VE281

Data Structures and Algorithms

Introduction and Asymptotic Analysis

VE281: Data Structures and Algorithms

- Time: Tuesday 10:00-11:40 am, Thursday 10:00-11:40 am.
- Location: Dong Xia Yuan 200
- Textbooks for your Reference:
  - "Data Structures and Algorithm Analysis," by Clifford Shaffer.
    Online available:
    http://people.cs.vt.edu/~shaffer/Book/C++3e20120605.pdf
  - "Data Structures and Algorithms with Object-Oriented Design Patterns in C++," by Bruno Preiss. Online available:
    http://www.brpreiss.com/books/opus4/html/book.htm

Instructor

- Weikang Qian
- Email: qianwk@sjtu.edu.cn
- Phone: 3420-4020
- Office: Room 119, JI Building
- Office hour
  - Tuesday 1:00 - 2:00 pm
  - Thursday 4:00 - 5:00 pm
  - Or *by appointment*

Teaching Assistants

- To be determined

Some Administrative Details

Grades

- Composition
  - Assignments: 50%
    - Include written assignments and programming assignments
  - Midterm exam (closed book, written exam): 20%
  - Final exam (closed book, written exam): 30%
- No late in submitting your assignments!
- We will assign grades on a curve, in keeping with past grades given in this course.
- Questions about the grading?
  - Must be mentioned to TAs or instructor within **one week** after receiving the item.

Some Administrative Details

Programming Assignments Details

- Programming Assignments require:
  - Read and understand a problem specification
  - Design a solution to this problem
  - Implement this solution simply and elegantly
  - Convince yourself that your solution is correct
- Grading programming assignments will be done by a combination of testing (correctness) and reading (correctness and simplicity/elegance).
- We will give you a few simple test cases to get started, but we will not tell you everything we will be testing for. It is up

Some Administrative Details

Collaboration and Cheating

- You must do all the assignments yourself.
- You may not discuss the specifics of your solution with anyone other than TAs and instructor.
- For programming assignments, we will run an automated test to check for unusually similar programs. Those that are similar to the extent that they appear to be derived from the same code base - in whole or in part - will be referred to the appropriate honor council.
- See the syllabus (to be posted) on Sakai for the full policy.

Some Administrative Details

Sakai

- Log into Sakai: http://sakai.umji.sjtu.edu.cn/portal

- Check the class webpage on the Sakai regularly for
  - Announcements
  - Slides

Some Administrative Details

Getting Help

- If you have questions, come to see TAs and instructor during the office hour!
  - We will hardly answer your questions through email, because it is inefficient.

Prerequisite

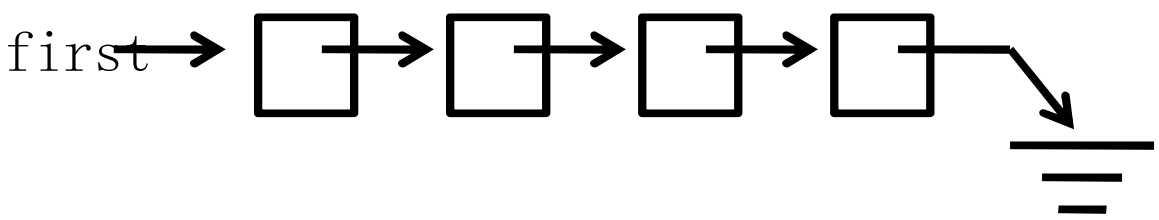
- Ve280 Programming and Elementary Data Structures is required.
  - Compiling and debugging on Linux operating systems
  - C++ programming
  - Recursion
  - Pointers and arrays
  - Structs and enums
  - I/O streams, including file I/O
  - Abstract data types and classes
  - Dynamical memory management
  - Template and polymorphism
- Knowledge of Discrete Mathematics (Ve203) is

Data Structures and Algorithms

- Data structures is concerned with the representation and manipulation of data.

- All programs manipulate data.
- So, all programs represent data in some way.

- Example: linked list



Logical versus Physical Form

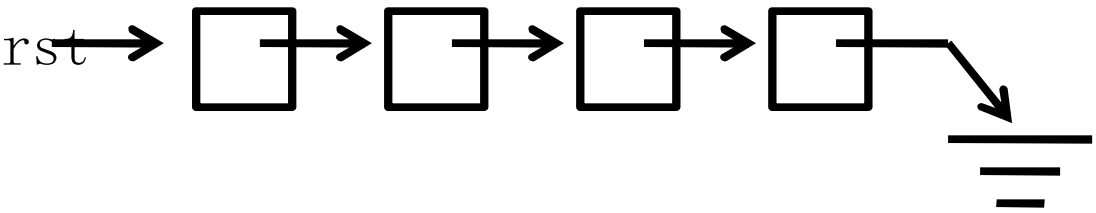
- Data items have both a logical and a physical form.

- Logical form: definition of the data item within an abstract data type (ADT).
  - Example: Integers in mathematical sense: +, -

- Physical form: implementation of the data item within a data structure.
  - Example: 32 bit integers.

Data Structure Example: Linked List

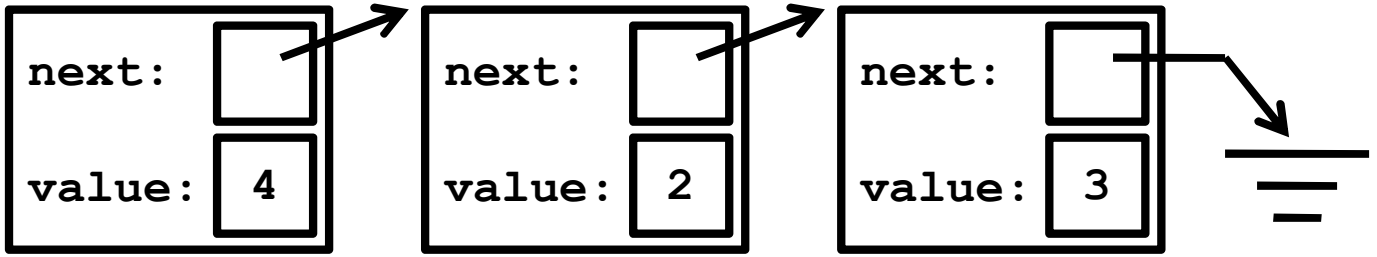
Logical Form



Physical Form

```
class IntList {
  node *first;
 public:
  ...
};
```

```
struct node {
  node *next;
  int   value;
};
```



Data Structures and Algorithms

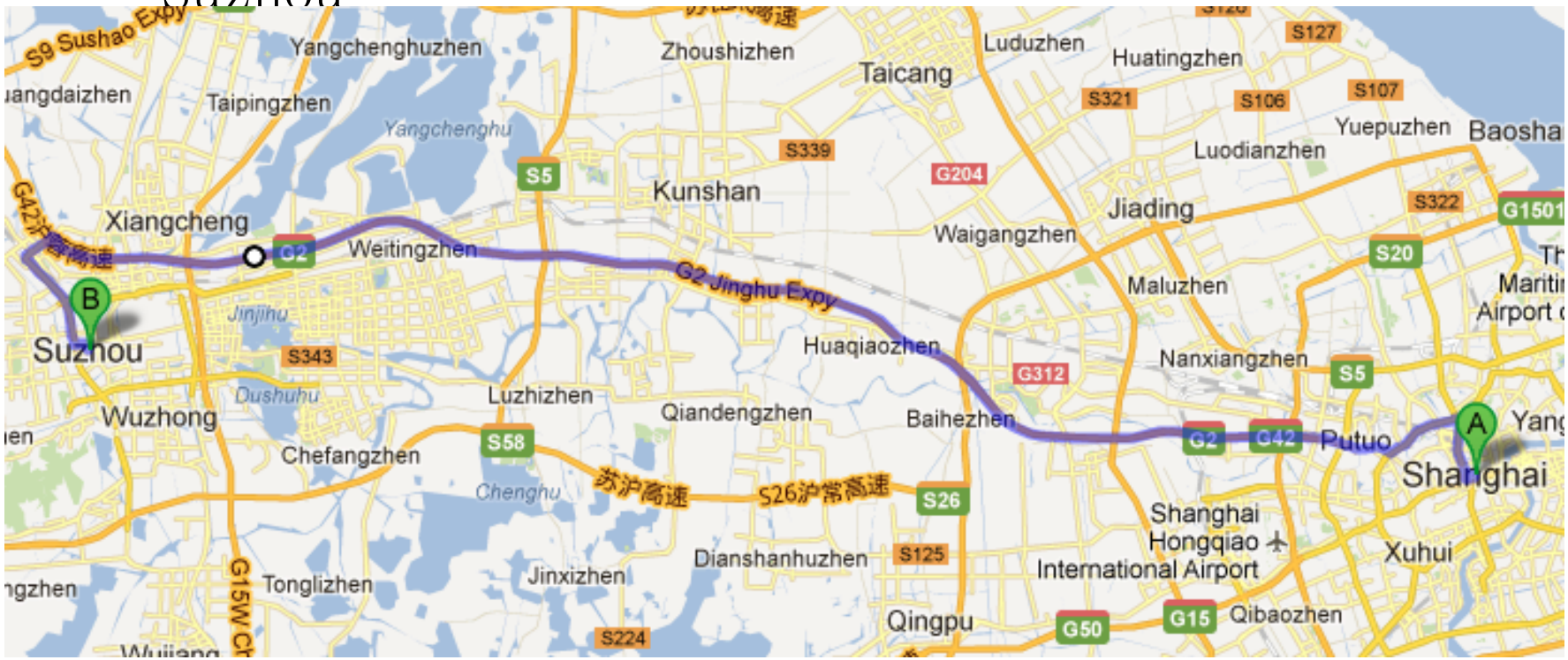
- Data manipulation requires an algorithm - a sequence of steps that solves a specific task.

- Data structures + Algorithms = Programs

- The study of data structures and algorithms is fundamental to Computer Science.

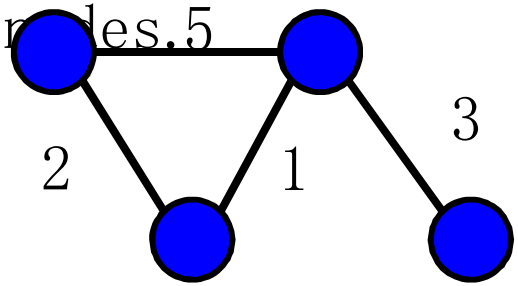
Real World Problem: Navigation

- Finding the shortest route from Shanghai to Suzhou



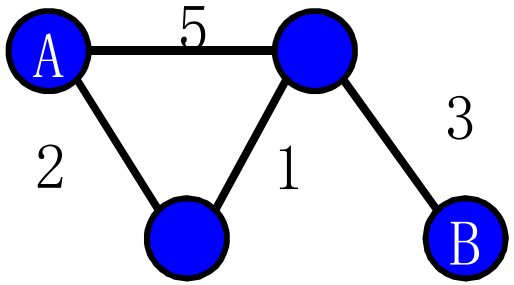
Real World Problem: Navigation

- What information do we need?
  - Streets.
  - Intersections of streets. (We assume that our departure place and destination are at certain intersections.)
- How do we store the information in computer?
  - Graph: consisting of “nodes” and “edges”.
  - Each edge has a weight to denote the distance between two nodes.



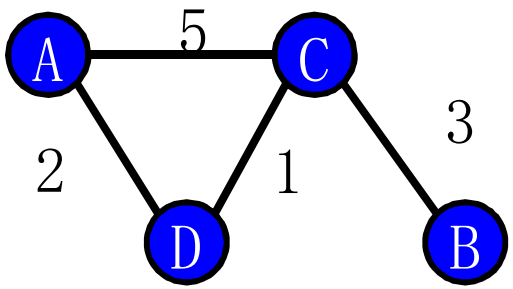
Real World Problem: Navigation

- The algorithm: finding the shortest path from a source node (A) to a sink node (B).



Challenges: Efficiency

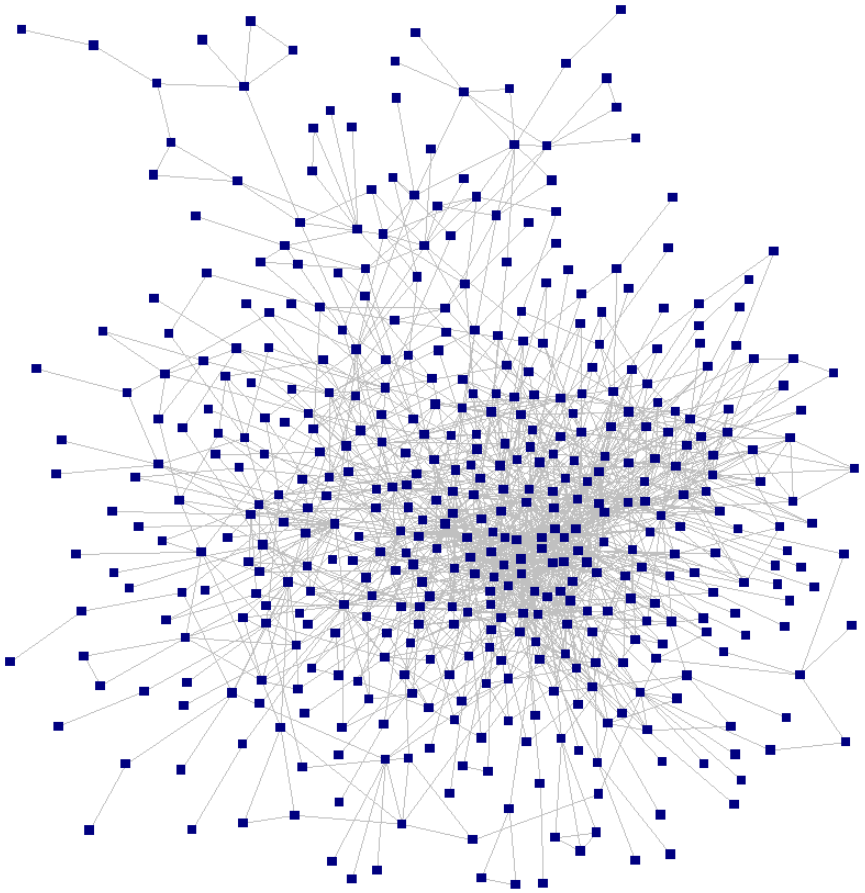
- For a small number of nodes, we can enumerate all the possible paths.



- Path A → C → B: 8;
- Path A → D → C → B: 6;
- The minimum is 6.

Challenges: Efficiency

- However, in real world, the graph is much more complicated.
- It is impossible to enumerate all the possible paths!
- How can we solve the problem?



More about Efficiency

- Choice of data structures or algorithms can make the difference between a program running in a few seconds or many days.
- Example: Number of comparisons for linear search and binary search (Worst Case)

Input Size	Linear	Binary	Ratio (L/B)
64	64	6	10.7
128	128	7	18.3
256	256	8	32
512	512	9	56.9
1024	1024	10	102.4

More about Efficiency

- A solution is said to be efficient if it solves the problem within its resource constraints.
  - Space: Memory
  - Time
- The cost of a solution is the amount of resources that the solution consumes.
- We value efficiency of the data structures and algorithms!
- We will learn how to analyze their efficiency

Course Objectives

- Learn the tool:
  - Common data structures and algorithms
  - And their efficiency

- Apply the tool
  - Choose the right tool: some tools are better for certain tasks than other tools. Do performance analysis.
  - Solve a problem using existing data structures and algorithms.
  - Improve the data structures and algorithms for a more efficient solution.

Topics

- Asymptotic algorithm analysis
- Data structures
  - Linked list, stacks, and queues
  - Trees, including binary search tree
  - Hash table
  - Heaps
  - Graphs
- Algorithms
  - Graph-related algorithms, such as minimum spanning tree, shortest path
  - Divide and conquer
  - Dynamic programming

Questions?

Asymptotic Algorithm Analysis

How to Measure Efficiency?

- Empirical comparison: run programs
  - Use the wall-clock time to measure the runtime
  - Empirical comparison could be tricky. It depends on
    - Compiler
    - Machine (CPU speed, memory, etc.)
    - CPU load

- Asymptotic Algorithm Analysis
  - For most algorithms, running time depends on the "size" of the input.
  - Running time is expressed as $T(n)$ for some function T on input size n.

Input Dependency: Example

- Summing an array of n elements

```
// REQUIRES: a is an array of size n
// EFFECTS: return the sum
int sum(int a[], unsigned int n) {
  int result = 0;
  for(unsigned int i = 0; i < n; i++)
    result += a[i];
  return result;
}
```

- The runtime is cn, where c is some constant.
- With n fixed, any array has roughly the **same** runtime.

Best, Worst, Average Cases

- In the example of summing an array, all inputs of a given size take the same time to run.
- However, in some other cases, this is not true, i.e., not all inputs of a given size take the same time to run.
- Example: linear search

```
// REQUIRES: a is an array of size n
// EFFECTS: return the index of the element
// equals key. If no such element, return n.
int search(int a[], unsigned int n, int key) {
  for(unsigned int i = 0; i < n; i++)
    if(a[i] == key) return i;
  return n;
}
```

Exercises

- What is the best case for linear search?
  - Data is found in the first place you look.
- What is the worst case?
  - Data is found in the last place.
  - Or data is not found.
- What is the average case?
  - Average performed over all possible inputs of a given size.

Best, Worst, Average Cases

- Best case: least number of steps required, given the ideal input
- Worst case: most number of steps required, given the most difficult input.
- Average case: average number of steps required, given any input.

Which Case to Use?

- Average case or worst case is common.
- While average time appears to be the fairest measure, it may be difficult to determine.
- Worst case is pessimistic, but it gives an upper bound.

Asymptotic Analysis: Big-Oh

- Definition: A non-negatively valued function, $T(n)$, is in the **set** $O(f(n))$ if there **exist** two positive constants c and n_0 such that $T(n) \leq cf(n)$ for all $n > n_0$.

- Usage: The algorithm is in $O(n^2)$ in best/average/worst case.

- Meaning: For all data sets big enough (i.e., $n > n_0$), the algorithm always executes in **less than** $cf(n)$ steps in best/average/worst case.

Big-Oh Example

- Finding a value X in an array using linear search (**average cost**).

- Solution:
  - Average cost is $T(n) = kn/2$, where k is a constant.
  - For all values of $n > 1$, $kn/2 \leq kn$.
  - Therefore, the definition is satisfied for $f(n) = n$, $n_0 = 1$, and $c = k$. ($T(n) \leq cf(n)$ for all $n > n_0$)
  - Hence, $T(n)$ is in $O(n)$.

Big-Oh Notation

- Big-oh notation indicates an **upper bound**.
- Example: If $T(n) = 3n^2$ then $T(n)$ is in $O(n^2)$.
- Look for the **tightest** upper bound:
  - While $T(n) = 3n^2$ is in $O(n^3)$, we prefer $O(n^2)$.

Exercise

- Consider $T(n) = c_1 n^2 + c_2 n$, where c_1 and c_2 are positive.
- What is the big-oh notation for $T(n)$? Prove it strictly, i.e., finding $f(n)$, c, and n_0 so that $T(n) \leq cf(n)$ for all $n > n_0$.

- Solution:
  - $c_1 n^2 + c_2 n \leq c_1 n^2 + c_2 n^2 \leq (c_1 + c_2) n^2$ for all $n > 1$.
  - $f(n) = n^2, c = c_1 + c_2, n_0 = 1$

A Common Misunderstanding

- "The best case for my algorithm is $n = 1$ because that is the fastest."

- Wrong!
- Big-oh refers to a **growth rate** as n grows to ∞.
- Best case is defined for the input of size n that is cheapest among all inputs of size n.

Relative of Big-Oh: Big-Omega

- Definition: For $T(n)$ a non-negatively valued function, $T(n)$ is in the **set** $\Omega(g(n))$ if there **exist** two positive constants c and n_0 such that $T(n) \geq cg(n)$ for all $n > n_0$.

- Meaning: For all data sets big enough (i.e., $n > n_0$), the algorithm always requires **more than** $cg(n)$ steps.

- Big-omega gives a lower bound.
- We usually want the greatest lower bound.