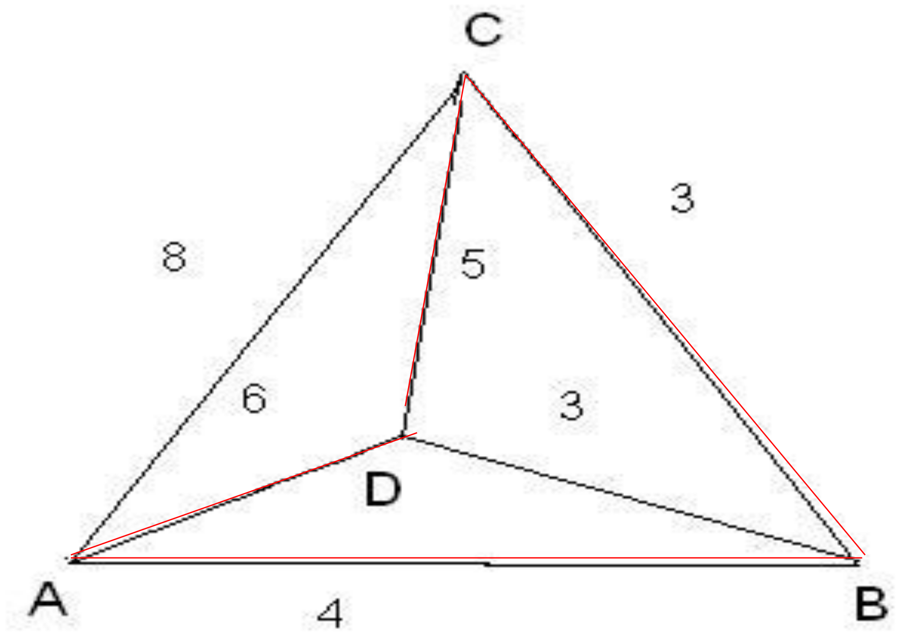
→A sequence of n+1 adjacent vertices :

$$v_{i_0}, v_{i_1}, \ldots, v_{i_{n-1}}, v_{i_0}.$$

Hamiltonian Circuit : Example

You plan a vacation and wish to visit spots A,B,C,D. How to minimize total distance driven?

Brute force solution : list all circuits
compute distances
choose tour of minimum total distance



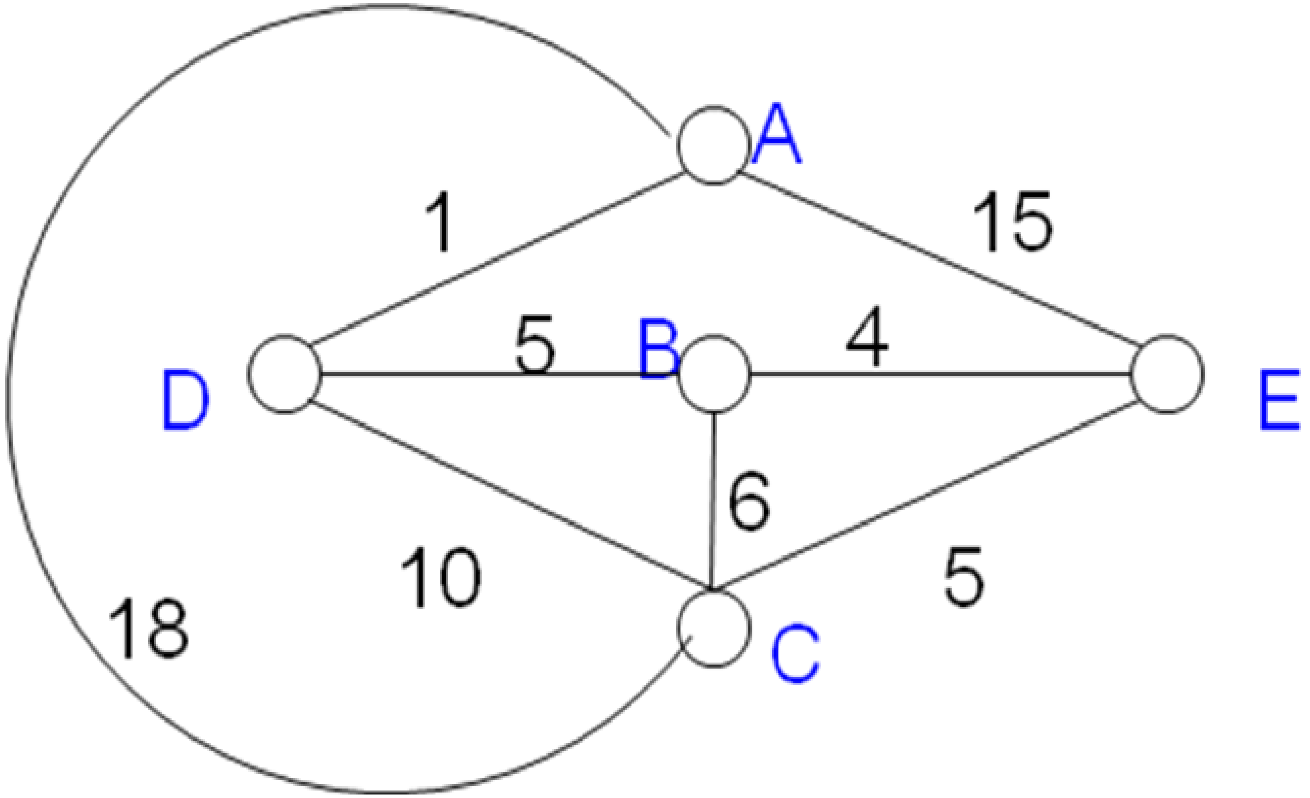
start at A:	Mileage
ABCDA	18
ABDCA	20
ACBDA	20
………	…...

Traveling Salesperson Problem

- A salesperson has a list of n cities, each of which (s)he must visit *exactly once* before returning to the initial city.
- There are direct roads between some pairs of cities as shown on a map.
- Find the route the salesperson should follow for the shortest possible round trip that both starts and finishes at the same given home city of the salesperson.

TSP : Illustration

Find the shortest TSP tour starting at A :



Hamiltonian Circuits and TSP

Given a directed graph G = (V , E):

city → vertex, road → edge, length of the road → edge weight.

TSP → Find a shortest Hamiltonian Circuit : A cycle that passes through all the vertices of the graph exactly once.

- Exhaustive search by Brute Force:
  - List all the possible Hamiltonian circuits (starting from any vertex)
  - Ignore the direction
  - How many candidate circuits do we have?

TSP :Complexity

paths =(n-1)! possible paths for a directed graph.

= (n-1)!/2   "   "   undirected

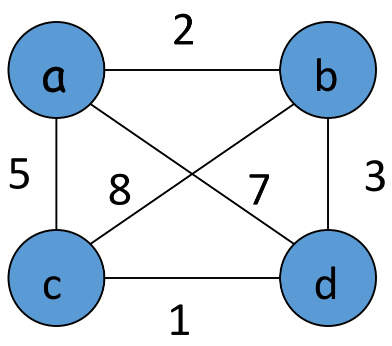
= n! possible paths if home city is not fixed

In all cases we have a $\Theta(n!)$ problem !

Intractable! Why ?

(Example : 20! > 10^{18})

TSP Solution by Exhaustive Search : Example



All possible tours starting at a	Total distance	
a → b → c → d → a	2+8+1+7 = 18	
a → b → d → c → a	2+3+1+5 = 11	⟵ **optimal**
a → c → b → d → a	5+8+3+7 = 23	
a → c → d → b → a	5+1+3+2 = 11	⟵ **optimal**
a → d → b → c → a	7+3+8+5 = 23	
a → d → c → b → a	7+1+8+2 = 18	

The Knapsack Problem

We are given n items and a knapsack:

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

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

→What is the maximum value that we can put into the knapsack ?

Example: Knapsack capacity W=16

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

Knapsack: Solution 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: how many subsets?
The total number of subsets for n is 2^n.
So,we have $\Theta(2^n)$ complexity !

Brute Force Searching in a Graph

(Review graph terminology and basic algorithms)

- Breadth-first search:
  - go level by level in the graph,uses a queue
- Depth-first search:
  - go as deep as you can then backtrack,uses a stack
- Both take $\Theta(V+E)$ time, where $|V|$ is the number of vertices and $|E|$ is the number of edges

We are going to consider graph related algorithms later.

Brute Force : Criptography

- A brute-force attack consists of an attacker trying all possible passwords with the hope of eventually guessing the password correctly.
- The attacker systematically checks all possible passwords and passphrases until the correct one is found.
- As the password's length increases, the amount of time to find the correct password increases exponentially.
- The resources required for a brute-force attack grow exponentially with increasing key size : $2^{\text{key size}}$

  key size: # of bits in the key.
- Modern symmetric algorithms typically use computationally stronger 128- to 256-bit keys which are (almost)impossible to crack.

Brute Force Cryptograpy :Example

- Lets say we have an alphanumeric 8-character password.

  → We can have 52 possible letters (English Alphabet)
- If we add the Numeric digits,we have 62 characters in total.
- Brute force will check all possible passwords:
  - For 8-character-password, the number of possible passwords is :
    62^8= 218.340.105.584.896
- Assume our system can check 1 result per second.

  → Checking all possibilities would take 218 trillion seconds

  → ~7 million years would be needed to crack the password.

Warning: Current day hardware and software for fast computing can reduce this time to just a few seconds!

Brute-Force Strengths

- Simplicity and wide applicability
- Yields reasonable algorithms for some important problems such as :
  matrix multiplication, sorting, searching, string matching…
- Also yields standard algorithms for simple computational tasks and graph traversal problems
- Brute-force techniques are inefficient, but we may use them to evaluate solutions found through other algorithms.
- On the other hand, Brute Force may be feasible for moderate size problems with the increased powers of current computers.

Brute-Force Weaknesses

- Rarely yields efficient algorithms
- Some brute-force algorithms are unacceptably slow
- Not as constructive and elegant as some other design techniques.
- In general ,Brute Force is the most naive way to search for solution to a computational problem!

  [In fact , to any real life problem ☺].