Computational Complexity

Cogan Shimizu
KASTLE Lab
Wright State University, Dayton, OH
http://www.coganshimizu.com

Outline

1. A motivating example
2. What is *Computational Complexity* all about?
3. More examples

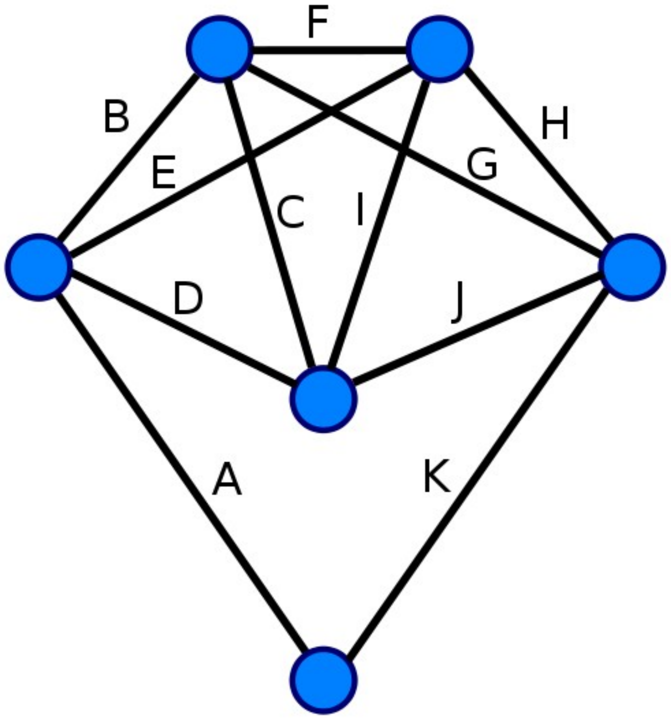
A motivating example

Input: A connected graph (undirected)

Output: “yes” if the graph has a path which

- **visits each edge exactly once and**
- **starts and ends on the same vertex.**

Output: “no” otherwise



Find an algorithm for this problem.

[Such graphs are called *Eulerian*.]

[Such paths are called *Eulerian cycles*.]

Naive algorithm 1 – brute force

- Graph consists of
  - m vertices: 1,2,...,m
  - n edges, written as (x,y) [edge between vertex x and vertex y]

```
for each permutation P of the n edges
        [i.e., P = ( (x1,y1),...,(xn,yn) )]
  output "yes" if P constitutes an Eulerian cycle
output "no" if no Eulerian cycle was found
```

Is this a good algorithm?

How to improve?

Naive algorithm 1 – brute force

- Graph consists of
  - m vertices: 1,2,...,m
  - n edges, written as (x,y) [edge between vertex x and vertex y]

```
for each permutation P of the n edges
        [i.e., P = ( (x_1,y_1),...,(x_n,y_n) )]
  output "yes" if P constitutes an Eulerian cycle
output "no" if no Eulerian cycle was found
```

How costly is this? (roughly, order of magnitude)

If no Eulerian cycle is found, we have to check n! permutations (that's the *worst case*).

n! is quite a lot!

n	n!
5	120
10	3,628,800
15	¼ 1.3 ¢ 10^{12}
20	¼ 2.4 ¢ 10^{18}
50	¼ 3 ¢ 10^{64}
70	¼ 10^{100}

10^{100} – That’s more than there are particles in the universe

n! is quite a lot!

- 10^1
- 10^2
- 10^3 Number of students in the college of engineering
- 10^4 Number of students enrolled at WSU
- 10^6 Number of people in Dayton Metro Area
- 10^7 Number of people in Ohio
  Number of seconds in a year
- 10^8 Number of people in the Midwest
- 10^10 Number of stars in the galaxy
  Number of people on earth
  Number of milliseconds per year
- 10^20 Number of stars in the universe
- 10^80 Number of particles in the universe

Naive algorithm 2 - backtracking

- Graph consists of
  - m vertices: 1,2,...,m
  - n edges, written as (x,y) [edge between vertex x and vertex y]

```
Fix the first vertex, say, x1.
Make a systematic depth-first search on the graph
              edges.
  For each resulting maximal path P, if P is an
              Eulerian cycle, output “yes”.
If no Eulerian cycle is found, output “no”.
```

Algorithm is better – but is it *significantly* better?

In the worst case, fully connected graph, we have to check m! paths.

When do we know we have *the best* algorithm?

Smart algorithm

Theorem:

A connected undirected graph is Eulerian if and only if every vertex has an even number of edges (counting loops twice).

```
For each vertex x
   if number of edges of x is odd, output “No” and
                                  stop.
Output “Yes”.
```

In the worst case, we have to make m checks, each of which consists of counting at most n edges.

Outline

1. A motivating example
2. What is *Computational Complexity* all about?
3. More examples

What is Comp. Complexity all about?

- Some problems seem hard but are not. Identify them.
- Some problems seem easy but are not. Identify them.

- Know when to stop searching for a smarter algorithm. [And instead turn to optimizations and heuristics.]

- What does “computationally hard problem” mean exactly?
- In what sense can we really say that some problem is computationally harder than some other problem?

What is Comp. Complexity all about?

- It's a part of *theoretical* computer science.
- It's a formal theory of the analysis of computational hardness of problems.
- It's probably rarely going to help you directly in practice.
- But indirectly, in form of having a systematic understanding of problem hardness, it is indispensable.

What is Comp. Complexity all about?

We will certainly also learn about the

P = NP?

problem.

What it is.
Why it is important.

Why some people make such a fuzz about it.

Some basic notions

- **Problem:**
  A mapping from input to output.

- **Algorithm:**
  A method or a process followed to solve a problem.

- **A problem can have many algorithms.**

Some basic notions

- **Problem:**
  A mapping from input to output.

- **Algorithm:**
  A method or a process followed to solve a problem.

- **We focus on problems. Algorithm analysis is also interesting, but not as foundationally important.**

Some basic notions

- **Problem:**
  A mapping from input to output.
  - **We use the *order of magnitude* of the number of steps needed to solve a problem.**
    - **measured as a number which depends on the *input size*.**
  - **We are really interested in the *worst case* scenario.**
    - **i.e., how many steps do we need if the input is as unfavorable as possible?**

Can also be studied:

best case (usually not that interesting)

average case (of practical interest for concrete algorithms)

Outline

1. A motivating example
2. What is *Computational Complexity* all about?
3. More examples

Linear Search

- **Input: An array *A* of integers, and a value *v*.**
- **Output: "Yes" if *v* is an element of *A*; "No" otherwise.**

A =

3	12	7	25	7	32	11	56	28	43	6	87	68	91	2

$v = 28$

Algorithms: *Exhaustive search; random search; sort and linear search; sort and binary search*

Linear Search

Not all inputs of a given size take the same time to run.

Sequential search for *v* in an array of *n* integers:

- **Begin at first element in array and look at each element in turn until *v* is found**

Best case:

Worst case:

Average case?

Linear Search – Average case

Case: i	Time: T(i)	Probability P(i)	Cost: T(i) * P(i)
1	1	1/n	1/n
2	2	1/n	2/n
3	3	1/n	3/n
…	…	…	…
n	n	1/n	1

$$\sum Cost = \tfrac{1}{n} + \tfrac{2}{n} + \cdots + \tfrac{n}{n}$$

$$= \frac{1}{n} \times \sum_{i=1}^{n} i$$

$$= \frac{1}{n} \times \frac{n(n+1)}{2} = \frac{n+1}{2}$$

Bubble Sort

- **Algorithm 1:**

```
for pass = 0…n-1 {
  for position = 0…n-1 {
    if (array[position] > array[position+1]) {
      swap (array[position], array[position+1])
    }
  }
}
```

Best, worst, average: $\Theta(n^2)$

Bubble Sort

- **Algorithm 2:**

```
for pass = 0…n-1 {
  for position = 0…n-pass-1 {
    if (array[position] > array[position+1]) {
      swap (array[position], array[position+1])
    }
  }
}
```

Best, worst, average: $\Theta(n^2)$

(Within a constant factor)

Bubble Sort

- **Algorithm 3:**

```
for pass = 0…n-1 {
  swaps = 0;
  for position = 0…n-pass-1 {
    if (array[position] > array[position+1]) {
      swap (array[position], array[position+1]);
      swaps++;
    }
  if (swaps == 0) return;
  }
}
```

Best: ㄹㅁ n,

Worst: ㄹㅁ (n^2)

Average = ??