Data Structures and Algorithms

数据结构与算法

乔海燕
qiaohy@mail.sysu.edu.cn

Lectures

- 3 hour lecture every week
- 2 hour lab session every weej
- Check course page regularly: elearing.sysu.edu.cn
- Important to attend both the lectures and lab sessions, and to finish assignments on time.

References

•*Data Structures and Algorithm Analysis in C++,* 4th edition, Mark A. Weiss.
•数据结构，严蔚敏，吴伟民，清华大学出版社。
•*The art of computer programming, V.1, V. 3,* Donald Knuth.
•*Introduction to Algorithms,* Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rovest and Clifford Stein.
• *Robert L. Kruse*. Data Structures and Program Design in C++, 2nd Edition. Prentice Hall 2000.
• C++ Official Website: http://www.cplusplus.com
•Tons of learning materials on the internet.

Grading Scheme

- Grading is based on
  - Home work (online exercises, written assignments or programming assignments) and attendance
  - Midterm examination
  - Final examination
  - Bonus: active participation
- Scores are given for both “theories” and “labs”.
- Theory grade: (home work, midterm, participation)60% + final 40%
- Lab grade: (assignments, tests, participation)70% + final 30%

Assignments

- Everything will be available online, and check course page regularly
- Written assignments and on-line exercises
  - Due by time specified
  - Finished independently
- Programming assignments
  - Due by time specified
  - Run on PC
  - group collaboration of 2 people is encouraged for the lab sessions.

Plagiarism Policy

•1st: both get negative;
• cannot be trusted;
• used to be failed in the final.

You are encouraged to collaborate in study groups.
But, you cannot copy or slightly change other students' solutions or codes.

Course Overview

- A fundamental computer science course
  - Essential for programming
  - Essential for advanced courses
- A challenging course, which concerns problem solving by programming, and needs
  - Programming
  - Mathematical thinking and Algorithmic thinking.

Why Data Structures Matter?

- Problem: Write a program which can search a person's telephone number from a pile of name cards?

From problem space to solution space

(左建卫，84458290）

（Zuo jianwei, 84458290)
(Liu Ning, 84138355)
…
(Gao Min, 39943222)

From problem space to solution space

Abstraction:

- Represent physical items by data, for example, a card is represented by its name and phone number ("Gao Ming", "40088888").
- A pile of cards -> a set of data of the form (name, phone number);
- Now the problem is: given a set of the pair (name, phone number) and a name target, which you want to look up, for example, "Gao Ming", is there a pair whose first component is target? If yes, return its second component. Otherwise, fail.

From Problem Space to Solution Space

Solutions: think about how you do it in practice

- We want to *organize* the set of data, for example, organize them into a sequence, and then you look at each data one by one.
- We say that the set of data has a linear *logical structure*

(Wang Bingbing, 98876544)
(Zuo jianwei, 84458290)
(Liu Ning, 84138355)
…
(Gao Min, 39943222)

From Problem Space to Solution Space

Design the Algorithm, which is a list of well-defined instructions that solve a problem.

- How to represent data? Use *data type*.
- How to represent each pair (name, phone number)?
  - Use record type
- How to represent the set of pairs, or how to store the set of data?
  - Using array, for example,
- The algorithm: given an array of records and the target, compares every record in the array with target until the name of the record equals to target.
- Implement the algorithm, or write the program.

Solution 1

- Input is a (*linear)* sequence of records and a target
- Storage: using arrays
- Algorithm: sequential search

C++:

```
struct Card {
  string name;
  string phone;
} ;
```

```
int sequentialSearch(Card st[], int n, const string &target) {
 /*search a record whose name is target and returns the position of the record if
search is successful, returns n otherwise.
 */
  int i;
  for (i = 0; i<n && st[i].name != target;i++);
  return i;
 }
```

Evaluation of the algorithm

- Is the algorithm correct?
  - Proving
- Is the implementation correct?
  - Testing
- How efficient is the search algorithm?
  - Algorithm analysis

Solution 2

- Input is a (*linear)* sequence of records and a target
- Storage: using linked lists
- Algorithm: sequential search

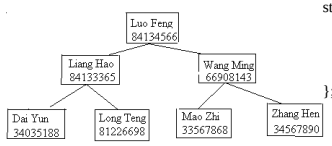
用C++表示：

```
struct Node{
  Card  data;
  Node * next;
} ;
```

```
Node* sequentialSearchLinked(Node *head, const string &target)
{
 /*search a record whose name is target and returns the pointer to the record if
search is successful, returns NULL otherwise.
 */
  Node *p=head;
  while (p!=NULL&&(p->data).name!=target) {
     p=p->next;
  };
  return p;
}
```

Solution 3 & 4

- Using binary search: input is ordered by name; storage using arrays and binary search;
- Using binary search trees: input is organised as an ordered tree, tree is stored using linked representation and using binary tree search.




```
struct TreeNode{
  Card data;
  TreeNode *llink,*rlink;
};
```

Solution 5

- Using ADT (abstract data type) map

```
int main() {
  map<const char*, string,ltstr> phones; //initialize a map container
  phones["Dai Yun"]      = "34035188"; //insert a record
  phones["Liang Hao"]   = "84133365";
  phones["Zhang Hen"]  = "34567890";
  phones["Wang Ming"] = "66908143";
  phones["Luo Feng"]     = "84134566";
  map<const char*, string, ltstr>::iterator cur = phones.find("Zhang Min");
  if (cur != phones.end())
  //cur points to the end of the map and to the record othereise.
      cout << "Zhang Min's phone is " << (*cur).second << endl;
  else
      cout << "Zhang Min doesn't exist in phones." <<endl;
}
```

From Problem Space to Solution Space

- Abstraction: what are the input data, the operations and output data?
- One needs to organise the data or design the logic structure to be able realize the operations;
- The data and operations can be packed into an ADT(Abstract Data Type);
- Decide how to store the data or design physical structures of the data;
- Design the algorithm;
- Evaluate the solution: running time and storage;
- Some solutions are much better than others. Some takes O(log n) running time while other take O(n).
- Implement the algorithm, and test it again its specification.

Course Outline

- Algorithmic asymptotic analysis
  - Big-Oh, big-Theta, and big-Omega
- ADTs: Lists, stacks, and queues
- Sorting
  - Insertion, mergesort, quicksort, heapsort, radix, etc.
  - Lowest bound on sorting
- Search trees
  - Binary search tree
  - AVL tree
  - B- tree
- Hashing
- Graphs
  - Breadth-first search and Depth-first search
  - Shortest paths
  - Minimal spanning trees

19

Course Objectives

Having successfully completed this course, the student will be able to:

- Choose the data structures that effectively model the information in a problem.
- Judge efficiency trade-offs among alternative data structure implementations or combinations.
- Apply algorithm analysis techniques to evaluate the performance of an algorithm and to compare data structures.
- Implement and know when to apply standard algorithms for searching and sorting.
- Design, implement, and test programs using a variety of data structures including hash tables, binary and general tree structures, search trees, tries, heaps, graphs, and B-trees.

Overall Goals of the Course

- Learn to solve problems by programming
- Learn to analyze and evaluate your solutions
- From programmer to architect

基本概念和术语

– 数据 (Data)
  - 数据是客观事物的符号表示，在计算机科学中指的是所有能输入到计算机中并被计算机程序处理的符号的总称。

– 数据元素 (Data Element)
  - 数据元素是数据的基本单位，在程序中通常作为一个整体来进行考虑和处理。
  - 一个数据元素可由若干个数据项 (Data Item)组成。数据项是数据的不可分割的最小单位。数据项是对客观事物某一方面特性的数据描述。

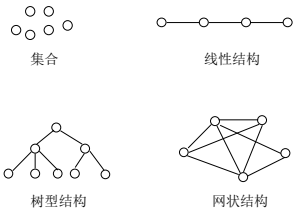
– 数据对象 (Data Object)
  - 数据对象是性质相同的数据元素的集合，是数据的一个子集。如字符集合 C = {'A', 'B', 'C', … } 。

基本概念和术语

– 数据的逻辑结构
  - 数据结构 (Data Structure) 是指相互之间具有 (存在) 一定联系 (关系) 的数据元素的集合。数据元素之间的相互联系 (关系) 称为数据的逻辑结构，它们可以是自然的或者是人为约定的。数据的逻辑结构有四种基本类型：
    – 集合：结构中的数据元素除了“同属于一个集合”外，没有其它关系。
    – 线性结构：结构中的数据元素之间存在一对一的关系。
    – 树型结构：结构中的数据元素之间存在一对多的关系。
    – 图状结构或网状结构：结构中的数据元素之间存在多对多的关系。

基本概念和术语

– 数据的逻辑结构
  - 数据的四种逻辑结构的图示：



基本概念和术语

– 数据的逻辑结构
- 例：设数据逻辑结构 B = (K, R)

  $K = \{k_1, k_2, \ldots, k_9\}$

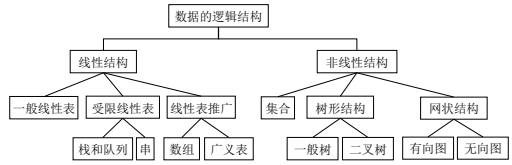
  $R = \{<k_1, k_3>, <k_1, k_8>, <k_2, k_3>, <k_2, k_4>, <k_2, k_5>, <k_3, k_9>,$

  $<k_4, k_7>, <k_4, k_6>, <k_5, k_6>, <k_8, k_9>, <k_9, k_7>\}$

  画出此逻辑结构的图示，并确定哪些是起点，哪些是终点。

基本概念和术语

- **基本概念和术语**
  – 数据的逻辑结构
  - 数据逻辑结构层次关系图



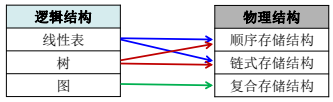
基本概念和术语

– 数据结构的存储方式 (数据的物理结构)
- 数据结构在计算机内部的存储方式包括数据元素的存储和数据元素之间的逻辑关系的表示，也称为数据的物理结构。
  – 数据元素之间的逻辑关系在计算机内部有两种不同的表示方法：顺序表示和非顺序表示。由此得出两种不同的存储结构：顺序存储结构和链式存储结构。
  – 顺序存储结构：
    » 用数据元素在存储器中的相对位置来表示数据元素之间的逻辑关系，通常表现为连续的存储地址。
  – 链式存储结构：
    » 在每一个数据元素中增加一个存放另一个元素地址的指针 (pointer)，用该指针来表示数据元素之间的逻辑关系，因此对地址的连续性没有要求。

基本概念和术语

– 数据结构的存储方式 (数据的物理结构)
- 数据的逻辑结构和物理结构是密不可分的两个方面，一个算法的设计取决于所选定的逻辑结构，而算法的实现依赖于所采用的存储结构。
- 例：在 C 语言中，用一维数组表示顺序存储结构；用带指针的结构体类型表示链式存储结构。

– 数据的逻辑结构与物理结构之间的关系



基本概念和术语

– 数据结构 (Data Structure)
- 数据结构的形式定义
  – 数据结构是一个二元组：

    Data-Structure = (D, S)

    其中：D 是数据元素的有限集，S 是 D 上关系的有限集。
- 数据结构的三个组成部分：
  – 逻辑结构：
    » 数据元素之间逻辑关系的描述

      Data-Structure = (D, S)
  – 存储结构：
    » 数据元素在计算机内部的存储及其逻辑关系的表示
  – 数据操作：
    » 对数据进行的运算

基本概念和术语

– 数据结构 (Data Structure)
- 数据结构的主要运算包括：

  (1) 建立 (Create) 一个数据结构；
  (2) 消除 (Destroy) 一个数据结构；
  (3) 从一个数据结构中删除 (Delete) 一个数据元素；
  (4) 向一个数据结构插入 (Insert) 一个数据元素；
  (5) 对一个数据结构进行访问 (Access)；
  (6) 对一个数据结构 (中的数据元素) 进行修改 (Modify)；
  (7) 对一个数据结构进行排序 (Sort)；
  (8) 对一个数据结构进行查找 (Search)。

基本概念和术语

- 数据类型 (Data Type)
  - 数据类型指的是一个值的集合和定义在该值集上的一组操作的总称。
  - 数据类型是和数据结构密切相关的一个概念。
    - 在 C 语言中数据类型有：基本类型和构造类型。
  - 数据结构不同于数据类型，也不同于数据对象，它不仅要描述数据类型的数据对象，而且要描述数据对象各元素之间的相互关系。

基本概念和术语

- 抽象数据类型 (Abstract Data Type, ADT)
  - ADT 指一个数学模型以及定义在该模型上的一组操作。
  - ADT 仅仅是一组逻辑特性描述， 与其在计算机内的表示和实现无关。因此，不论 ADT 的内部结构如何变化，只要其数学特性不变，都不影响其外部的使用特征。
  - ADT 的形式化定义

    ADT = (D, S, P)

    其中：D 是数据对象，S 是 D 上的关系集，P 是对 D 的基本操作集。

基本概念和术语

- 抽象数据类型 (Abstract Data Type, ADT)
  - ADT 的一般定义形式

    ADT <抽象数据类型名> {
    数据对象: <数据对象的定义>
    数据关系: <数据关系的定义>
    基本操作: <基本操作的定义>
    } ADT <抽象数据类型名>

    其中数据对象和数据关系的定义用伪代码描述。

基本概念和术语

- 抽象数据类型 (Abstract Data Type, ADT)
  - ADT 基本操作的定义

    <基本操作名> (<参数表>)
    初始条件： <初始条件描述>
    操作结果： <操作结果描述>
    - 初始条件：描述操作执行之前数据结构和参数应满足的条件；若不满足，则操作失败，返回相应的出错信息。
    - 操作结果：描述操作正常完成之后，数据结构的变化状况和应返回的结果。

基本概念和术语

- 算法 (Algorithm)
  - 算法是对特定问题求解方法 (步骤) 的一种描述，是指令的有限序列，其中每一条指令表示一个或多个操作。
  - 算法具有以下五个特性
    (1) 有穷性：一个算法必须总是在执行有穷步之后结束，且每一步都在有穷时间内完成。
    (2) 确定性：算法中每一条指令必须有确切的含义，不存在二义性。
    (3) 可行性：一个算法是能行的。即算法描述的操作都可以通过已经实现的基本运算执行有限次来实现。
    (4) 输入：一个算法有零个或多个输入，这些输入取自于某个特定的对象集合。
    (5) 输出：一个算法有一个或多个输出，这些输出是同输入有着某些特定关系的量。

基本概念和术语

- 算法 (Algorithm)
  - 一个算法可以用多种方法描述，主要有：
    - 使用自然语言描述
    - 使用形式语言描述
    - 使用计算机程序设计语言描述
  - 算法和程序是两个不同的概念。
    - 一个计算机程序是对一个算法使用某种程序设计语言的具体实现。
    - 算法必须可终止意味着并非所有的计算机程序都是算法。

基本概念和术语

– 算法 (Algorithm)
  - 算法设计要求与算法的评价
    (1) 正确性 (Correctness)：算法应满足具体问题的需求。
    (2) 可读性 (Readability)：算法应容易供人阅读和交流。可读性好的算法有助于对算法的理解和修改。
    (3) 健壮性 (Robustness)：算法应具有容错处理能力。当输入非法或错误数据时，算法应能适当地作出反应并进行处理，而不会产生莫名其妙的输出结果。
    (4) 通用性 (Generality)：算法应具有一般性 ，即算法的处理结果对于一般的数据集合都成立。
    (5) 效率与存储量需求：效率指算法执行的时间；存储量需求指算法执行过程中所需要的最大存储空间。一般地，这两者与问题的规模有关。