Lecture – 01

Introduction

Today’s Agenda

- Why CSE221 may change your life?
- Various administrative issues.
- What is algorithm?
- What is this course about?

The Nature of This Course

- This is one of the most important courses of computer science
  - It plays a central role in both the science and the practice of computing
  - It tells you how to design a program to solve important problems efficiently, effectively and professionally
  - The knowledge in this course differentiates a ‘real’ computer-science student from other students

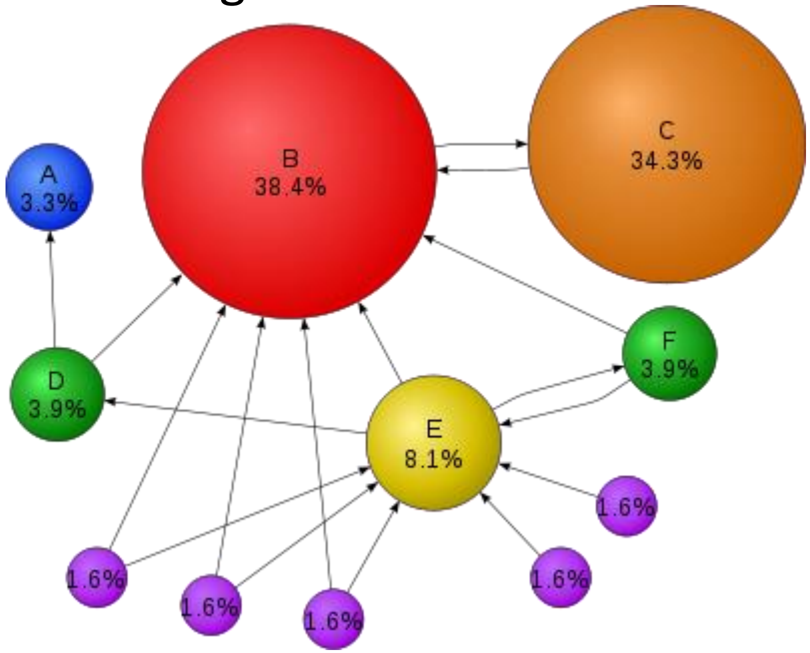
Welcome to ICT2107

- What you can expect from me:
  - Helpful, encouraging; inspiring and enjoying class
  - Good grades if u really work hard.
- What I expect from you:
  - Turn in all homework and participate classes
  - You learn some critical techniques from this course
  - You show signs to be able to invent new algorithms

Algorithms may change your life, don’t think so?

Why you want to study Algorithms?

- Making a lot of money out of a great algorithm…
- $1,000,000,000?
- Example: PageRank algorithm by Larry Page—The soul of Google search engine

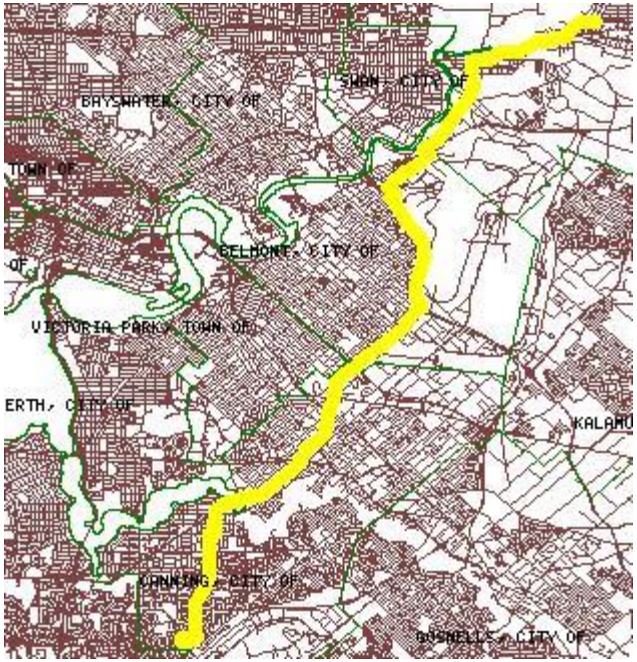


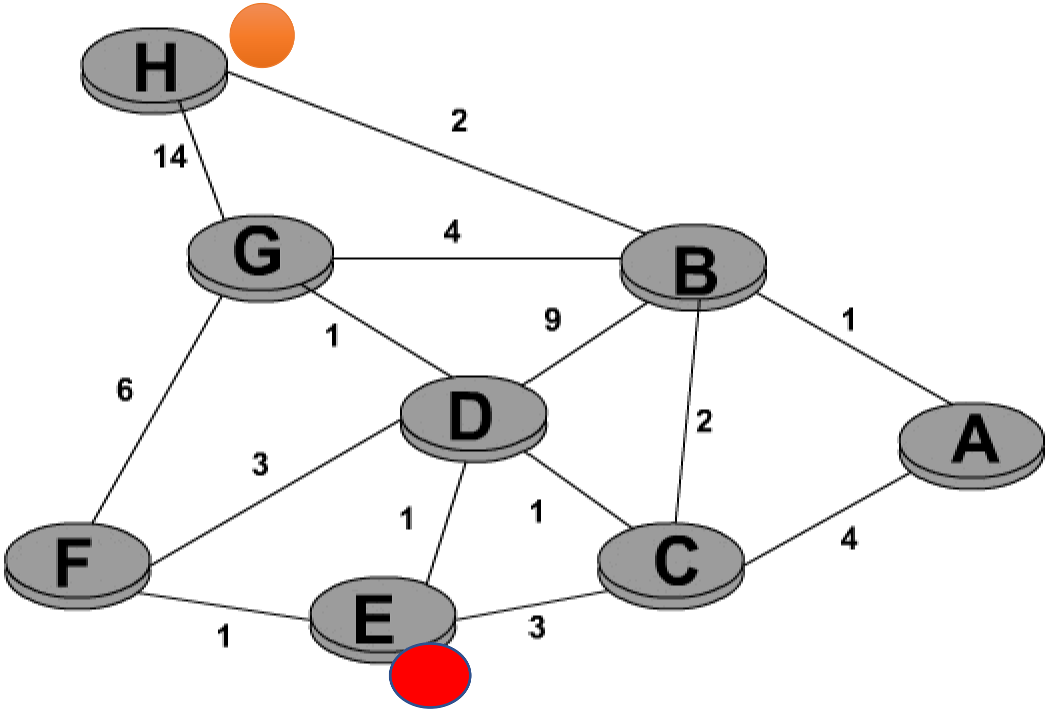
$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)}$$

Google total assets: $31 billions on 2008

Why you want to study Algorithms?

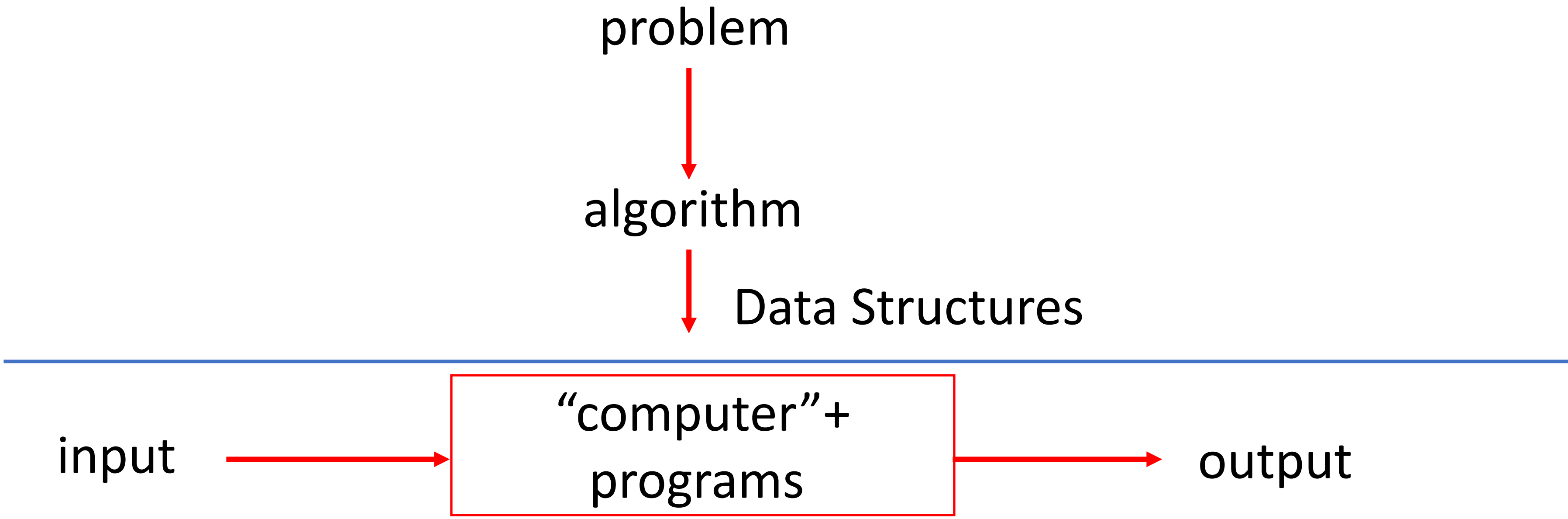
- Simply to be cool to invent something in computer science
- Example: Shortest Path Problem and Algorithm
- Used in GPS and Map quest or Google Maps





Algorithm

An *algorithm* is a sequence of unambiguous instructions/operations for solving a problem, i.e., for obtaining a required output for any legitimate input in a finite amount of time.



problem

algorithm

Data Structures

input

"computer"+ programs

output

Fundamentals of Algorithmic Problem Solving

1.Understanding the problem

2.Learning the capabilities of a computational device

3.Choose between exact and approximate problem solving

4.Deciding on appropriate data structure

5.Algorithm design techniques

6.Methods of specifying an algorithm

Pseudocode (for, if, while, //, ←, indentation…)

7.Prove an algorithm’s correctness – mathematic induction

8.Analyzing an algorithm – Simplicity, efficiency, optimality

9.Coding an algorithm

Example: Sorting

- Statement of problem:
  - ***Input:*** A sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$
  - ***Output:*** A reordering of the input sequence $\langle a'_1, a'_2, \ldots, a'_n \rangle$
  - so that $a'_i \leq a'_j$ whenever $i < j$
- Instance: The sequence $\langle 5, 3, 2, 8, 3 \rangle$
- Algorithms:
  - Selection sort
  - Insertion sort
  - Merge sort
  - (many others)

Some Important Points

- Each step of an algorithm is clear-cut
- The range of inputs has to be specified carefully
- The same algorithm can be represented in different ways
- The same problem may be solved by different algorithms
- Different algorithms may take different time to solve the same problem – we may prefer one to the other

In general

- A good algorithm is a result of repeated effort and rework
  - Better data structure
  - Better algorithm design
  - Better time or space efficiency
  - Easy to implement
  - Optimal algorithm

Some Well-known Computational Problems

- Sorting
- Searching
- Shortest paths in a graph
- Minimum spanning tree
- Primality testing
- Traveling salesman problem
- Knapsack problem
- Chess
- Towers of Hanoi

This Course is Focused on

- How to design algorithms
- How to express algorithms -- pseudocode
- Proving correctness
- Efficiency Analysis
    - Theoretical analysis
    - Empirical analysis
- Optimality

Algorithm Design Strategies

- Brute force
- Divide and conquer
- Decrease and conquer
- Transform and conquer
- Greedy approach
- Dynamic programming
- Backtracking and branch and bound
- Space and time tradeoffs

Invented or applied by many genius in CS

Analysis of Algorithms

- How good is the algorithm?
  - Correctness
  - Time efficiency
  - Space efficiency
- Does there exist a better algorithm?
  - Optimality

In general: What is an Algorithm?

- Recipe, process, method, technique, procedure, routine,… with following requirements:
- Finiteness: terminates after a finite number of steps
- Definiteness: carefully and clearly specified
- Input: valid inputs are clearly specified
- Output: can be proved to produce the correct output given a valid input
- Effectiveness: steps are sufficiently simple and basic

Thank You!!!
www.thebodytransformation.com