

Lecture 1

Introduction to Algorithms

Md. Asif Bin Khaled

What's An Algorithm?



Algorithms


Informally, an **algorithm** is any **well-defined computational** procedure that takes some value, or set of values, as **input** and produces some value, or set of values, as **output**. An algorithm is thus a sequence of computational steps that transform the input into output.

We can also view an algorithm as a tool for solving a well-specified **computational problem**. In theoretical computer science, a **computational problem** is collection of questions that computers might be able to solve. For example, the problem of factoring.

Algorithms Continued.

— — —

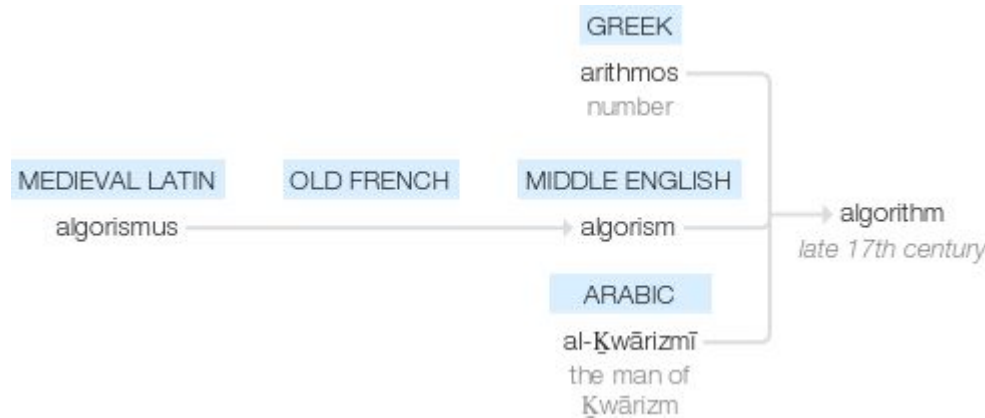
So an Algorithm is

1. Finite
2. Well-Defined 
3. Computer-Implementable

Instructions

Etymology of Algorithm

The actual term '**algorithm**' is often cited as originated from the 9th Century Persian mathematician **Abu Abdullah Muhammad ibn Musa Al-Khwarizmi**. He's also known as "**the father of Algebra**".



Analysis of Algorithm

— — —

A theoretical study of computer-program **performance** and **resource** usage.

Formally,

In computer science, the analysis of algorithms is the process of **finding the computational complexity of algorithms** – the amount of **time, storage, or other resources** needed to execute them.

We want to make our programs **fast**.

Why Analyze an Algorithm?

1. **Classify problems and algorithms by difficulty.** We want to classify, not just algorithms but also problems according to their level of difficulty.
2. **Predict performance, compare algorithms, tune parameters.** We want to be able to predict performance and compare algorithms. We want to know how much time our algorithms are going to take. And we want to know which algorithm is better than which other one for our particular tasks.
3. **Better Understand and improve implementations and algorithms.**

What's more Important than Performance while Engineering?

— — —

1. Modularity
2. Correctness
3. Maintainability
4. Functionality
5. Robustness
6. User-Friendliness
7. Simplicity
8. Extensibility
9. Scalability
10. Reliability

Why Study Algorithm and Performance?

— — —

1. Algorithms help us to understand scalability.
2. Algorithmic mathematics provides a language for talking about program behavior.
3. Performance often draws the line between what is feasible and what is not.
4. Performance is the currency of computing.

What Makes an Algorithm Good

A good Algorithm has mainly two prime factors,

1. **Correctness**

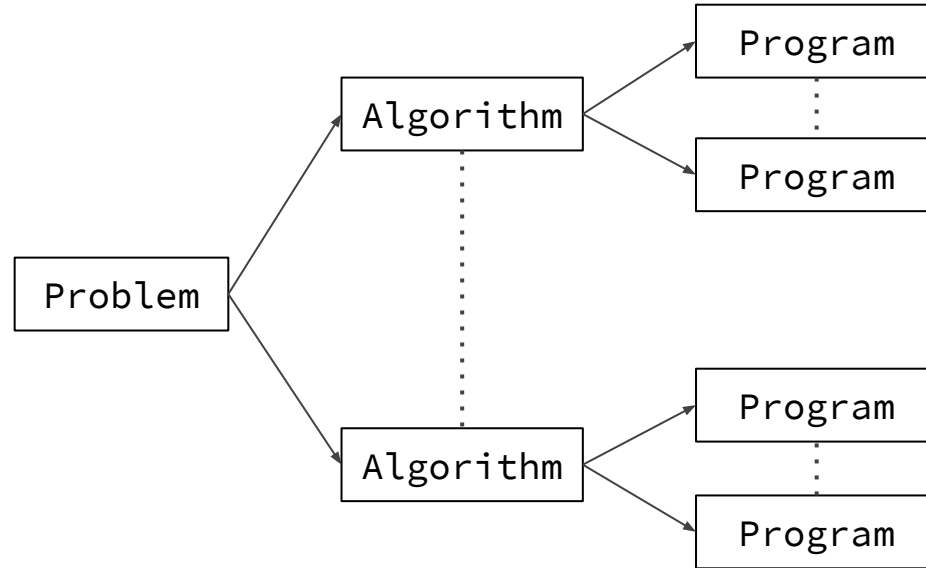
An Algorithm is said to be correct if for every input instance it halts with the correct output.

2. **Efficiency**

In computer science, algorithmic efficiency is a property of an algorithm which relates to the number of computational resources used by the algorithm. It can be mainly classified into,

- a. Space Efficiency** - the memory required, also called, space complexity
- b. Time Efficiency** - the time required, also called time complexity

Problem, Algorithms and Programs



Choosing Comparatively Bad Algorithm

There are differences even between correct algorithms. We can have many correct algorithms for the same problem, and they can be different in terms of efficiency. In most of the cases, we usually choose the most efficient algorithm. However, sometimes we are willing to sacrifice a little bit of efficiency for ease of implementation.

For example, Say we have two algorithms **A** and **B** for the same problem. **A** is more efficient than **B**, but **B** is much easier to implement. Then if we have resources and if **B** does not take too much away from us then we can choose **B** to solve our problem.

Pseudocode

Pseudocode is an **informal** way of programming description that does not require any strict programming language syntax or underlying technology considerations. It has no syntax like any of the programming language and thus can't be compiled or interpreted by the computer. **English like syntax that resembles a programming language.**

A programmer who needs to implement a specific algorithm, especially an **unfamiliar** one, will often start with a pseudocode description, and then convert that description into the target programming language and modify it to interact correctly with the rest of the program.

Pseudocode (Bubble Sort) Continued.

Bubble_Sort(A) :

```
for i=1 to A.length-1:
```

```
for j=A.length downto i+1:
```

```
if A[j]<A[j-1]:
```

Swap $A[j]$ with $A[j-1]$

The Robot Tour Optimization Problem

Suppose, we are given a robot arm equipped with a tool, say soldering iron. In a manufacturing circuit board all the chips must be soldered to their designated contact points in the substrate. To program for the robot arm for this job, we must first construct an ordering of the contact points so that robot can visit all the contact points. The robot arm will then proceed back to the first contact point to prepare for the next board, thus turning the tool-path into a closed tour, or cycle.

Time is money, so we want to minimize the tour as much as possible.

The Robot Tour Optimization Problem Continued.

— — —

Problem Definition

Problem: Robot Tour Optimization

Input: A set **S** of n points in the plane.

Output: The shortest cycle tour that visits each point in the set **S**.

The Robot Tour Optimization Problem Continued.

Nearest_Neighbor(P):

Pick and visit an initial point p_0 from P

$p = p_0$

$i = 0$

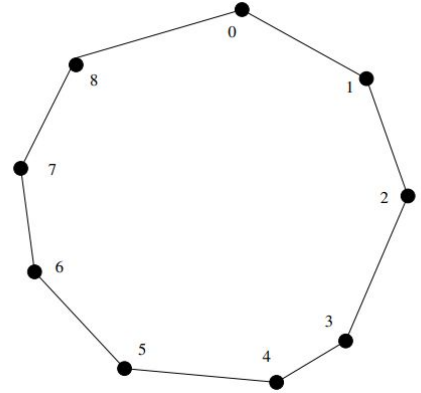
While there are still unvisited points

$i = i + 1$

Select p_i to be the closest unvisited point to p_{i-1}

Visit P_i

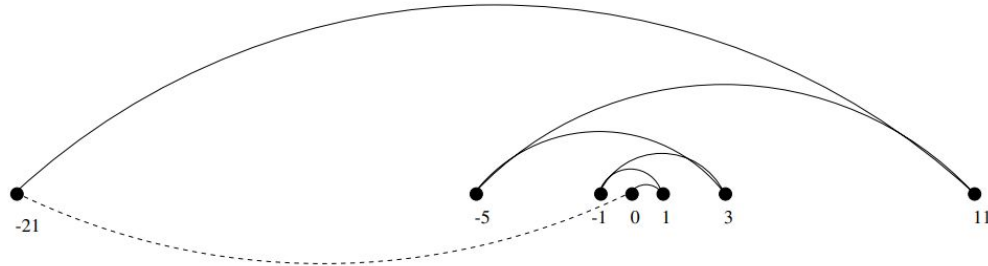
Return to p_0 from p_{n-1}



The Robot Tour Optimization Problem Continued.

It is normal to think of the nearest neighbor algorithm because it makes sense to visit the nearby points before we visit faraway points to reduce the total travel time. The algorithm works fine in the figure mentioned earlier.

However, in terms of our current problem the algorithm is completely **wrong**. The algorithm always finds a tour, but it doesn't ensure ensure the shortest tour.



The Robot Tour Optimization Problem Continued.

A much better tour for these points starts from the leftmost point and visits each point as we walk right before returning at the rightmost point. This way really gives us the optimal solution.

Now what if we rotate the points by 90 degree.

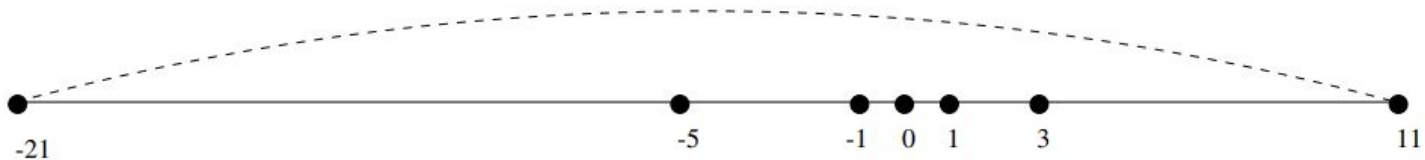
Still correct? Now all the points are equally leftmost. Maybe now we pick another approach to solve the problem.



The Robot Tour Optimization Problem

A different idea might be to repeatedly connect the closest pair of endpoints whose connection will not create a problem, such as premature termination of the cycle. Connecting the final two endpoints gives us a cycle.

This approach is somewhat more complicated and less efficient than the previous one however, but at least it gives the correct solution in this case.



The Robot Tour Optimization Problem Continued.

— — —

Closest_Pair(P):

Let n be the number of points in set P .

for $i=1$ to $n-1$:

$d=\infty$

 for each pair of endpoints (s,t) from **distinct** vertex chains

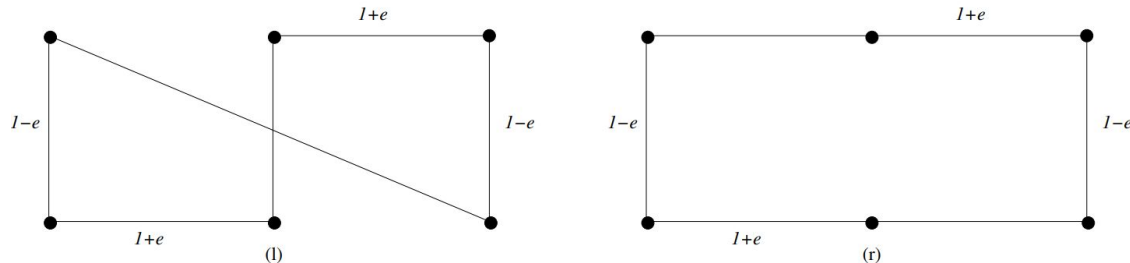
 if $\text{dist}(s,t) \leq d$ then $s_m=s$, $t_m=t$, and $d=\text{dist}(s,t)$

 Connect (s_m, t_m) by an edge

Connect the two endpoints by an edge

The Robot Tour Optimization Problem Continued.

However, the **ClosestPair** algorithm is not correct for all scenarios. In the left(l) figure the closest pair of points stretch across the gap not around the boundary. After we pair off these points $(1-e)$, the closest remaining pairs will connect these pairs alternately around the boundary. The total path of the closest pair tour is $3(1-e)+2(1+e)+\sqrt{((1-e)^2+(2(1+e))^2)}$ and this tour path is 20% farther than the path in the right(r) figure.



The Robot Tour Optimization Problem Continued.

Another approach to solve this problem would be by enumerating all possible ordering of the set of points, and then select the ordering that minimizes the total length. However, since all ordering are considered, we are guaranteed to end up with the shortest possible tour. This algorithm is correct however it is extremely slow. If we have 1000 points then we would get 1000! Which even the fastest computer would come close to finish it.

The quest for an efficient algorithm to solve the problem, called the **traveling salesman problem (TSP)**. We will come to that later on this semester.

The Robot Tour Optimization Problem Continued.

— — —

OptimalTSP(P):

$d = \infty$

for each of the $n!$ Permutations P_i of point set P

 if ($\text{cost}(P_i) \leq d$) then $d = \text{cost}(P_i)$ and $P_{\min} = P_i$

return P_{\min}

Applications of Algorithm

1. The compression algorithm helps us to transfer data faster and which is fundamental for services like video conferencing, HDTV, voice mails, etc.
2. Route or Path finding algorithms help us to find shortest destinations and this is required for services like maps, router, etc.
3. Rendering algorithms help to color 3D models based on the lighting arrangements in a virtual room.
4. Optimization and Scheduling algorithm guides the satellites on how and when to arrange or rearrange their positions in the space.

Applications of Algorithm Continued

— — —

5. Exact pattern matching algorithms helps in services like search engines, spam filters, several areas of natural language processing, etc.
6. Algorithms like Scale Invariant Feature Transform (SIFT) and Speed Up Robust Features (SURF) help in facial recognition.
7. Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on biologically inspired operators such as mutation, crossover and selection.

Applications of Algorithm Continued

— — —

8. LCS (Longest Common Subsequence) algorithm can help us understand how similar a DNA sequence is to another DNA sequence.
9. Algorithms also helps us to take pictures of black holes.
10. Algorithms are also used to find new protein sequence to for creating new vaccines.
11. Algorithms also paves the way for recommendation systems used by netflix, ecommerce sites etc.

Reference

— — —

1. Wikipedia
2. MIT Course Number: 6.046J / 18.410J
3. **Introduction to Algorithms** by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein.
4. Analysis of Algorithms by Princeton University, Coursera