

VE281

Data Structures and Algorithms

Introduction and Asymptotic Algorithm Analysis

Time and Location

- **Time:** Monday 4:00-5:40 pm, Wednesday 4:00-5:40 pm, and Friday (odd week) 4:00-5:40 pm
- **Location:** East Middle Hall 1-107

Instructor

- Weikang Qian
- Email: qianwk@sjtu.edu.cn
- Phone: 3420-4020
- Office: Room 421, JI Building
- Office hour
 - Monday 1:00 – 2:00 pm
 - Wednesday 1:00 – 2:00 pm
 - Or *by appointment*

Teaching Assistants

- Zhou, Zhuangzhuang
 - Email: zhouzhuangzhuang@sjtu.edu.cn
 - Cell phone: 15600283638



- Wu, Yichen
 - Email: 553018598@qq.com
 - Cell phone: 13052445050



Textbooks for Reference (Not Required)

- “Data Structures and Algorithm Analysis,” by Clifford Shaffer.
Online available:
<http://people.cs.vt.edu/~shaffer/Book/C++3e20120605.pdf>
- “Algorithms,” by S. Dasgupta, C. Papadimitriou, and U. Vazirani.
- “Introduction to Algorithms, 3rd edition,” by Thomas Cormen et al., MIT Press, 2009.
- “Data Structures and Algorithms with Object-Oriented Design Patterns in C++,” by Bruno Preiss.

Grading

- Composition
 - In class quiz: 5%; No intern on the lecture day!
 - 6 written assignments: 20%
 - 5 programming assignments: 30%
 - Midterm exam: 20%
 - Final exam: 25%
- We will curve the final grades, if necessary.
- Questions about the grading?
 - Must be mentioned to the instructor or the TAs **within one week** after receiving the item.

Programming Assignments

- We require you to develop your programs using C++ on **Linux operating systems** with the compiler g++.
- C++11 standard is allowed.
 - Compile with the option `-std=c++11`
- We will grade your programs in the Linux environment: they must compile and run correctly on this operating system.
- Do experiments on algorithms, e.g., sorting algorithm

Assignment Deadline

- Each written assignment must be turned in before class on each due date.
- Each programming assignment (PA) must be turned in by 11:59 pm on the due date to be accepted for full credit.
 - However, we still allow you to submit your PA within 3 days after the due date, but there is a late penalty.

Hours Late	Scaling Factor
(0, 24]	80 %
(24, 48]	60 %
(48, 72]	40 %

- No PA will be accepted if it is more than 3 days late!

Assignment Deadline

- In very occasional cases, we accept deadline extension request.
 - Contact me, not TAs!
 - **ONLY** be granted for **documented** medical/personal emergencies that could not have been anticipated.
 - **NOT** granted for reasons such as accidental erasure/loss of files and outside conflicting commitments.

Some Suggestions

- Taking notes in class is a good idea.
- Start doing the homework early!
 - Don't wait until the last minute. Numerous lessons before
- Back up your code frequently in case your computer crashes.
 - Consequence: “computer crash” is NOT a reason for late submission!

Exams

- Written exams.
 - Some short questions
 - Some algorithm design problems
- Closed book and closed notes.
- No electronic devices are allowed.
 - These include laptops and cell phones.

Collaboration and Cheating

- You may discuss in oral with your classmates.
- **But** you must do all the assignments yourself.
- Some behaviors that are considered as cheating:
 - Reading another student's answer/code, including keeping a copy of another student's answer/code.
 - Copying another student's answer/code, in whole or in part.
 - Having someone else write part of your assignment.
 - Using test cases of another student.

“**Another student**” includes a student in the current semester or in the previous semester.

Collaboration and Cheating

- The previous lists of behaviors are deliberate cheating, but some unintentional actions could make you look like cheating. For example,
 - Testing your code with another one's account. Another's code may be overwritten by you. So, we see two identical copies.
 - You use another's computer to upload your code (in some cases like network/computer problems), but upload another's copy.
- We suggest you not to do those “dangerous” things.
 - If due to network/computer problem, you cannot upload, then send your code to TA's by email. By this way, you can double check the attachment.

Collaboration and Cheating

- In summary, you should be responsible for all answers/codes you submit. If you submit a copy of another student's work (or overwrite another student's work), it is considered cheating, **no matter of the reason!**

Collaboration and Cheating

- Any suspect of cheating will be reported to **the Honor Council at JI**.
- For programming assignments, we will run an automated test to check for unusually similar programs. Those that are highly similar - in whole or in part - will be reported to **the Honor Council at JI**.
- The result of confirmed cheating:
 - Reduction of the grade for that assignment to zero, **plus**
 - Reduction of the final grade for the course by one grade point, e.g., B+ \rightarrow C+

Canvas

- Log into Canvas: <https://umjicanvas.com>
- Check the class webpage on the Canvas regularly for
 - Announcements
 - Slides
 - Assignments

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
 - Compiling and debugging on Linux operating systems
 - C++ programming, including pointers, arrays, structs, etc.
 - Recursion
 - I/O streams, including file I/O
 - Classes
 - Dynamical memory management
 - Template
 - Linked list, stack, and queue

Prerequisite

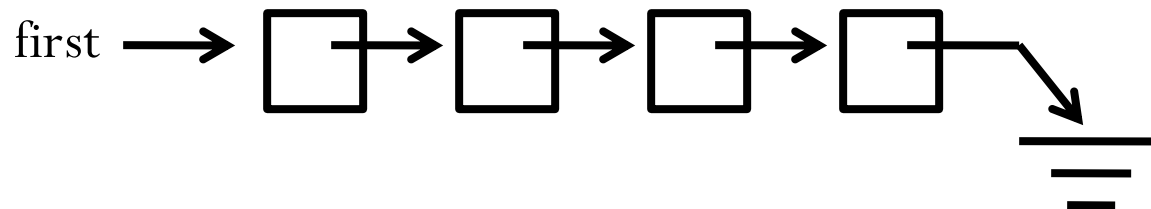
- Ve203 Discrete Mathematics
 - Computational complexity analysis
 - Some basic sorting algorithm, e.g., bubble sort, insertion sort, merge sort
 - Divide-and-conquer algorithm, master theorem
 - Graph, graph representation, depth first search, Dijkstra's algorithm (shortest path)
- Some important concepts will be reviewed

References and Copyright

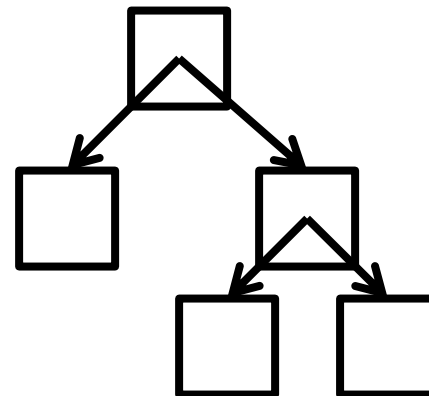
- Slides used (modified when necessary)
 - Sugih Jamin, University of Michigan
 - Sartaj Sahni, University of Florida
 - Bert Huang, Columbia University
 - Tim Roughgarden, Stanford University
 - Clifford Shaffer, Virginia Tech

Data Structures and Algorithms

- Data structure is a particular way of organizing data in a computer so that it can be used efficiently.
 - Example: linked list



- We can store a set of records as a linked list
 - or as a tree (to be talked later).

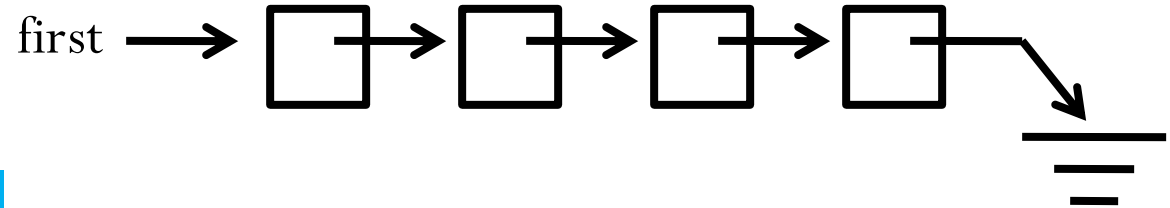


Logical versus Physical Form

- A data structure have both a **logical** and a **physical** form.
- Logical form: definition of the data structure at an abstraction level.
- Physical form: implementation of the data structure.

Data Structure Example: Linked List

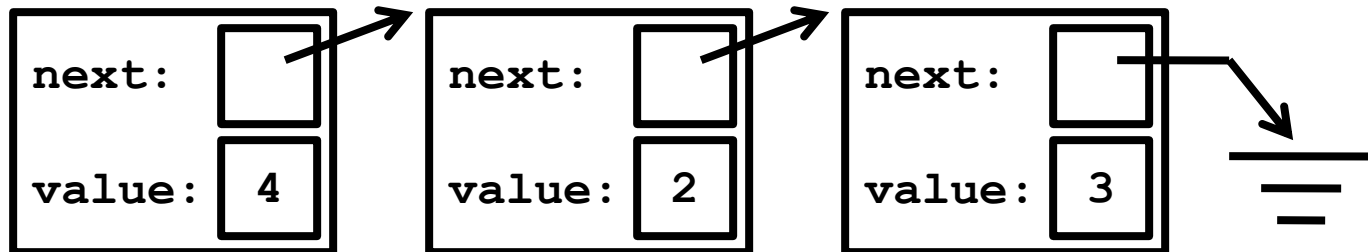
Logical Form



Physical Form

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

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

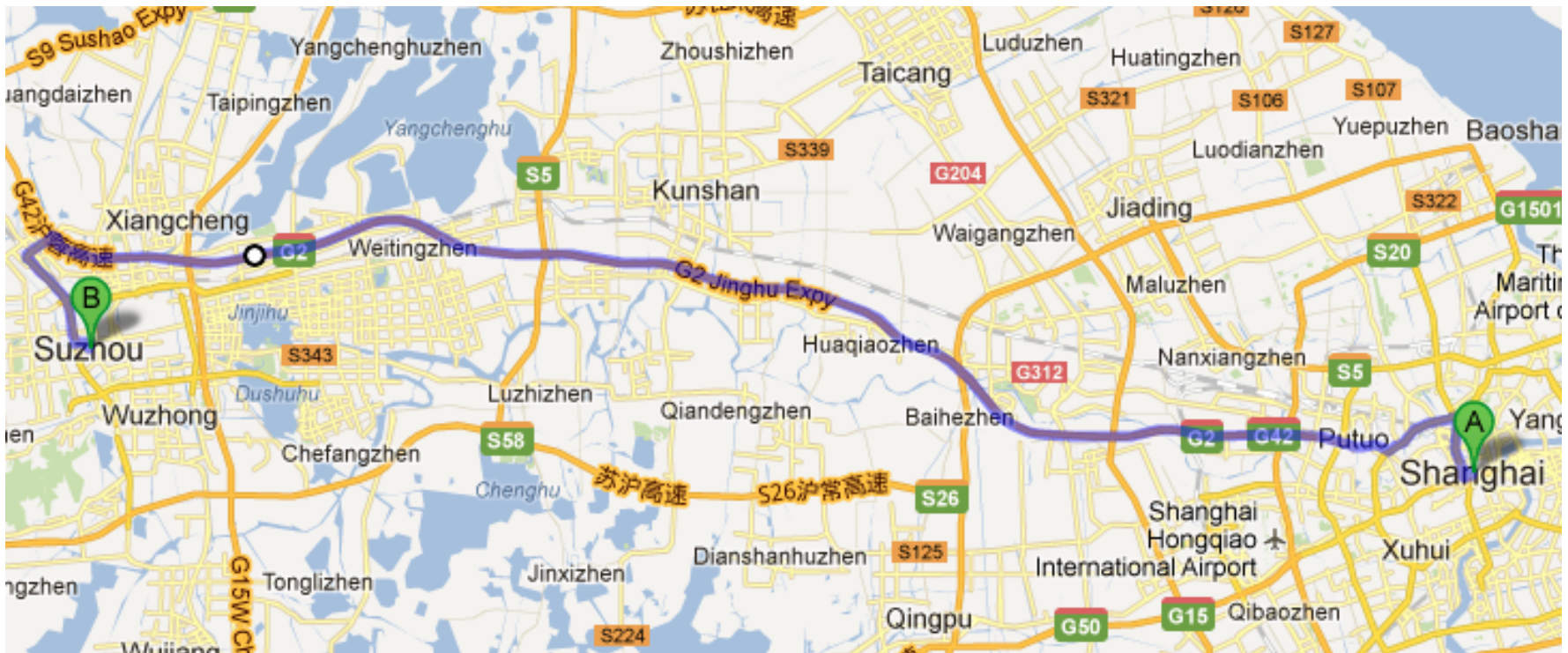


Data Structures and Algorithms

- Data manipulation requires an algorithm – a sequence of steps that solve a specific task.
- Data structures + Algorithms = Programs
- The study of data structures and algorithms is fundamental to Computer Science.
 - Database related to balanced binary search tree.
 - Computer networks related to shortest path algorithm.
 - ...

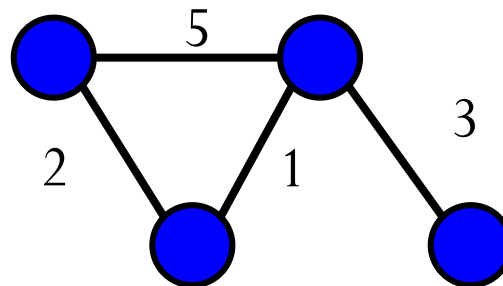
Real World Problem: Navigation

- Finding the shortest route from Shanghai to Suzhou



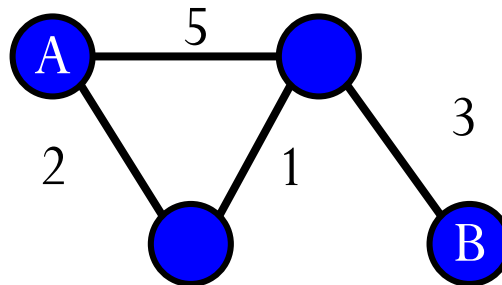
Real World Problem: Navigation

- 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.



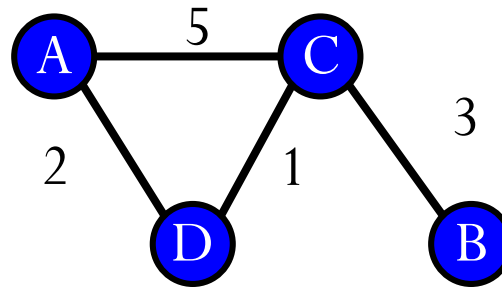
Real World Problem: Navigation

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



Challenges: Efficiency

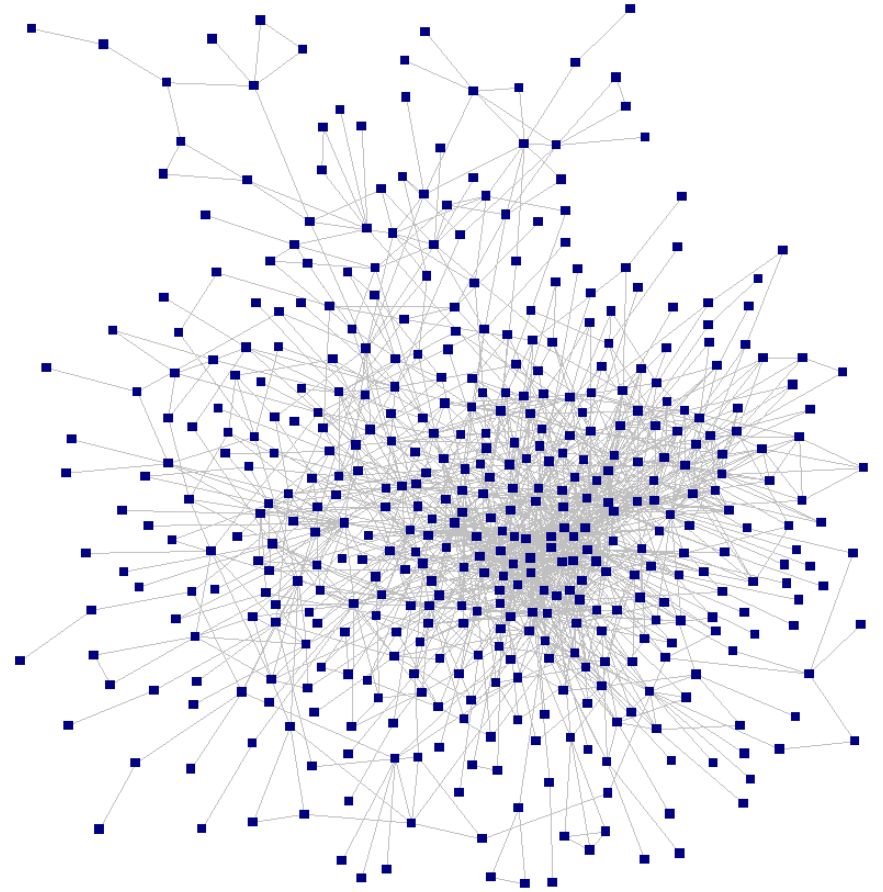
- For a small number of nodes, we can enumerate all the possible paths.



- Path $A \rightarrow C \rightarrow B$: 8;
- Path $A \rightarrow D \rightarrow C \rightarrow B$: 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?
 - Dijkstra's algorithm



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, i.e. memory consumption
 - Time ✓ **Our major concern**
- 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
 - Solve a problem using existing data structures and algorithms.
 - Choose the right tool: some tools are better for certain tasks than other tools. Do performance analysis.

Topics

- Asymptotic Algorithm Analysis
- Data structures
 - Trees, including binary search tree, balanced binary search tree
 - Hash table
 - Heaps
 - Graphs
- Algorithms
 - Sorting and searching
 - Graph-related algorithms, such as minimum spanning tree, topological sorting
 - Dynamic programming
 - Branch-and-bound

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 roughly 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, corresponding to the ideal input
- Worst case: most number of steps required, corresponding to the most difficult input.
- Average case: average number of steps required, given any input.

A Common Misunderstanding

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

- Wrong!
- Best case is a special input case of size n that is **cheapest** among all input cases of size n .

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.
 - Sometime, it requires domain knowledge, e.g., the distribution of inputs.
- Worst case is pessimistic, but it gives an upper bound.
 - Bonus: worst case usually easier to analyze.