Tamkang University Software Engineering Group
淡 江 軟 體 工 程 實 驗 室
http://www.tkse.tku.edu.tw/

資料結構與處理
Data Structure

本教材僅供修習學生閱讀使用，敬請尊重智慧財產權，勿非法使用本教材內容

Presented by：Ying-Hong Wang
E-mail：inhon@mail.tku.edu.tw
Date :December 7, 2014

課前說明

- 上課用書
  - Data Structures and Algorithms in Java 4/e Goodrich著 John Wiley出版 新月圖書代理
  - 請尊重智慧財產權，勿非法影印使用教科書與參考書籍及使用盜版軟體
- 上課方式
  - 將採多元授課方式，包含板書、 投影片講授、團隊討論或個人問答、團體或個人報告等翻轉或PBL教學，須牢記本課程的成績卡座號
- 上課規定
  - 進入手機改設震動或關機、不要私下講話 (私下講話者扣學期成績)
  - 上課時間僅可喝水、不可吃東西，用餐請利用課間(課前、課後)完成
- 考試方式
  - Closed book

課前說明

- 成績評定
  - 出席:15% 實習課:15% 作業+小考:40% 期中考:15% 期末考:15%
  - 出席成績：每週必點名一次，預計點名15次，每人每學期有3次免責權，缺席三次以上者才開始扣出席成績。相對地，全勤者學期成績另加3分。點名係用以區別出席上課與否，故無論任何原因缺席者，均不需要請假，亦不補點。
  - 遲到、早退者，列入紀錄，每累計達三次者，視同缺席一次
  - 一般作業可以遲交，逾時每24小時內扣10分，扣至0分為止
  - 隨堂作業、小考、課堂問答、指定報告等不受理補交或補考，敬請隨時攜帶課本、講義、筆記等。隨堂作業與小考僅收A4格式紙，敬請隨身準備，非指定格式，視同未交。
  - 期末考比校訂時間提前一週，含實習課內容。

TSEG Tamkang Universty Software Engineering Group 淡江軟體工程實驗室 http://www.tkse.tku.edu.tw/

課前說明

- 課前測驗
  - 本週五(9/19)辦理課前測驗
  - 0830起，測驗時間60分鐘
  - 本測驗不計分、無補測
  - 但是，未參加者需退選本課程或不可加選本課程
  - 測驗結果將作為分組依據、無分組無分數
  - 分組團體由授課教師分配，不可自行選換組
  - 課程進行中，個人表現與團隊表現都會左右每一位同學的全學期成績
  - 每個人的成績都是依賴全學期一點一滴的累加
  - 請跟大一的自己比較，知識要有增長、技能要有精進

http://mail.tku.edu.tw/inhon

對同學們的建議

- 在專業養成上
  - 奠定紮實的基礎
  - 養成終身學習的習慣
- 在人格養成上
  - 建立正確的態度
  - 處事三態：真誠、負責、合群

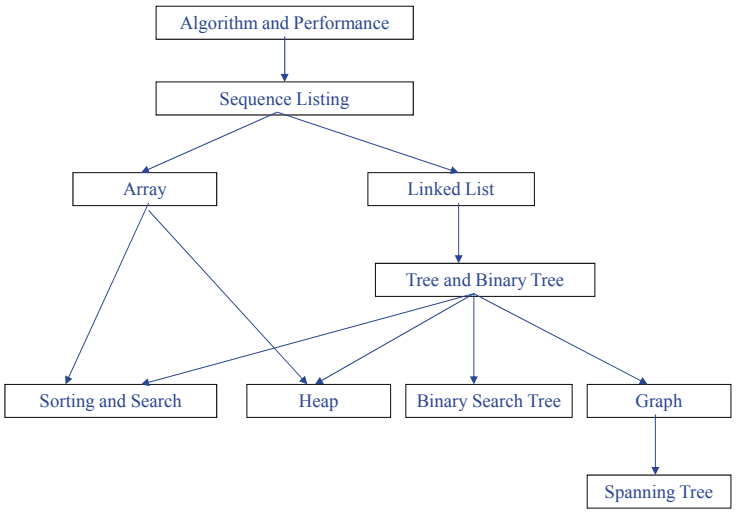
TSEG Tamkang University Software Engineering Group 淡江軟體工程實驗室 http://www.tkse.tku.edu.tw/

課程綱要

- Bridge-In
- Introduction
- Algorithm , Complexity analysis and Performance Evaluation
- Linked List
- Tree and Binary Tree
- Searching and Sorting
- Graphs
- 實習課部分
  - Array and Matrix (含Loaction的計算)
  - Stack and Queue in Array(含Infix與Postfix、Prefix的轉換)
  - Sorting and Search
  - Hashing functions

http://mail.tku.edu.tw/inhon

Learning Objects Architecture



Bridge-In

Why do we need to learn Data Structure

- The Problems of Programming
- Executable Programs v.s. Corrected Programs
- Corrected Programs v.s. Performance Programs
- Space and Execution Time

TSEG Tamkang Universty Software Engineering Group 淡江軟體工程實驗室 http://www.tkse.tku.edu.tw/

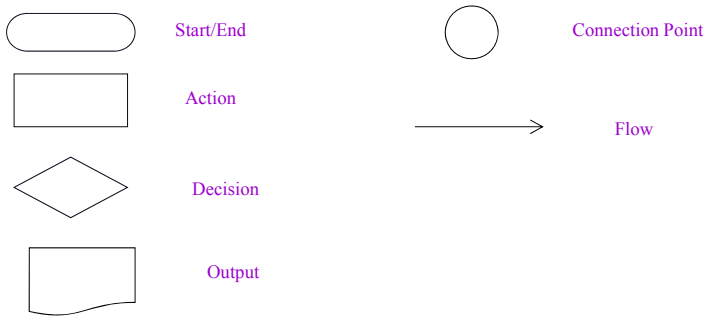
Presentation of Program Design Logic

- Flow Chart
  - Notation and Flow
- Algorithm
  - Pseudo Code
  - Programming Language Like
  - Natural Language

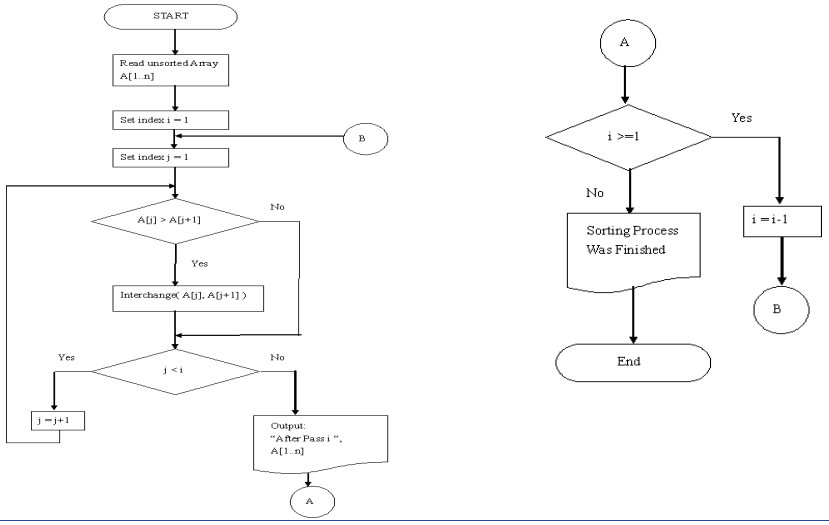
http://mail.tku.edu.tw/inhon

Presentation of Program Design Logic

- Notations of Flow Chart



Example in Bubble Sort



Comparison of Performance

- Bubble Sort
  - http://www2.lssh.tp.edu.tw/~hlf/class-1/lang-c/bubble_sort.htm
  - http://hwcode.blogspot.tw/2011/11/bubble-sort.html
- Quick Sort
  - http://flyinsky76.pixnet.net/blog/post/23556348-quicksort
  - http://program-lover.blogspot.tw/2008/11/quicksort.html
- 任務
  - 利用亂數產生10筆整數，同時交給Bubble sort與Quick sort排序，並記錄二者的執行時間。再分別執行1000、1000000筆資料

Introduction

Introduction

- System Life Cycles
- Decomposition
- Algorithms
- Performance Analysis
  - Space Complexity
  - Time Complexity
- Asymptotic Notation

TSEG Tamkang Universty Software Engineering Group 淡江軟體工程實驗室 http://www.tkse.tku.edu.tw/

System Life Cycles

- Requirements
- Analysis
- Design
- Refinement and Coding
- Verification
  - Correction proofs
  - Testing
  - Error Removal

http://mail.tku.edu.tw/inhon

Algorithmic Decomposition vs Object Oriented Decomposition

- **Algorithmic Decomposition:** view software as a process, break down the software into modules that represent steps of the process. Data structures required to implement the program are a secondary concern.
- **Object-Oriented Decomposition:** view software as a set of well-defined objects that model entities in the application domain.
  - Advantages
    - Encourages reuse of software
    - Allow software evolve as system requirements change
    - More intuitive

Algorithm、Performance Analysis & Performance Measurement

Algorithm

- Definition: An algorithm is a finite set of instructions that, if followed, accomplishes a particular task. In addition, all algorithms must satisfy the following criteria:
  1) Input. Zero more quantities are externally supplied.
  2) Output. At least one quantity is produced.
  3) Definiteness. Each instruction is clear and unambiguous.
  4) Finiteness. If we trace out the instructions of an algorithm, then for all cases, the algorithm terminates after a finite number of steps.
  5) Effectiveness. Every instruction must be basic enough to be carried out, in principle, by a person using only pencil and paper. It is not enough that each operation be definite as in 3) it also must be feasible.

TSEG Tamkang Universty Software Engineering Group 淡江軟體工程實驗室 http://www.tkse.tku.edu.tw/

Algorithm vs. Program

- Major Difference between Algorithm and Program by above Criteria

  It is -

  Finiteness.

  A program can always active and does not stop, for example Operating System, ATM, Alarm Monitor, and so on.

http://mail.tku.edu.tw/inhon

Selection Sort - Example

- Suppose we must devise a program that sorts a collection of $n \geq 1$ integers.

  From those integers that are currently unsorted, find the smallest and place it next in the sorted list.

- Problem in the above statement
  - Does not describe where and how the integers are initially sorted.
  - Does not indicate where to place the result.

TSEG Tamkang University Software Engineering Group 淡江軟體工程實驗室 http://www.tkse.tku.edu.tw/

C++ Program for Selection Sort

```
void sort (int *a, const int n)
// sort the n integers a[0] to a[n-1] into non-decreasing order
    for (int i = 0; i < n; i++)
    {
        int j = i;
        // find smallest integer in a[i] to a[n-1]
        for (int k = i + 1; k < n; k++)
            if (a[k] < a[j]) j = k;
        // interchange
        int temp = a[i]; a[i] = a[j]; a[j] = temp;
    }
}
```

http://mail.tku.edu.tw/inhon

Binary Search -Example

- Assume that we have $n \geq 1$ distinct integers that are already sorted and stored in the array $a[0] \ldots a[n-1]$. Our task is to determine if the integer x is present and if so to return j such that $x = a[j]$; otherwise return -1. By making use of the fact that the set is sorted, we conceive the following efficient method:

  Let *left* and *right*, respectively, denote the left and right ends of the list to be searched. Initially, *left* = 0 and *right* = n – 1. Let *middle* = (*left* + *right*) / 2 be the middle position in the list. If we compare a[*middle*] with *x*, we obtain one of the three results:

  (1) $x < a[middle]$. In this case, if *x* is present, it must be in the positions between 0 and *middle* – 1. Therefore, we set *right* to *middle* – 1.

  (2) $x == a[middle]$. In this case, we return *middle*.

  (3) $x > a[middle]$. In this case, if *x* is present, it must be in the positions between *middle*+1 and *n*-1. So, we set *left* to *middle*+1.

Algorithm for Binary Search

```
int BinarySearch (int *a, const int x, const int n)
// Search the sorted array a[0], ... , a[n-1] for x
{
    for (initialize left and right; while there are more elements;)
    {
        let middle be the middle element;
        switch (compare (x, a[middle])) {
            case '>': set left to middle+1; break;
            case '<': set right to middle -1; break;
            case '=': found x;
        } // end of switch
    } // end of for
    not found;
} // end of BinarySearch
```

Algorithm for Binary Search (Cont.)

```
int BinarySearch (int *a, const int x, const int n)
// Search the sorted array a[0], ... , a[n-1] for x
{
    for (int left = 0, right = n - 1; left <= right;)
    {
        int middle = (left + right) / 2;
        switch (compare (x, a[middle])) {
            case '>': left = middle+1; break;     // x > a[middle]
            case '<': right = middle -1; break;   // x < a[middle]
            case '=': return middle;              // x == a[middle]
        } // end of switch
    } // end of for
    return -1;
} // end of BinarySearch
```


Recursive Algorithms

- Direct Recursive
- Indirect Recursive
- E.g., [Recursive binary search]

```
int BinarySearch (int *a, const int x, const int left, const int right)
// Search the sorted array a[left], ... , a[right] for x
{
    if (left <= right;) {
        int middle = (left + right) / 2;
        switch (compare (x, a[middle])) {
            case '>': return BinarySearch(a, x, middle+1, right); // x > a[middle]
            case '<': return BinarySearch(a, x, left, middle -1);  // x < a[middle]
            case '=': return middle;                               // x == a[middle]
        } // end of switch
    } // end of for
    return -1;  // not found
} // end of BinarySearch
```

Performance Analysis

- Space Complexity: The space complexity of a program is the amount of memory it needs to run to completion.
- Time Complexity: The time complexity of a program is the amount of computer time it needs to run to completion.

Space Complexity

- A fixed part that is independent of the characteristics of the inputs and outputs. This part typically includes the instruction space, space for simple variables and fixed-size component variables, space for constants, etc.
- A variable part that consists of the space needed by component variables whose size is dependent on the particular problem instance being solved, the space needed by referenced variables, and the recursion stack space.
- The space requirement S(P) of any program P is written as S(P) = c +Sp where c is a constant

Time Complexity

- The time, T(P), taken by a program P is the sum of the compile time and the run (or execution) time. The compile time does not depend on the instance characteristics. We focus on the run time of a program, denoted by tp (instance characteristics).
- Note that a step count does not necessarily reflect the complexity of the statement.
- Step per execution (s/e): The s/e of a statement is the amount by which count changes as a result of the execution of that statement.

Time Complexity in C/C++

- General statements in a C/C++ program

	Step count
– **Comments**	**0**
– **Declarative statements**	**0**
– **Expressions and assignment statements**	**1**
– **Iteration statements**	**N**
– **Switch statement**	**N**
– **If-else statement**	**N**
– **Function invocation**	**1 or N**
– **Memory management statements**	**1 or N**
– **Function statements**	**0**
– **Jump statements**	**1 or N**

Time Complexity Iterative Example

```
float sum (float *a, const int n)
{
    float s = 0;
    for (int i = 0; i < n; i++)
        s += a[i];
    return;
}
```

Source

Step Count of Iterative Example

```
float sum (float *a, const int n)
{
    float s = 0;
    count++;                    // count is global
    for (int i = 0; i < n; i++)
    {
        count++;    // for for
        s += a[i];
        count++;    // for assignment
    }
    count++;                    // for last time of for
    count++;                    // for return
    return;
}
```

Add Count

Step Count of Iterative Example (Simplified)

```
void sum (float *a, const int n)
{
    for (int i = 0; i < n; i++)
        count += 2;
    count +=3;
}
```

If initially count = 0, then the total of steps is 2n + 3.

Time Complexity of Recursive Example

```
float rsum (float *a, const int n)
{
    if (n <= 0) return 0;
    else return (rsum(a, n–1) + a[n-1]);
}
```

Source

Step Count of Recursive Example

```
float rsum (float *a, const int n)
{
    count++;                    // for if conditional
    if (n <= 0) {
        count++;    // for return
        return 0;
    }
    else {
        count++;    // for return
        return (rsum(a, n–1) + a[n-1]);
    }
}
```

Add Count

Assume $t_{rsum}(0) = 2$

$t_{rsum}(n) = 2 + t_{rsum}(n-1)$

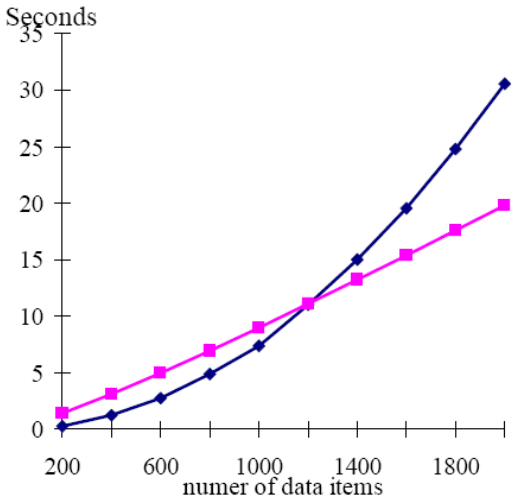
$= 2 + 2 + t_{rsum}(n-2)$

$= 2*2 + t_{rsum}(n-2)$

$= 2n + t_{rsum}(0)$

$= 2n + 2$

An Example of Time Complexity Comparison



VAX8800, Two CPUs Supercomputer. Each CPU operates at 22.22MHz, 45ns cycle time.

Insertion sort by VAX8800

Quicksort by PC/XT

IBM PC/XT, Single Intel 80286 CPU. It operates at 6MHz, 166.65ns cycle time.

Asymptotic Notation

- Determining step counts help us to compare the time complexities of two programs and to predict the growth in run time as the instance characteristics change.
- But determining exact step counts could be very difficult. Since the notion of a step count is itself inexact, it may be not worth the effort to compute the exact step counts.
- Definition [Big "oh"]: $f(n) = O(g(n))$ iff there exist positive constants c and n_0 such that $f(n) \leq cg(n)$ for all n, $n \geq n_0$

TSEG Tamkang Universty Software Engineering Group 淡江軟體工程實驗室 http://www.tkse.tku.edu.tw/

Examples of Asymptotic Notation

- $3n + 2 = O(n)$

  $3n + 2 \leq 4n$ for all $n \geq 2$
- $100n + 6 = O(n)$

  $100n + 6 \leq 101n$ for all $n \geq 6$
- $10n^2 + 4n + 2 = O(n^2)$

  $10n^2 + 4n + 2 \leq 11n^2$ for all $n \geq 5$

http://mail.tku.edu.tw/inhon

Asymptotic Notation (Cont.)

Theorem 1.2: If $f(n) = a_m n^m + \ldots + a_1 n + a_0$, then $f(n) = O(n^m)$.

Proof:

$$f(n) \le \sum_{i=0}^{m} |a_i| n^i$$

$$\le n^m \sum_{0}^{m} |a_i| n^{i-m} \qquad \text{for } n \ge 1$$

$$\le n^m \sum_{0}^{m} |a_i|$$

So, $f(n) = O(n^m)$

Asymptotic Notation (Cont.)

- Definition: [Omega] $f(n) = \Omega(g(n))$ iff there exist positive constants c and n_0 such that $f(n) \ge cg(n)$ for all n, $n \ge n_0$.
- Example:
  - $3n + 2 = \Omega(n)$
  - $100n + 6 = \Omega(n)$
  - $10n^2 + 4n + 2 = \Omega(n^2)$

Asymptotic Notation (Cont.)

- Definition: $f(n) = \Theta(g(n))$ iff there exist positive constants c_1, c_2, and n_0 such that $c_1 g(n) \le f(n) \le c_2 g(n)$ for all n, $n \ge n_0$.

Asymptotic Notation (Cont.)

- Theorem 1.3: If $f(n) = a_m n^m + \ldots + a_1 n + a_0$ and $a_m > 0$, then $f(n) = \Omega(n^m)$.

- Theorem 1.4: If $f(n) = a_m n^m + \ldots + a_1 n + a_0$ and $a_m > 0$, then $f(n) = \Theta(n^m)$.

Practical Complexities

- If a program P has complexities $\Theta(n)$ and program Q has complexities $\Theta(n^2)$, then, in general, we can assume program P is faster than Q for a sufficient large n.
- However, caution needs to be used on the assertion of "sufficiently large".

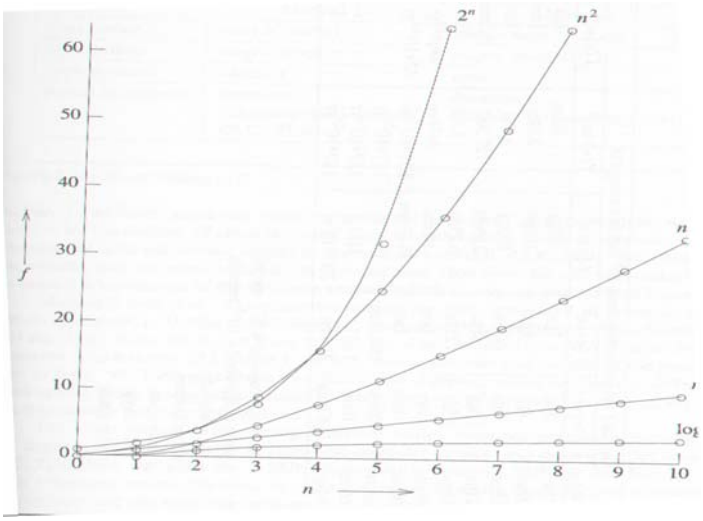
TSEG Tamkang Universty Software Engineering Group 淡江軟體工程實驗室 http://www.tkse.tku.edu.tw/

Function Values

log n	n	n log n	n^2	n^3	2^n
0	1	0	1	1	2
1	2	2	4	8	4
2	4	8	16	64	16
3	8	24	64	512	256
4	16	64	256	4096	65536
5	32	160	1024	32768	4294967296

http://mail.tku.edu.tw/inhon

Plot of Function Values



生活小格言

事事如意

常想一二

不思八九

取自張忠謀隨筆

Linked List

Bridge In: Review of Sequential Representations

- Previously introduced data structures, including Array, queue, and stack, they all have the property that successive nodes of data object were stored a fixed distance apart.
- The drawback of using Array for ordered lists is that operations, such as insertion and deletion, become expensive.
- Also sequential representation tends to have less space efficiency when handling multiple various sizes of ordered lists.

Bridge In: Review of Sequential Representations

- Question 1:

19
34
68
87

How: Add 52 with sequentially

19
34
52
68
87

- Question 2:

19
34
52
68
87

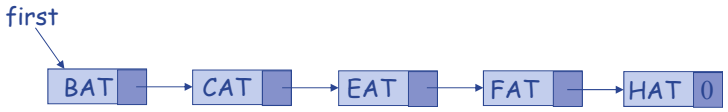
How: Del 34 with sequentially

19
52
68
87

Linked List

- A better solutions to resolve the aforementioned issues of sequential representations is **linked lists**.
- Elements in a linked list are not stored in sequential in memory. Instead, they are stored all over the memory. They form a list by recording the address of next element for each element in the list. Therefore, the list is linked together.
- A linked list has a **head pointer** that points to the first element of the list.
- By following the links, you can **traverse** the linked list and visit each element in the list one by one.

Linked List-Diagram Representation



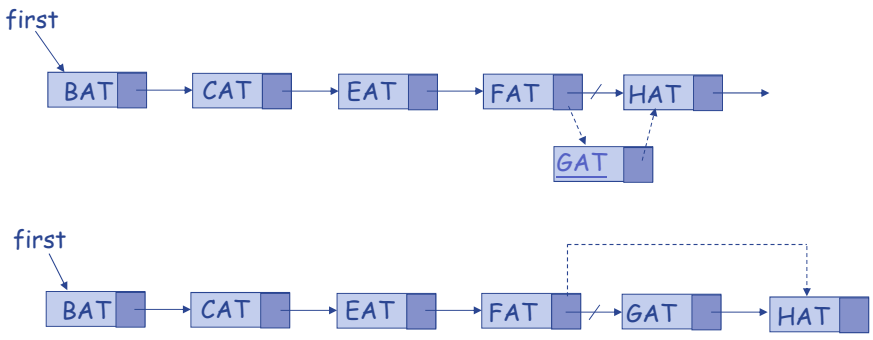
Linked List Insertion

- To **insert** an element into the three letter linked list:
  - Get a node that is currently unused; let its address be x.
  - Set the data field of this node to GAT, for example.
  - Set the link field of x to point to the node after FAT, which contains HAT.
  - Set the link field of the node containing FAT to x.

Linked List Deletion

- To **delete** an element from the three letter linked list:
  - Find the element that immediately proceeds GAT, for example
  - Which is FAT; let its address be x.
  - Set the link field of x to point to the node after GAT, which contains HAT.
  - There is no need to move the data around. Even though the link field of GAT still contains a pointer to HAT, GAT is no longer in the list

Linked List Insertion And Deletion



Node Representation in C++

Sources: Fundamentals of Data Structures in C++ Ellis Horowitz

```
template <class T>
class ChainNode
{
  private:
   T data;
   ChainNode<T> *link;

   // constructors come here
};
```

Node Representation in Java

Sources: Data Structures & Algorithms in Java Michael Goodrich

```
public class Node {
  // Instance variables:
  private Object element;
  private Node next;
  /** Creates a node with null references to its element and next node. */
  public Node() {
    this(null, null);
  }
  /** Creates a node with the given element and next node. */
  public Node(Object e, Node n) {
      element = e;
      next = n;
  }
  // Accessor methods:
  public Object getElement() {
    return element;
  }
  public Node getNext() {
    return next;
  }
  // Modifier methods:
  public void setElement(Object newElem) {
      element = newElem;
  }
  public void setNext(Node newNext) {
      next = newNext;
  }
}
```

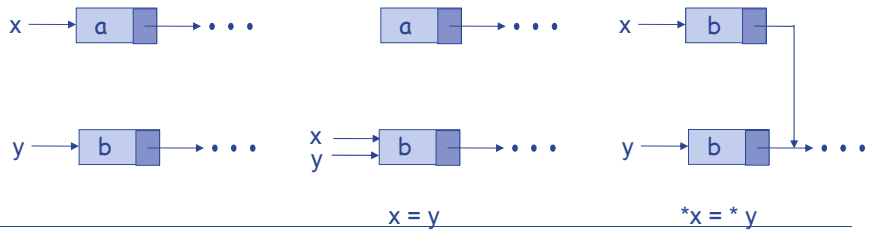
Node Representation in C

```
#include<stdlib.h>
#include<stdio.h>
struct list_el {
   int val;
   struct list_el * next;
};
typedef struct list_el item;
void main() {
   item * curr, * head;
   int i;
   head = NULL;
   for(i=1;i<=10;i++) {
      curr = (item *)malloc(sizeof(item));
      curr->val = i;
      curr->next = head;
      head = curr;
   }
   curr = head;
   while(curr) {
      printf("%d\n", curr->val);
      curr = curr->next ;
   }
}
```

Sources:
http://www.macs.hw.ac.uk/~rjp/Coursewww/Cwww/linklist.html

Pointer Manipulation in C++

- Addition of integers to pointer variable is permitted in C++ but sometimes it has no logical meaning.
- Two pointer variables of the same type can be compared.
  - x == y, x != y, x == 0



List Manipulation Operations

- The basic linked list operations:
  - add: Add an item (or array of items) to the end of the list. Adding an item to this list in this simple implementation is O(1) in Big O notation because we keep a reference to the last node in the list and simply append to it.
  - remove: Remove an item (or array of items) from the list. Items are located by equality. Complexity for removal is O(n) because the the list is iterated from the first element to the last, testing each item for equality.
  - indexOf: Indicates the position of an item in the list, returning -1 if not found. Again, O(n) because of the list being iterated.
  - size: Shows the number of items in the list. O(n) because of iteration.
  - elementAt: Gives the position in the list of an element.

Add new node Operation

```
public void add(Object inputData) {
  ListNode node = new ListNode(inputData);

  /* Make sure we cater for the case where the list is empty */
  if (this.firstNode.getData() == null) {
    this.firstNode = node;
    this.lastNode = node;
  }
  else {
    this.lastNode.setNext(node);
    this.lastNode = node;
  }

  this.size++;
}

public void add(Object [] inputArray) {
  for (int i = 0; i < inputArray.length; i++) {
      this.add(inputArray[i]);
  }
}
```

Sources: Data Structures & Algorithms in Java
Michael Goodrich

Remove one node Operation (1)

```
public void remove(Object inputData) {
  ListNode currentNode = this.firstNode;

  if (this.size == 0) {
    return;
  }

  boolean wasDeleted = false;

  /* Are we deleting the first node? */
  if (inputData.equals(currentNode.getData())) {

    /* Only one node in list, be careful! */
    if (currentNode.getNext() == null) {
      this.firstNode.setData(null);
      this.firstNode = new ListNode();
      this.lastNode = this.firstNode;
      this.size--;
      return;
    }

    currentNode.setData(null);
    currentNode = currentNode.getNext();
    this.firstNode = currentNode;
    this.size--;
    return;
  }
```

Sources: Data Structures & Algorithms in Java
Michael Goodrich

Remove one node Operation (2)

Sources: Data Structures & Algorithms in Java
Michael Goodrich

```
while (true) {
  /* If end of list, stop */
  if (currentNode == null) {
    wasDeleted = false;
    break;
  }

  /* Check if the data of the next is what we're looking for */
  ListNode nextNode = currentNode.getNext();
  if (nextNode != null) {
    if (inputData.equals(nextNode.getData())) {

      /* Found the right one, loop around the node */
      ListNode nextNextNode = nextNode.getNext();
      currentNode.setNext(nextNextNode);

      nextNode = null;
      wasDeleted = true;
      break;
    }
  }

  currentNode = currentNode.getNext();
}
```

Remove one node Operation (3)

```
  if (wasDeleted) {
    this.size--;
  }
}

public void remove(Object [] inputArray) {
  for (int i = 0; i < inputArray.length; i++) {
    this.remove(inputArray[i]);
  }
}
```

Sources: Data Structures & Algorithms in Java
Michael Goodrich

Helper Operations - 1

```
public int size() {
  return this.size;
}
```

Sources: Data Structures & Algorithms in Java
Michael Goodrich

```
public Object elementAt(int inputPosition) {

  if (inputPosition >= this.size || inputPosition < 0) {
    return null;
  }

  ListNode currentNode = this.firstNode;

  for (int position = 0; position < inputPosition ; position++) {
      currentNode = currentNode.getNext();
  }

  return currentNode.getData();
}
```

Helper Operations - 2

```
public int indexOf(Object inputData) {
  ListNode currentNode = this.firstNode;
  int position = 0;
  boolean found = false;

  for (; ; position++) {
    if (currentNode == null) {
      break;
    }

    if (inputData.equals(currentNode.getData())) {
      found = true;
      break;
    }

    currentNode = currentNode.getNext();
  }

  if (!found) {
    position = -1;
  }

  return position;
}
```

Sources: Data Structures & Algorithms in Java
Michael Goodrich

Define A Linked List Template

- Reusability of Linked List
- A linked list is a container class, so its implementation is a good template candidate.
- Member functions of a linked list should be general that can be applied to all types of objects.

Define A Linked List Template

Sources: Fundamentals of Data Structures in C++ Ellis Horowitz

```
template < class T > class Chain;  // 前向宣告
template < class T >class ChainNode {
    friend class Chain <T>;
    private:
        T data;
        ChainNode<T>* link;
};
template <class T>class Chain {
    public:
        Chain( ) {   first = 0;  } // 建構子將first初始化成0
        // 鏈的處理運算函式s    .              .
    private:
        ChainNode<T> * first;
}
```


Template of Linked Lists (Chain)

Sources: Fundamentals of Data Structures in C++ Ellis Horowitz

```
template<class T>
class Chain
{
  public:
    Chain() {first = 0;}
      // constructor, empty chain
    ~Chain(); // destructor
    bool IsEmpty() const {return first == 0;}
    // other methods defined here
  private:
    ChainNode<T>* first;
};
```

Destructor

Sources: Fundamentals of Data Structures in C++ Ellis Horowitz

```
template<class T>
chain<T>::~chain()
{// Chain destructor. Delete all nodes
 // in chain.
   while (first != NULL)
   {// delete first
      ChainNode<T>* next = first->link;
      delete first;
      first = next;
   }
}
```


IndexOf

Sources: Fundamentals of Data Structures in C++ Ellis Horowitz

```
template<class T>
int Chain<T>::IndexOf(const T& theElement) const
{
   // search the chain for theElement
   ChainNode<T>* currentNode = first;
   int index = 0;  // index of currentNode
   while (currentNode != NULL &&
          currentNode->data != theElement)
   {
      // move to next node
      currentNode = currentNode->next;
      index++;
   }
// make sure we found matching element
   if (currentNode == NULL)
      return -1;
   else
      return index;
}
```

Insert An Element

Sources: Fundamentals of Data Structures in C++ Ellis Horowitz

```
template<class T>
void Chain<T>::Insert(int theIndex,
                              const T& theElement)
{
   if (theIndex < 0)
       throw "Bad insert index";

   if (theIndex == 0)
      // insert at front
      first = new chainNode<T>
              (theElement, first);
```


Insert An Element (Cont.)

```
else
   {  // find predecessor of new element
      ChainNode<T>* p = first;
      for (int i = 0; i < theIndex - 1; i++)
      {if (p == 0)
       throw "Bad insert index";
       p = p->next;}
      // insert after p
      p->link = new ChainNode<T>
                    (theElement, p->link);
   }
}
```

Sources: Fundamentals of Data Structures in C++ Ellis Horowitz

Delete An Element

```
template<class T>
void Chain<T>::Delete(int theIndex)
{
   if (first == 0)
   throw "Cannot delete from empty chain";
   ChainNode<T>* deleteNode;
   if (theIndex == 0)
   {// remove first node from chain
      deleteNode = first;
      first = first->link;
   }
```

Sources: Fundamentals of Data Structures in C++ Ellis Horowitz

TSEG Tamkang University Software Engineering Group 淡江軟體工程實驗室 http://www.tkse.tku.edu.tw/

Delete An Element(Cont.)

```
else
{  // use p to get to beforeNode
   ChainNode<T>* p = first;
   for (int i = 0; i < theIndex - 1; i++)
   {if (p == 0)
      throw "Delete element does not exist";
    p = p->next;}
   deleteNode = p->link;
   p->link = p->link->link;
 }
 delete deleteNode;
}
```

Sources: Fundamentals of Data Structures in C++ Ellis Horowitz

http://mail.tku.edu.tw/inhon

Attaching A Node To The End Of A List

```
Template <class Type>
Void List<Type>::Attach(Type k)
{
    ListNode<Type>*newnode = new ListNode<Type>(k);
    if (first == 0) first = last =newnode;
    else {
        last->link = newnode;
        last = newnode;
    }
};
```

Sources: Fundamentals of Data Structures in C++ Ellis Horowitz

Concatenating Two Chains

```
Template <class Type>
void List<Type>:: Concatenate(List<Type> b)
// this = (a1, …, am) and b = (b1, …, bn) m, n ≥ ,
// produces the new chain z = (a1, …, am, b1, bn) in this.
{
    if (!first) { first = b.first; return;}
    if (b.first) {
      for (ListNode<Type> *p = first; p->link; p = p->link); // no body
      p->link = b.first;
    }
}
```

Sources: Fundamentals of Data Structures in C++ Ellis Horowitz

參考學習

- http://www.youtube.com/watch?v=195KUinjBpU
- http://www.youtube.com/watch?v=iR5wyCaIayk

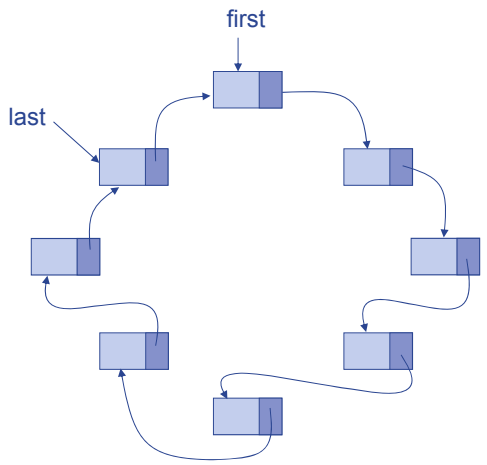
When Not To Reuse A Class

- If efficiency becomes a problem when reuse one class to implement another class.
- If the operations required by the application are complex and specialized, and therefore not offered by the class.

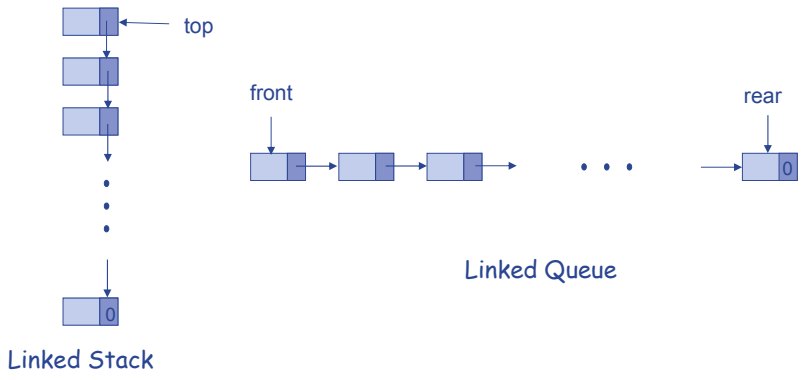
Circular Lists

- By having the link of the last node points to the first node, we have a circular list.
  - Need to make sure when **current** is pointing to the last node by checking for **current->link == first**.
  - Insertion and deletion must make sure that the circular structure is not broken, especially the link between last node and first node.

Diagram of A Circular List



Linked Stacks and Queues



Linked Stacks and Queues (Cont.)

- 以下課程採『翻轉學習(Flipped Learning)與PBL(Problem Based Learning)」方式教學與學習方式進行
- 請同學參讀以下教材與延伸閱讀教材
- 完成任務二之分組作業
- 2014. 11. 14進行上述教學型式

Example of Stack in Linked List

- Class Stack
- Operations on Class Stack
  - Add (Push) function in Stack
  - Delete (Pop) function in Stack
  - Helper functions in Stack

TSEG Tamkang University Software Engineering Group 淡江軟體工程實驗室 http://www.tkse.tku.edu.tw/

Example of Stack in Linked List

- Class Link_Node by C++

```
template <class T> class LinkedStack; //Forward Declaration
template <class T>
class Link_Node {
   friend class LinkedStack <T>;
   private:
      T  data;
      Link_Node <T> *link;
};
```

Sources: Fundamentals of Data Structures in C++ Ellis Horowitz

http://mail.tku.edu.tw/inhon

Example of Stack in Linked List

- Class Stack

```
template <class T>
class LinkedStack {
   public:
      LinkedStack() {top = 0}; //Constructor
      void Push (const T& InputData); //Add new data to stack
      void Pop(const T& GetData);  //Get out the data from stack
      void First(const T& FirstData); //Find the first data in stack but not take
      ~LinkedStack(); //Release the space of stack
}
   private:
      Link_Node <T> *top;
};
```

Sources: Fundamentals of Data Structures in C++ Ellis Horowitz

Example of Stack in Linked List

```
template <class T>
void LinkedStack <T>::Push(const T& InputData) {
    top = new Link_Node <T>(InputData, top);
}

template <class T>
void LinkStack <T>::Pop(const T&GetData  )
{// 刪除堆疊的頂端節點
    if (IsEmpty()) throw "Stack is empty. Cannot delete.";
    Link_Node <T> *delNode = top;
    top = top→link;  // 移除頂端節點
    GetData = delNode->data;
    delete delNode;  // 釋回此節點
}
```

Sources: Fundamentals of Data Structures in C++ Ellis Horowitz

Example of Stack in Linked List

```
T LinkedStack<T>::Top() const {
// Return top element.
   if (IsEmpty()) cout<<"Stack empty:";
   else return top->data;
}
```

Sources: http://electrofriends.com/source-codes/software-programs/cpp-programs/cpp-data-structure/c-program-to-implement-stack-using-linked-list/

```
void print() const {
// could just be initialized
// the asterisk is commonly
// put next to the type in C++
  node* temp = top;
  while (temp != NULL) {
      cout << temp-> data << ","; temp = temp->next;
  }
}
```

Sources: http://codereview.stackexchange.com/questions/33195/stack-implementation-using-linked-list

TSEG Tamkang University Software Engineering Group 淡江軟體工程實驗室 http://www.tkse.tku.edu.tw/

Example of Stack in Linked List

- Class Link_Node by Java

```
public interface Stack<E> {
 /**
  * Return the number of elements in the stack.
  * @return number of elements in the stack.
  */
  public int size();
 /**
  * Return whether the stack is empty.
  * @return true if the stack is empty, false otherwise.
  */
  public boolean isEmpty();
 /**
  * Inspect the element at the top of the stack.
  * @return top element in the stack.
  * @exception EmptyStackException if the stack is empty.
  */
  public E top()
    throws EmptyStackException;
```

Sources: Data Structures & Algorithms in Java Michael Goodrich

http://mail.tku.edu.tw/inhon

Example of Stack in Linked List

```
/**
 * Insert an element at the top of the stack.
 * @param element to be inserted.
 */
 public void push (E element);
/**
 * Remove the top element from the stack.
 * @return element removed.
 * @exception EmptyStackException if the stack is empty.
 */
 public E pop()
   throws EmptyStackException;
}
//end#fragment Stack
```

Sources: Data Structures & Algorithms in Java
Michael Goodrich

參考學習

- http://electrofriends.com/source-codes/software-programs/cpp-programs/cpp-data-structure/c-program-to-implement-stack-using-linked-list/
- http://codereview.stackexchange.com/questions/33195/stack-implementation-using-linked-list
- http://www.java-tips.org/java-se-tips/java.lang/linked-list-based-stack-implementation.html
- http://www.youtube.com/watch?v=V2Tb_v6vuqg
- http://www.youtube.com/watch?v=MuwxQ2IB8lQ
- http://www.youtube.com/watch?v=_xOvZGvdBfE

Example of Queue in Linked List

- Class Queue
- Operations on Class Queue
  - Add function in Queue
  - Delete function in Queue
  - Helper functions in Queue

Example of Queue in Linked List

```
template <class T>
void LinkedQueue <T>:: Add(const T& e)
{
    if (IsEmpty( )) front = rear = new ChainNode(e,0); //空佇列
    else rear = rear→link = new ChainNode(e,0); // 連接節點並且更新rear
}

template <class T>
void LinkedQueue <T>:: Del(const T& GetData)
{// 刪除佇列的第一個元素
    if (IsEmpty()) throw "Queue is empty. Cannot delete.";
    ChainNode<T> *delNode = front;
    front = front→link;   // 移除鏈的第一個節點
   GetData = delNode → data;
    delete delNode;    // 釋回此節點
}
```

Sources: Fundamentals of Data Structures in C++ Ellis Horowitz

Example of Queue in Linked List

Sources: Data Structures & Algorithms in Java
Michael Goodrich

```
//begin#fragment Queue
public interface Queue<E> {
 /**
  * Returns the number of elements in the queue.
  * @return number of elements in the queue.
  */
  public int size();
 /**
  * Returns whether the queue is empty.
  * @return true if the queue is empty, false otherwise.
  */
  public boolean isEmpty();
 /**
  * Inspects the element at the front of the queue.
  * @return element at the front of the queue.
  * @exception EmptyQueueException if the queue is empty.
  */
```


TSEG Tamkang University Software Engineering Group 淡江軟體工程實驗室 http://www.tkse.tku.edu.tw/

Example of Queue in Linked List

```
public E front() throws EmptyQueueException;
 /**
  * Inserts an element at the rear of the queue.
  * @param element new element to be inserted.
  */
  public void enqueue (E element);
 /**
  * Removes the element at the front of the queue.
  * @return element removed.
  * @exception EmptyQueueException if the queue is empty.
  */
  public E dequeue() throws EmptyQueueException;
}
//end#fragment Queue
```

Sources: Data Structures & Algorithms in Java
Michael Goodrich

http://mail.tku.edu.tw/inhon

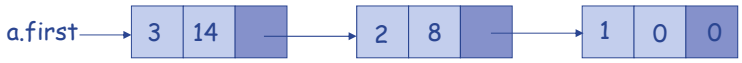
參考學習

- http://www.sanfoundry.com/cpp-program-implement-queue-linked-list/
- http://www.sanfoundry.com/java-program-implement-queue-using-linked-list/
- http://www.youtube.com/watch?v=A5_XdiK4J8A
- http://www.youtube.com/watch?v=PGWZUgzDMYI
- http://www.youtube.com/watch?v=esI7P2wzLGU

Revisit Polynomials

- 以下教材內容僅出現於: **Fundamentals of Data Structures in C++ Ellis Horowitz 一書**
- 提供概念參考

Revisit Polynomials



$$a = 3x^{14} + 2x^{8} + 1$$

$$b = 8x^{14} - 3x^{10} + 10x^{6}$$

Program 4.23 Polynomial Class Definition

```
struct Term
// all members of Terms are public by default
{
    int coef;           // coefficient
    int exp;            // exponent
    void Init(int c, int e) {coef = c; exp = e;};
};

class Polynomial
{
    friend Polynomial operator+(const Polynomial&, const Polynomial&);
    private:
        List<Term> poly;
};
```

Sources: Fundamentals of Data Structures in C++ Ellis Horowitz

Program 4.24 Polynomial Operation--Add

```
Polynomial Polynomial::operator+(const Polynomial& b) const
{// 相加多項式 *this（a）與b並且回傳它們的和
    Term temp;
    Chain <Term>::ChainIterator ai = poly.begin(),
                                bi =b. poly.begin();
    Polynomial c;
    while (ai&&bi) { // 目前的節點不是空的
            if (ai→exp = = bi→exp) {
                    int sum = ai→coef + bi→coef;
                    if (sum) c.poly.InsetBack (temp.Set(sum, ai→exp));
                    ai++; bi++; // 前進至下一個項目
            }
            else if (ai→exp < bi→exp) {
                    c.poly.InsertBack(temp.Set(bi→coef, bi→exp)) ;
                    bi++; // b的下一個項目
            }
            else {
                    c.poly.InsertBack(temp.Set(ai→coef , ai→exp)) ;
                    ai++; // a的下一個項目
            }
}
```

Sources: Fundamentals of Data Structures in C++ Ellis Horowitz

TSEG Tamkang University Software Engineering Group 淡江軟體工程實驗室 http://www.tkse.tku.edu.tw/

Program 4.24 Polynomial Operation--Add

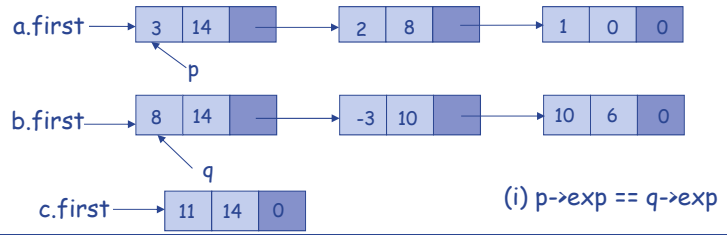
```
while (ai) { // 複製a剩下的部份
        c.poly.InsertBack(temp.Set(ai→coef,ai→exp)) ;
        ai++;
}
while (bi) { // 複製b剩下的部份
        c.poly.InsertBack (temp.Set(bi→coef,bi→exp)) ;
    bi++ ;
}
return c;
}
```

Sources: Fundamentals of Data Structures in C++ Ellis Horowitz

http://mail.tku.edu.tw/inhon

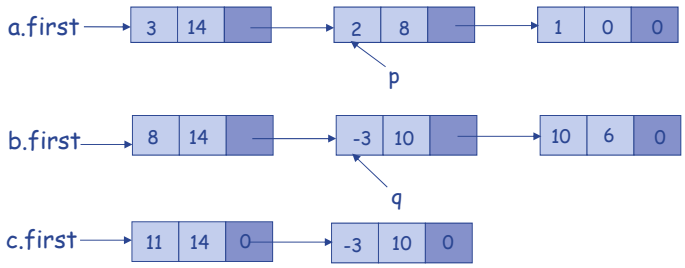
Operating On Polynomials

- With linked lists, it is much easier to perform operations on polynomials such as adding and Subtracting.
  - E.g., adding two polynomials a and b, Program 4.24
  - Time Complexity of Program 4.24 is O(m+n)



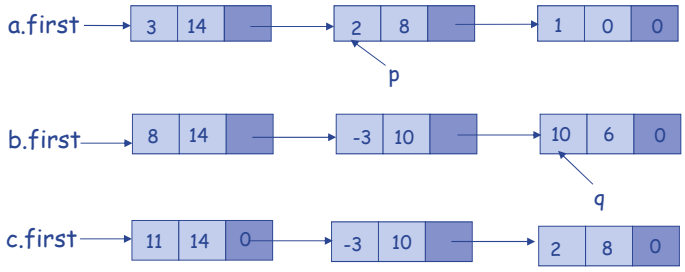
(i) p->exp == q->exp

Operating On Polynomials



(ii) p->exp < q->exp

Operating On Polynomials



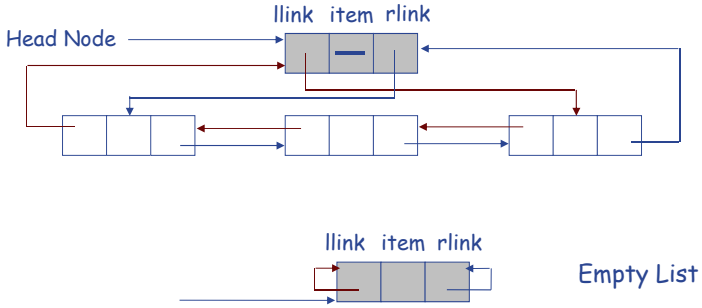
(iii) p->exp > q->exp

Doubly Linked Lists

- The problem of a singly linked list is that supposed we want to find the node precedes a node *ptr*, we have to start from the beginning of the list and search until find the node whose link field contains *ptr*.
- To efficiently delete a node, we need to know its preceding node. Therefore, doubly linked list is useful.
- A node in a doubly linked list has at least three fields: left link field (llink), a data field (item), and a right link field (rlink). A doubly linked list may or may not be circular.

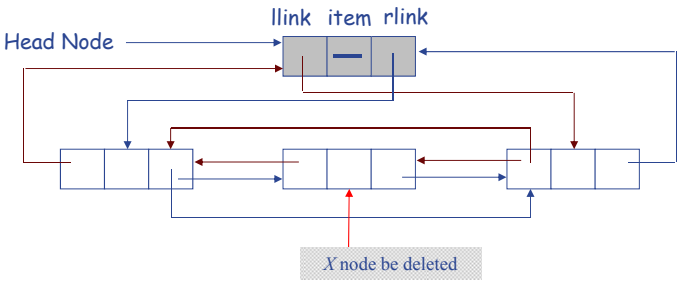
Doubly Linked List

- A head node is also used in a doubly linked list to allow us to implement our operations more easily.
- Figure 4.27 and Figure 4.28 illustrate one example of Doubly linked circular list with Head node and Empty List, respectively.

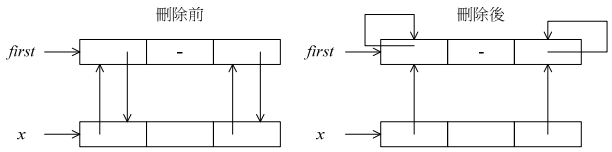


Deletion From A Doubly Linked Circular List

- Program 4.32 shows the class definition of a doubly linked list of integer.
- Program 4.33 and 4.34 present the delete and insert functions of Doubly Linked list Class in specific node pointer.

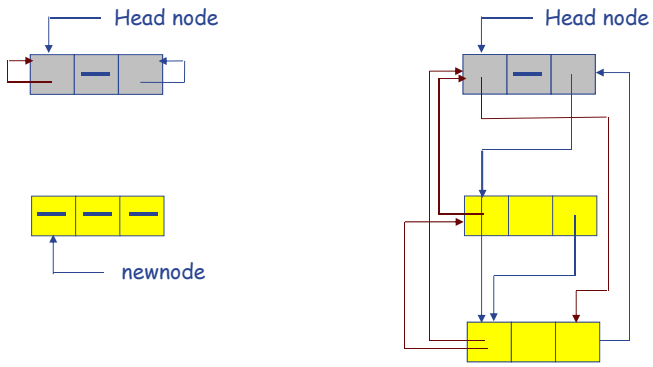


Deletion From A Doubly Linked Circular List



Insertion Into An Empty Doubly Linked Circular List

- Program 4.35 shows the Insertion function of Doubly linked list



After Two steps of Insertion

Class Definition of Double Linked List (1)

```
class DblList; //Forward declaration
class DblListNode {
friend class DblList;
private:
  int data;
  DblListNode *left, *right;
  };
class DblList {
public:
 // 串列處理運算
  .
private:
  DblListNode *first;  // 指向標頭節點
};
```

Sources: Fundamentals of Data Structures in C++ Ellis Horowitz

Insert and Delete Functions of DblList (2)

```
void DblList :: Delete(DblListNode *x)
{
   if (x = = first) throw "Deletion of header node not permited"; //first 指向一個無Data之Head Node
   else {
      x→left→right = x→right;
      x→right→left = x→left;
   }
   delete x;
}
void DblList :: Insert(DblListNode *p, DblListNode *x)
{ // 插入節點p到節點x的右邊
   p→left = x; p→right = x→right;
   x→right→left = p; x→right = p;
}
```

Sources: Fundamentals of Data Structures in C++ Ellis Horowitz

Class Definition of Double Linked List (1)

```
//begin#fragment DLNode
public class DLNode<E> {
  private E element;
  private DLNode<E> next, prev;
  DLNode() { this(null, null, null); }
  DLNode(E e, DLNode<E> p, DLNode<E> n) {
    element = e;
    next = n;
    prev = p;
  }
  public void setElement(E newElem) { element = newElem; }
  public void setNext(DLNode<E> newNext) { next = newNext; }
  public void setPrev(DLNode<E> newPrev) { prev = newPrev; }
  public E getElement() { return element; }
  public DLNode<E> getNext() { return next; }
  public DLNode<E> getPrev() { return prev; }
}
//end#fragment DLNode
```

Sources: Data Structures & Algorithms in Java
Michael Goodrich

Class Definition of Double Linked List (2)

```
//begin#fragment DNode
public class DNode<E> implements Position<E> {
  private DNode<E> prev, next;          // References to the nodes before and after
  private E element; // Element stored in this position
  /** Constructor */
  public DNode(DNode<E> newPrev, DNode<E> newNext, E elem) {
    prev = newPrev;
    next = newNext;
    element = elem;
  }
  // Method from interface Position
  public E element() throws InvalidPositionException {
    if ((prev == null) && (next == null))
      throw new InvalidPositionException("Position is not in a list!");
    return element;
  }
```

Sources: Data Structures & Algorithms in Java
Michael Goodrich

Class Definition of Double Linked List (3)

```
// Accessor methods
  public DNode<E> getNext() { return next; }
  public DNode<E> getPrev() { return prev; }
  // Update methods
  public void setNext(DNode<E> newNext) { next = newNext; }
  public void setPrev(DNode<E> newPrev) { prev = newPrev; }
  public void setElement(E newElement) { element = newElement; }
}
//end#fragment DNode
```

Sources: Data Structures & Algorithms in Java
Michael Goodrich

Class Definition of Double Linked List (1)

```
class Node {
   Node prev;
   Node next;
   int data;

   public Node(int d) { data = d; prev = null; next = null; } // Constructor
}

class LinkedList {
   Node head;
   public LinkedList() { head = null; }  //Constructor
```

Sources:
http://codereview.stackexchange.com/questions/33040/java-doubly-linked-list

Class Definition of Double Linked List (2)

```
public void insert(int d) {
  if (head == null) { head = new Node(d);
      return;
  }
  if (head.data > d) {
      Node holder = head;
      Node newNode = new Node(d);
      head = newNode;
      head.next = holder;
      holder.prev = newNode;
  return;
}
```

Sources:
http://codereview.stackexchange.com/questions/33040/java-doubly-linked-list

Class Definition of Double Linked List (3)

```
  Node tmpNode = head;
  while (tmpNode.next != null && tmpNode.next.data < d) {
    tmpNode = tmpNode.next;
  }
  Node prevTmp = tmpNode;
  Node insertedNode = new Node(d);
  if (tmpNode.next != null) {
    Node nextTmp = tmpNode.next;
    insertedNode.next = nextTmp;
    nextTmp.prev = insertedNode;
  }
  prevTmp.next = insertedNode;
  insertedNode.prev = prevTmp;
}
```

Sources:
http://codereview.stackexchange.com/questions/33040/java-doubly-linked-list

Class Definition of Double Linked List (4)

```
public void delete(int d) {
  if (head == null) {
    System.out.println("The list is empty.");
    return;
  }
  if (head.data == d) {
    head = head.next;
    if (head != null) {
      head.prev = null;
    }
    return;
  }
  Node tmpNode = head;
  while (tmpNode != null && tmpNode.data != d) {
    tmpNode = tmpNode.next;
  }
```

Sources:
http://codereview.stackexchange.com/questions/33040/java-doubly-linked-list

Class Definition of Double Linked List (5)

```
  if (tmpNode == null) {
    System.out.println("That node does not exist in the list");
    return;
  }
  if (tmpNode.data == d) {
     tmpNode.prev.next = tmpNode.next;
     if (tmpNode.next != null) {
        tmpNode.next.prev = tmpNode.prev;
  }
 }
}
```

Sources:
http://codereview.stackexchange.com/questions/33040/java-doubly-linked-list

參考學習

- http://www.youtube.com/watch?v=JdQeNxWCguQ
- http://www.youtube.com/watch?v=pBrz9HmjFOs
- http://www.youtube.com/watch?v=k0pjD12bzP0
- http://www.youtube.com/watch?v=VOQNf1VxU3Q
- http://www.youtube.com/watch?v=MZmmSbLJsB4
- http://algs4.cs.princeton.edu/13stacks/DoublyLinkedList.java.html
- http://codereview.stackexchange.com/questions/33040/java-doubly-linked-list

Generalized Lists

- 以下教材內容僅出現於: **Fundamentals of Data Structures in C++ Ellis Horowitz 一書**
- 提供概念參考

Generalized Lists

- Definition: A generalized list, A, is a finite sequence of $n \geq 0$ elements, $\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_{n-1}$, where α_i, is either an atom or a list. The elements $\alpha_i, 0 \leq i \leq n-1$, that are not atoms are said to be the sublists of A.
- A list A is written as $A = (\alpha_0, \ldots, \alpha_{n-1})$, and the length of the list is n.
- Conventionally, a capital letter is used to represent a list and a lower case letter is to represent an atom.
- The α_0 is the head of list A and the rest $(\alpha_1, \ldots, \alpha_{n-1})$ is the tail of list A.

Generalized List Examples

- $D = (\ )$: the null, or empty, list; its length is zero.
- $A = (a, (b, c))$: a list of length of two; its first element is the atom a, and its second element is the linear list (b, c).
- $B = (A, A, (\ ))$: A list of length of three whose first two elements are the list A, and the third element is the null list D.
- $C = (a, C)$: is a recursive list of length two; C corresponds to the infinite list $C = (a, (a, (a, \ldots)))$.

Generalized Lists

- head(A) = 'a' and tail(A) = (b, c), head(tail(A)) = 'b' and tail(tail(A)) = 'c' .
- Two important consequences of the *generalized list* definition
  - Lists may be shared by other lists, in example B
  - Lists may be recursive, in example C

Generalized List Application Example

$$p(x,y,z) = x^{10}y^3z^2 + 2x^8y^3z^2 + 3x^8y^2z^2 + x^4y^4z + 6x^3y^4z + 2yz$$

- Consider the polynomial *P(x, y, z)* with various variables. It is obvious the sequential representation is not suitable to this.
- Initial idea is

Coef	*ExpX*	*ExpY*	*ExpZ*	*Link*

- What if a linear list is used?
  - The size of the node will vary in size, causing problems in storage management.
- Let's try the generalized list.

Generalized List Application Example

- *P(x, y, z)* can be rewritten as follows:

$$((x^{10}+2x^8)y^3+3x^8y^2)z^2+((x^4+6x^3)y^4+2y)z$$

- The above can be written as $Cz^2 + Dz$. Both *C* and *D* are polynomials themselves but with variables *x* and *y* only.
- If we look at polynomial C only, it is actually of the form $Ey^3 + Fy^2$, where *E* and *F* are polynomial of *x* only.
- Continuing this way, every polynomial consists of a variable plus coefficient-exponent pairs. Each coefficient is itself a polynomial.

PolyNode Class in C++

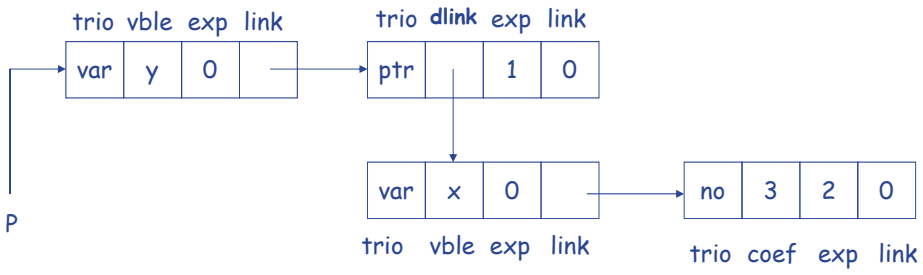
```
enum Triple{ var, ptr, no };
class PolyNode
{
    PolyNode *link;
    int exp;
    Triple trio;
    union {
        char vble;
        PolyNode *dlink;
        int coef;
    };
};
```

Generalized List Node for Polynomial

PolyNode in C++ (Cont.)

- trio == var: the node is a head node.
  - vble indicates the name of the variable. Or it is an integer point to the variable in a variable table.
  - exp is set to 0.
- trio == ptr: coefficient itself is a list and is pointed by the field dlink. exp is the exponent of the variable on which the list is based on.
- trio == no, coefficient is an integer and is stored in coef. exp is the exponent of the variable on which the list is based on.

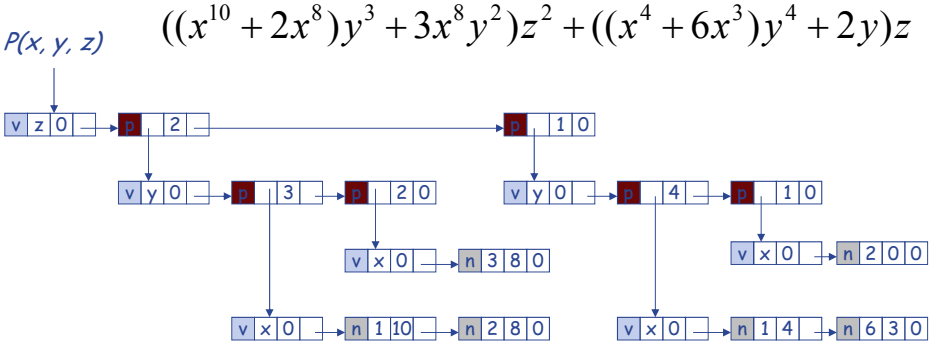
Representing $3x^2y$



Using PolyNode to represent the $3x^2y$

Representation of *P(x, y, z)*

$$((x^{10}+2x^8)y^3+3x^8y^2)z^2+((x^4+6x^3)y^4+2y)z$$



Using PolyNode to represent P(*x*, *y*, *z*) included trio field

Tree

Bridge-In: Why to use Tree structure

- Why do we need to apply Tree structure in Computer Structure
  - In computer science, a tree is an abstract model of a hierarchical structure
- Applications:
  - Organization charts
  - File systems
  - Programming environments

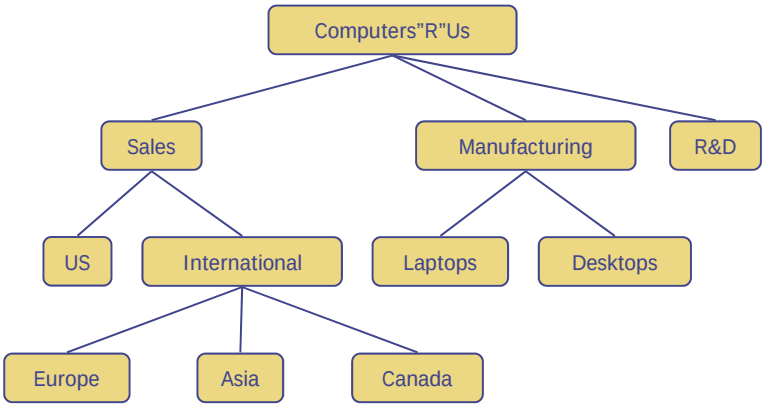
TSEG Tamkang Universty Software Engineering Group 淡江軟體工程實驗室 http://www.tkse.tku.edu.tw/

Bridge-In: What is a Tree structure

- What is a Tree structure in Computer Structure
  - A tree consists of nodes with a parent-child relation
- For examples

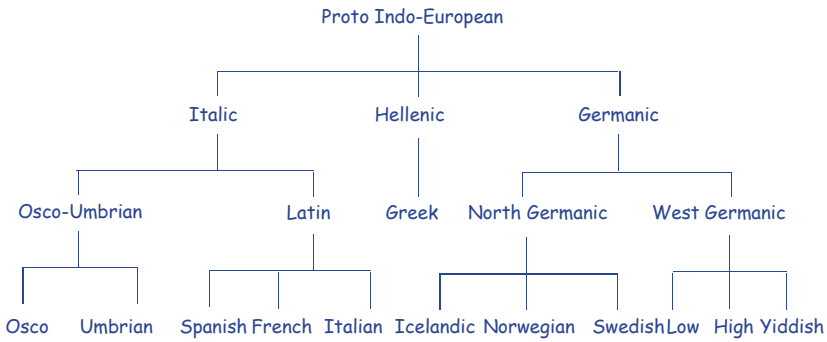
http://inhon.tkse.tku.edu.tw

Organization Chart



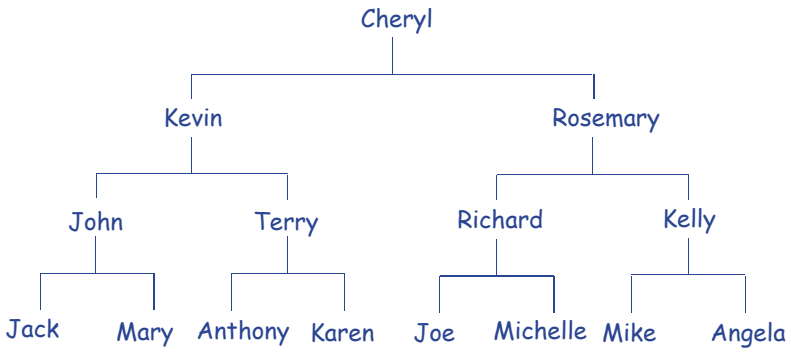
General Tree

Lineal Genealogical Chart



General Tree

Pedigree Genealogical Chart



Binary Tree

Trees

- Definition: A tree is a finite set of one or more nodes such that:
  - There is a specially designated node called the root.
  - The remaining nodes are partitioned into $n \geq 0$ disjoint sets $T_1, \ldots, T_n$, where each of these sets is a tree. We call $T_1, \ldots, T_n$ the subtrees of the root.

Tree Terminology

- Normally we draw a tree with the **root** at the top.
- The degree of a node is the number of **subtree**s of the **node**.
- The degree of a tree is the maximum **degree** of the nodes in the tree.
- A node with degree zero is a **leaf** or **terminal** node.
- A node that has subtrees is the **parent** of the roots of the subtrees, and the roots of the subtrees are the **children** of the node.
- Children of the same parents are called **siblings**.

TSEG Tamkang Universty Software Engineering Group 淡江軟體工程實驗室 http://www.tkse.tku.edu.tw/

Tree Terminology (Cont.)

- The **ancestors** of a node are all the nodes along the path from the root to the node.
- The **descendants** of a node are all the nodes that are in its subtrees.
- Assume the root is at **level** 1, then the level of a node is the level of the node's parent plus one.
- The **height** or the **depth** of a tree is the maximum level of any node in the tree.

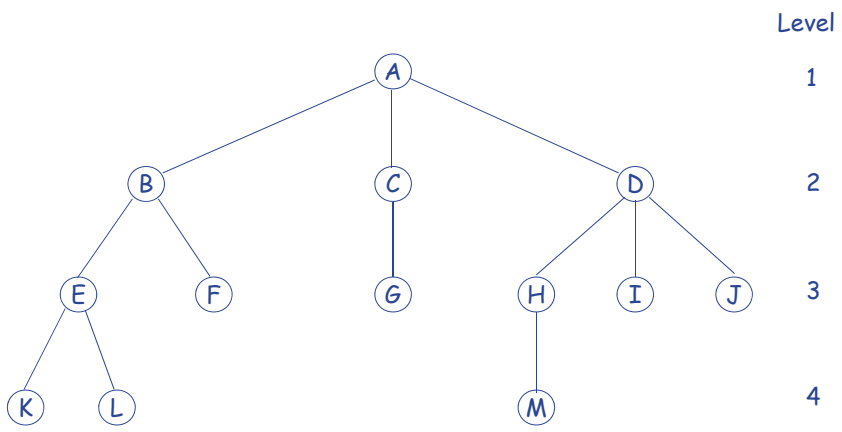
http://inhon.tkse.tku.edu.tw

Tree ADT

- We use positions to abstract nodes
- Generic methods:
  - integer size()
  - boolean isEmpty()
  - Iterator iterator()
  - Iterable positions()
- Accessor methods:
  - position root()
  - position parent(p)
  - Iterable children(p)
- Query methods:
  - boolean isInternal(p)
  - boolean isExternal(p)
  - boolean isRoot(p)
- Update method:
  - element replace (p, o)
- Additional update methods may be defined by data structures implementing the Tree ADT

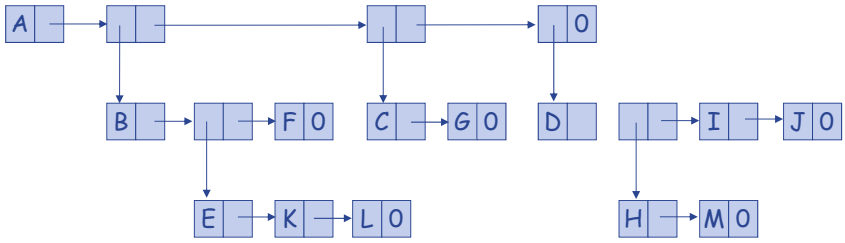
TSEG Tamkang University Software Engineering Group 淡江軟體工程實驗室 http://www.tkse.tku.edu.tw/

A Sample Tree



http://inhon.tkse.tku.edu.tw

List Representation of Trees



Possible Node Structure For A Tree of Degree

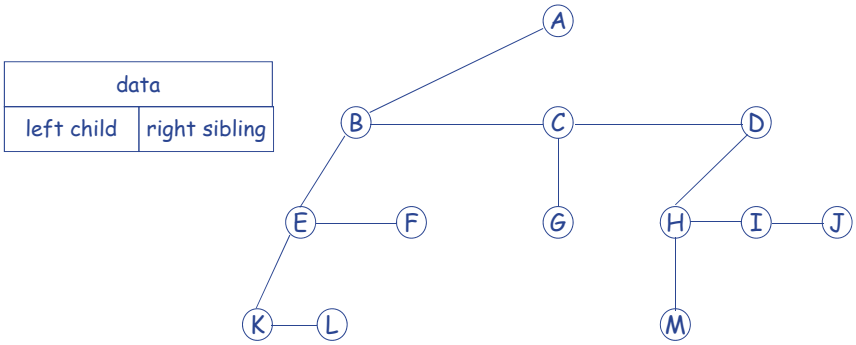
- Lemma 5.1: If T is a k-ary tree (i.e., a tree of degree k) with n nodes, each having a fixed size as in Figure 5.4, then n(k-1) + 1 of the nk child fileds are 0, n ≥ 1.

Data	Child 1	Child 2	Child 3	Child 4	…	Child k

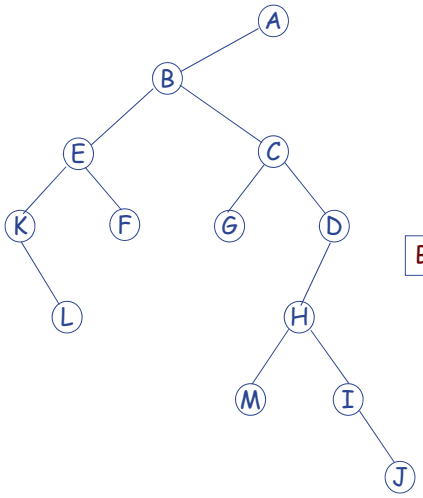
Wasting memory!

Representation of Trees

- Left Child-Right Sibling Representation
  - Each node has two links (or pointers).
  - Each node only has one leftmost child and one closest sibling.

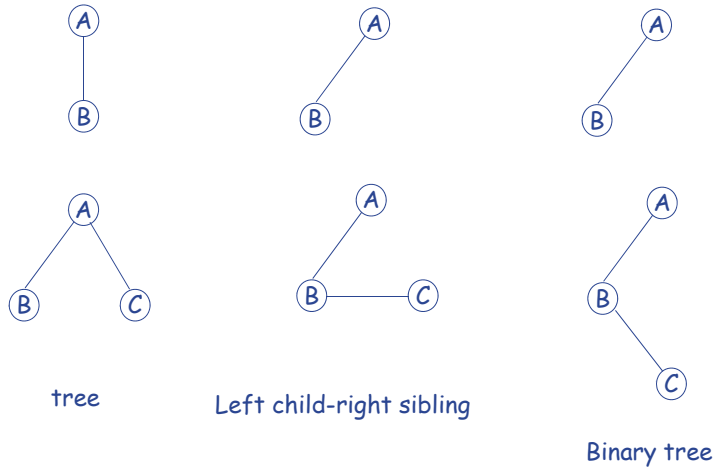


Tree Representation- Degree Two



Binary Tree!

Tree Representations



tree

Left child-right sibling

Binary tree

Forests

- Definition: A forest is a set of $n \geq 0$ disjoint trees.
- When we remove a root from a tree, we'll get a forest. E.g., Removing the root of a binary tree will get a forest of two trees.

Transforming A Forest Into A Binary Tree

- Definition: If $T_1, \ldots, T_n$ is a forest of trees, then the binary tree corresponding to this forest, denoted by $B(T_1, \ldots, T_n)$,
  - is empty if $n = 0$
  - has root equal to root (T_1); has left subtree equal to $B(T_{11}, T_{12}, \ldots, T_{1m})$, where $T_{11}, T_{12}, \ldots, T_{1m}$ are the subtrees of root (T_1); and has right subtree $B(T_2, \ldots, T_n)$.

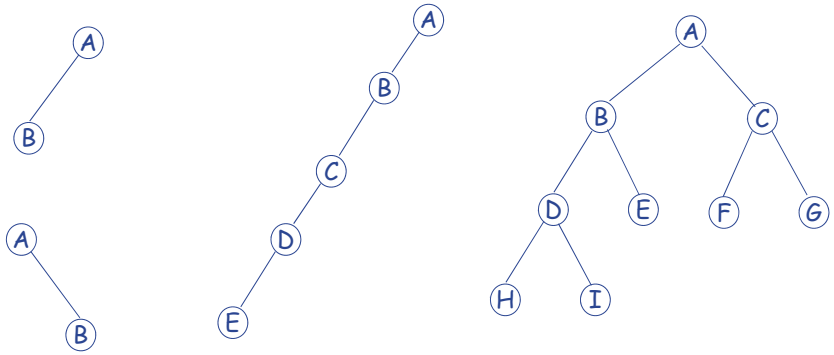
TSEG Tamkang University Software Engineering Group 淡江軟體工程實驗室 http://www.tkse.tku.edu.tw/

Binary Tree

- Definition: A binary tree is a finite set of nodes that is either empty or consists of a root and two disjoint binary trees called the left subtree and the right subtree.
- Recall the Definition of Tree, Two major differences between Tree and Binary tree
  - There is no tree with zero nodes. But there is an empty binary tree.
  - Binary tree distinguishes between the order of the children while in a tree we do not.

http://inhon.tkse.tku.edu.tw

Binary Tree Examples



Two different Binary tree

Skewed Binary tree

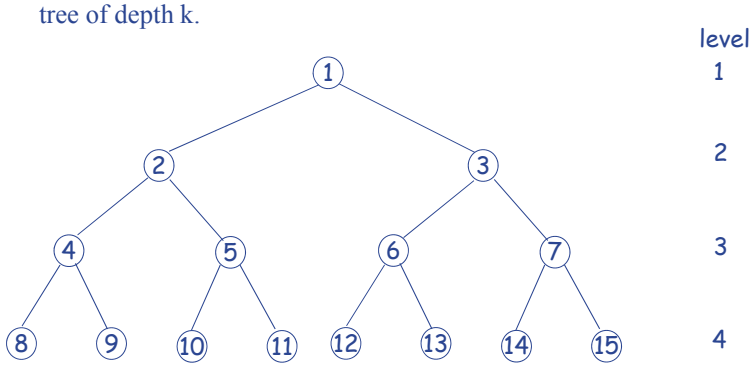
Complete Binary tree

The Properties of Binary Trees

- **Lemma 5.2** [Maximum number of nodes]
  1) The maximum number of nodes on level i of a binary tree is 2^{i-1}, $i \geq 1$.
  2) The maximum number of nodes in a binary tree of depth k is $2^k - 1$, $k \geq 1$.
- **Lemma 5.3** [Relation between number of leaf nodes and nodes of degree 2]: For any non-empty binary tree, T, if n_0 is the number of leaf nodes and n_2 the number of nodes of degree 2, then $n_0 = n_2 + 1$.
- **Definition**: A **full binary tree** of depth k is a binary tree of depth k having $2^k - 1$ nodes, $k \geq 0$.

Complete Binary Tree Definition

- **Definition**: A binary tree with n nodes and depth k is ***complete*** iff its nodes correspond to the nodes numbered from 1 to n in the full binary tree of depth k.



Array Representation of A Complete Binary Tree

- Lemma 5.4: If a complete binary tree with n nodes is represented sequentially, then for any node with index i, $1 \leq i \leq n$, we have:
  - parent(i) is at if $i \neq 1$. If $i = 1$, i is at the root and has no parent.
  - left_child(i) is at $2i$ if $2i \leq n$. If 2i > n, then i has no left child.
  - right_child(i) is at $2i + 1$ if $2i + 1 \leq n$. If $2i + 1 > n$, then i has no right child.
- Position zero of the array is not used.

Proof of Lemma 5.4 (2)

- Assume that for all j, $1 \le j \le i$, left_child(j) is at $2j$. Then two nodes immediately preceding left_child(i + 1) are the right and left children of i. The left child is at $2i$. Hence, the left child of $i + 1$ is at $2i + 2 = 2(i + 1)$ unless $2(i + 1) > n$, in which case $i + 1$ has no left child.

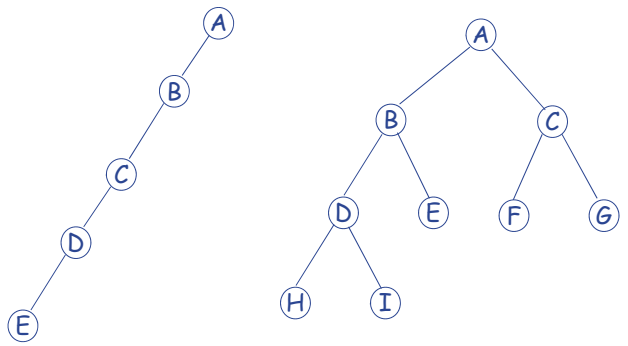
Array Representation of Binary Trees

[1]	A
[2]	B
[3]	—
[4]	C
[5]	—
[6]	—
[7]	—
[8]	D
[9]	—
⋮	⋮
[16]	E

[1]	A
[2]	B
[3]	C
[4]	D
[5]	E
[6]	F
[7]	G
[8]	H
[9]	I

Array representation of the binary tress Skewed and Complete Binary trees

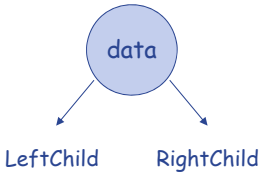
Array Representation of Binary Trees
- Mapping to original structure of Binary Tree



Linked Representation

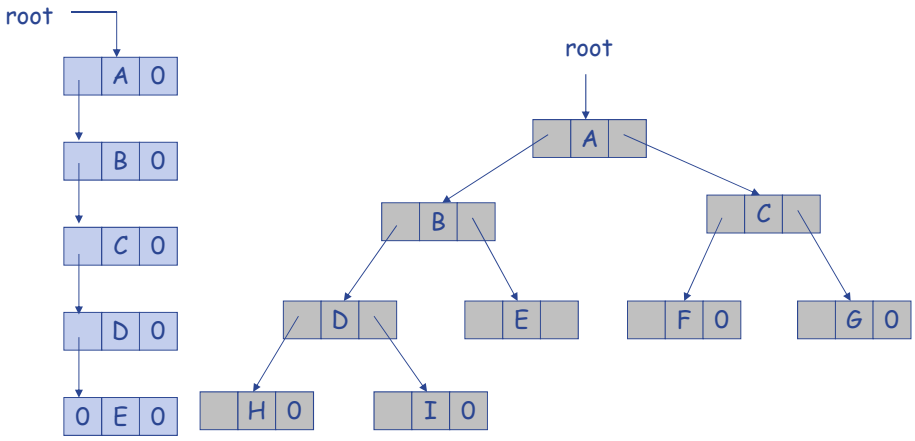
```
class Tree;
class TreeNode {
friend class Tree;
private:
    TreeNode *LeftChild;
    char data;
    TreeNode *RightChild;
};

class Tree {
public:
// Tree operations
.
private:
    TreeNode *root;
};
```

Node Representation



LeftChild	data	RightChild

Linked List Representation For The Binary Trees

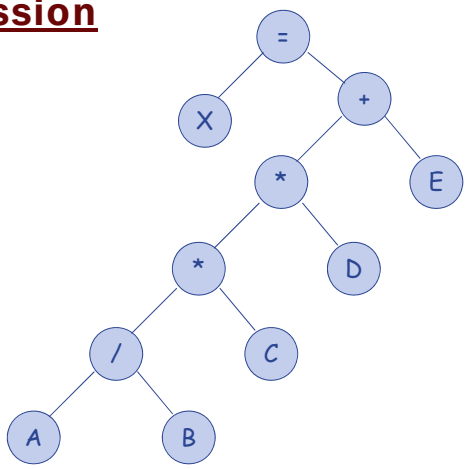


Linked representation of the binary tress of Skewed and Complete Binary Trees

Tree Traversal

- When visiting each node of a tree exactly once, this produces a linear order for the node of a tree.
- There are 3 traversals if we adopt the convention that we traverse left before right: LVR (inorder), LRV (postorder), and VLR (preorder).
- When implementing the traversal, a recursion is perfect for the task.

Binary Tree With Arithmetic Expression



Binary Tree with arithmetic expression

Tree Traversal

- Inorder Traversal: X=A/B*C*D+E

  ⇒Infix form, Example Program

- Preorder Traversal: =X+**/ABCDE

  => Prefix form , Extended Learning

- Postorder Traversal: XAB/C*D*E+=

  => Postfix form , Extended Learning

Example Program for Inorder in C++

```
void Tree::inorder()
//Driver calls workhorse for traversal of entire tree. The driver is
//declared as a public member function of Tree
{
   inorder(root);
}

void Tree::inorder(TreeNode *CurrentNode)
//Workhorse traverses the subtree rooted at CurrentNode.
//The workhorse is declared as a private member function of Tree.
{
   if(CurrentNode) {
      inorder(CurrentNode->LeftChild);
      cout<<CurrentNode->data;
      inorder(CurrentNode->RightChild);
}
```

Recursive inorder traversal

Example Program for Inorder in Java

```
public class BNode {
   public BNode leftBNode, rightBNode; // the nodes
   public AnyClass anyClass; //the AnyClass objext
   public BNode(AnyClass anyClass ) {//constructor
      this.anyClass= anyClass; this.leftBNode = null; this.rightBNode = null;
   }
}
public class BinTree {
   BNode theBTRootNode;
   public BinTree() // constructor
      { theBTRootNode = null; }
}
void inorder(BNode theRootNode) {
   if (theRootNode != null) {
      inorder(theRootNode.leftBNode);
       theRootNode.show();
      inorder(theRootNode.rightBNode);
   }
}
```

Recursive inorder traversal

Iterative Inorder Traversal

```
void Tree::NonrecInorder()
// nonrecursive inorder traversal using a stack
{
      Stack<TreeNode *> s;   // declare and initialize stack
      TreeNode *CurrentNode = root;
      while (1) {
            while (CurrentNode) { // move down LeftChild fields
                s.Add(CurrentNode);       // add to stack
                CurrentNode = CurrentNode->LeftChild;
            }
            if (!s.IsEmpty()) {              // stack is not empty
                CurrentNode = *s.Delete(CurrentNode);
                cout << CurrentNode->data << endl;
                CurrentNode = CurrentNode->RightChild;
            }
            else break;
      }
}
```

Nonrecursive inorder traversal

Theorem to Ordering of Expression

- One Inorder with one Postorder or Preorder sequences can determine an unique binary tree meet the specific sequences
  - Inorder + Postorder sequences
  - Inorder + Preorder sequences
- Only one sequence or there are One Postorder and One Preorder sequences will find more than two binary trees to meet the specific sequence(s).

Level-Order Traversal

- All previous mentioned schemes use stacks.
- Level-order traversal uses a queue.
- Level-order scheme visit the root first, then the root's left child, followed by the root's right child.
- All the node at a level are visited before moving down to another level.

Level-Order Traversal of A Binary Tree

```
void Tree::LevelOrder()
// Traverse the binary tree in level order
{
    Queue<TreeNode *> q;
    TreeNode *CurrentNode = root;
    while (CurrentNode) {
        cout << CurrentNode->data<<endl;
        if (CurrentNode->LeftChild) q.Add(CurrentNode->LeftChild);
        if (CurrentNode->RightChild) q.Add(CurrentNode->RightChild);
        CurrentNode = *q.Delete();
    }
}
```

+*E*D/CAB

Some Other Binary Tree Functions

- With the inorder, postorder, or preorder mechanisms, we can implement all needed binary tree functions. E.g.,
  - Insert Node into Tree / Delete Node from tree
  - Copying Binary Trees, Extended Learning
  - Testing Equality, Example Program
    - Two binary trees are equal if their topologies are the same and the information in corresponding nodes is identical.

Example Program for Equal() in C++

```
//Driver-assumed to be a friend of Class Tree
Int operator==(const Tree& s, const Tree& t)
{
   return equal(s.root, t.root);
}

//Workhorse-assumed to be a friend of Class TreeNode
int equal(TreeNode *a, TreeNode *b)
//This function return 0 if the subtrees at a and b are not equivalent
//Otherwise, it will return 1
{
   if((!a) && (!b)) return 1; //both a and b is Empty
   if (a && b // both a and b are not empty
         && (a->data == b->data)  // data is the same
         && equal(a->LeftChild, b->LeftChild)  //Left subtree are the same
         && equal(a->RightChild, b->RightChild))  //Right subtree are the same
            return 1;
    return 0;
}
```


Traversal Without A Stack

- Use of parent field to each node.
- Use of two bits per node to represents binary trees as threaded binary trees.
- Threaded Binary Tree
  - For extended learning

延伸學習

- Threaded Binary Tree
  - http://en.wikipedia.org/wiki/Threaded_binary_tree
  - https://www.youtube.com/watch?v=HFbA0klBKIg
  - http://www.sanfoundry.com/cpp-program-implement-threaded-binary-tree/
- Tree and Binary Tree
  - http://www.cyut.edu.tw/teacher/ft00009/Chap05-Trees.pdf
  - http://www.cs.berkeley.edu/~kamil/teaching/su02/080802.ppt
  - http://sjchen.im.nuu.edu.tw/Datastructure/98/course06.pdf
  - http://math.hws.edu/eck/cs225/s03/binary_trees/
  - http://www.newthinktank.com/2013/03/binary-tree-in-java/

Binary Search Tree

- Binary search tree provide a better performance for search, insertion, and deletion.
- Definition: A binary serach tree is a binary tree. It may be empty. If it is not empty then it satisfies the following properties:
  - Every element has a key and no two elements have the same key (i.e., the keys are distinct)
  - The keys (if any) in the left subtree are smaller than the key in the root.
  - The keys (if any) in the right subtree are larger than the key in the root.
  - The left and right subtrees are also binary search trees.

Binary Search Tree (Cont.)

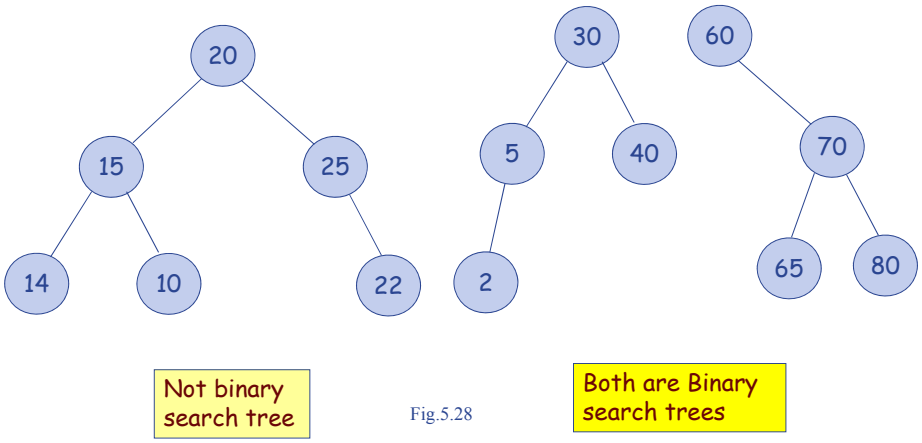


Fig.5.28

Searching A Binary Search Tree

- If the root is 0, then this is an empty tree. No search is needed.
- If the root is not 0, compare the x with the key of root.
  - If x equals to the key of the root, then it's done.
  - If x is less than the key of the root, then no elements in the right subtree can have key value x. We only need to search the left tree.
  - If x larger than the key of the root, only the right subtree is to be searched.
- Program 5.20 in Recursive
- Program 5.21 in Iterative

Searching A Binary Search Tree

- Program 5.20

```
template <class K, class E> // Driver, class K is Data Type of Key value, E is node element
pair<K, E>* BST<K, E> :: Get(const K& k)
{ // Search the identified key value k from the binary search tree, starting at root
 // if it is found, return the pointer to the node element with k; else return 0 (Null).
   return Get(root, k);
}
template <class K, class E> // Major function
pair<K, E>* BST<K, E> :: Get(TreeNode <pair <K, E> >* p, const K& k)
{
  if (!p) return 0;
  if (k < p→data.key) return Get(p→leftChild, k);
  if (k > p→data.key) return Get(p→rightChild, k);
  return &p→data;
}
```


Searching A Binary Search Tree

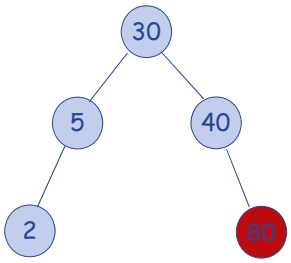
- Program 5.21

```
template <class K, class E>
pair<K, E>* BST<K, E> :: Get(const K& k)
{
  TreeNode < pair<K, E> > *currentNode = root;
  while (currentNode) {
     if (k < currentNode →data.key)
       currentNode = currentNode →leftChild;
     else if (k > currentNode →data.key)
                currentNode = currentNode →rightChild;
          else return & currentNode →data;
    }

    // Not found
    return 0;
}
```

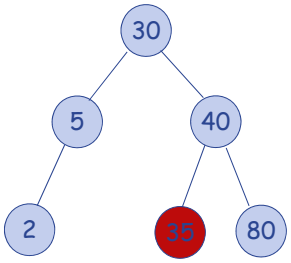
Insertion To A Binary Search Tree

- Before insertion is performed, a search must be done to make sure that the value to be inserted is not already in the tree.
- If the search fails, then we know the value is not in the tree. So it can be inserted into the tree.
- It takes O(h) to insert a node to a binary search tree.

Inserting Into A Binary Search Tree



Insert 80 into Fig. 5.28 (b)



Insert 35 into the example by left

Insertion Into A Binary Search Tree

```
Template <class Type>
Boolean BST<Type>::Insert(const Element<Type>& x)
// insert x into the binary search tree
{
    // Search for x.key, q is the parent of p
    BstNode<Type> *p = root; BstNode<Type> *q = 0;
    while(p) {
        q = p;
        if (x.key == p->data.key) return FALSE; // x.key is already in tree
        if (x.key < p->data.key) p = p->LeftChild;
        else p = p->RightChild;
    }
    // Perform insertion
    p = new BstNode<Type>;
    p->LeftChild = p->RightChild = 0; p->data = x;
    if (!root) root = p;
    else if (x.key < q->data.key) q->LeftChild = p;
      else q->RightChild = p;
    return TRUE;
}
```

O(h)

Program 5.23

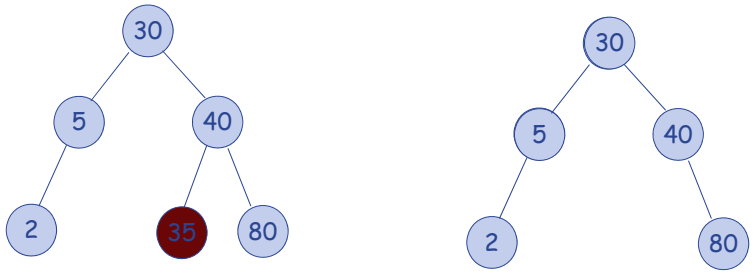
TSEG Tamkang University Software Engineering Group 淡江軟體工程實驗室 http://www.tkse.tku.edu.tw/

Deletion From A Binary Search Tree

- Delete a leaf node (Delete directly)
  - A leaf node which is a right child of its parent
  - A leaf node which is a left child of its parent
- Delete a non-leaf node
  - A node that has one child (Delete and Change link)
  - A node that has two children
    - Replaced by the largest element in its left subtree, or
    - Replaced by the smallest element in its right subtree
- Again, the delete function has complexity of O(h)

http://inhon.tkse.tku.edu.tw

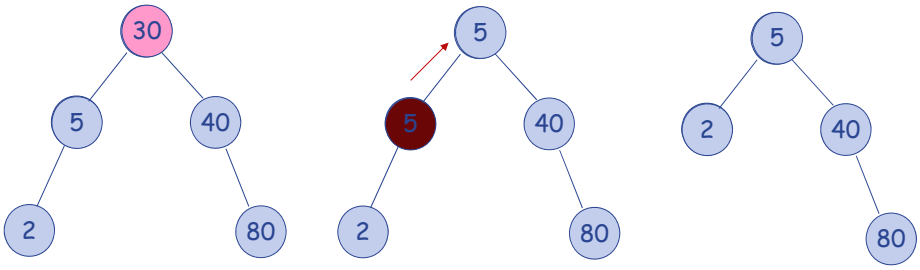
Deleting From A Binary Search Tree



Delete 35, it is a leaf

TSEG Tamkang University Software Engineering Group 淡江軟體工程實驗室 http://www.tkse.tku.edu.tw/

Deleting From A Binary Search Tree



Delete 30, it has two children

Replace it in the largest value in its left subtree

Delete the original replacing node, it is an example of on-child node

http://inhon.tkse.tku.edu.tw

延伸學習

- Lecture
  - http://en.wikipedia.org/wiki/Binary_search_tree
  - http://www.csie.ntnu.edu.tw/~u91029/Order.html
  - http://www.cs.swarthmore.edu/~newhall/unixhelp/Java_bst.pdf
  - http://algs4.cs.princeton.edu/32bst/
  - https://www.youtube.com/watch?v=pYT9F8_LFTM
  - https://www.youtube.com/watch?v=COZK7NATh4k
- Programming
  - http://codereview.stackexchange.com/ (in C++)
  - http://cplusplus.happycodings.com/algorithms/code4.html (in C++)
  - http://algs4.cs.princeton.edu/32bst/BST.java.html (in Java)

Priority Queues

- In a priority queue, the element to be deleted is the one with highest (or lowest) priority.
- An element with arbitrary priority can be inserted into the queue according to its priority.
- A data structure supports the above two operations is called max (min) priority queue.

Examples of Priority Queues

- Suppose a server that serve multiple users. Each user may request different amount of server time. A priority queue is used to always select the request with the smallest time. Hence, any new user's request is put into the priority queue. This is the min priority queue.
- If each user needs the same amount of time but willing to pay more money to obtain the service quicker, then this is max priority queue.

TSEG Tamkang University Software Engineering Group 淡江軟體工程實驗室 http://www.tkse.tku.edu.tw/

Priority Queue Representation

- **Unorder Linear List**
  - **Addition complexity: O(1)**
  - **Deletion complexity: O(n)**
- **Chain**
  - **Addition complexity: O(1)**
  - **Deletion complexity: O(n)**
- **Ordered List**
  - **Addition complexity: O(n)**
  - **Deletion complexity: O(1)**

Three ways to represent a Priority Queue

Comparison of the time complexity of their operation

http://inhon.tkse.tku.edu.tw

Max (Min) Heap

- Heaps are frequently used to implement priority queues. The complexity is O(log n).
- Definition: A max (min) tree is a tree in which the key value in each node is no smaller (larger) than the key values in its children (if any). A max heap is a complete binary tree that is also a max tree. A min heap is a complete binary tree that is also a min tree. (it means it can be implement in Array)

Max Heap Examples

14	12	7	10	8	6		

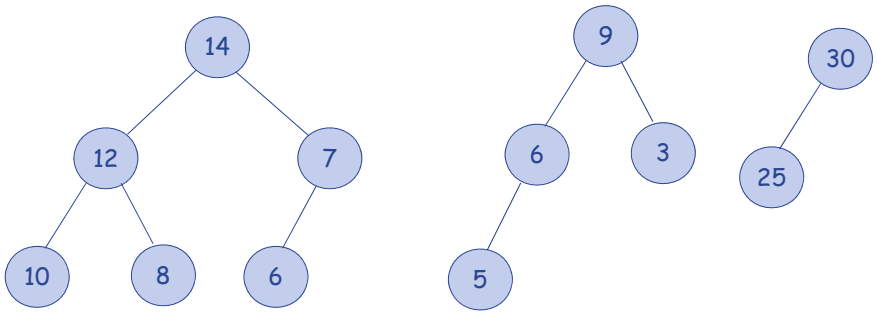


Fig. 5.24 Max Heaps

Min Heap Examples

2	7	4	10	8	6		

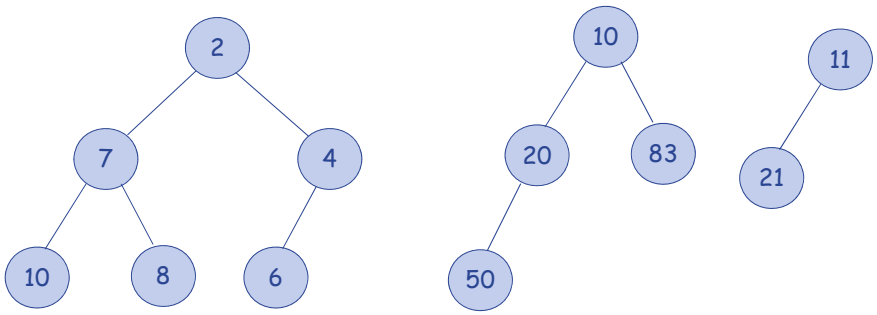
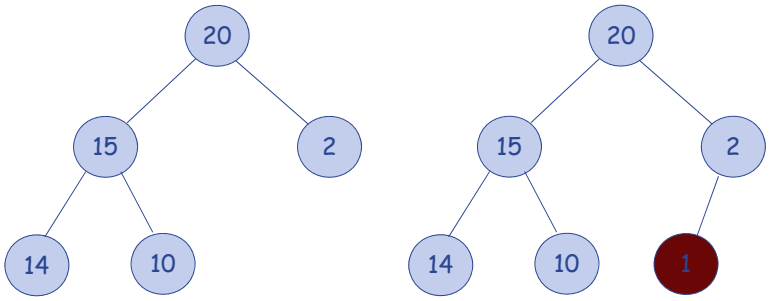


Fig. 5.25 Min Heaps

Basic operations for Max / Min heap

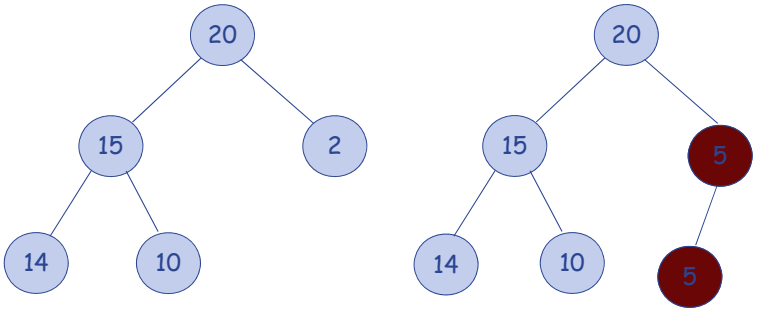
- Creation of an Empty heap (Constructor)
  - Program 5.15 for Max Heap
- Insertion of a new element into the heap
  - Program 5.16 for Max Heap
- Deletion of the Largest element from the heap
  - Program 5.17 for Max Heap
- These operations are defined in ADT5.2

Insertion Into A Max Heap (1)



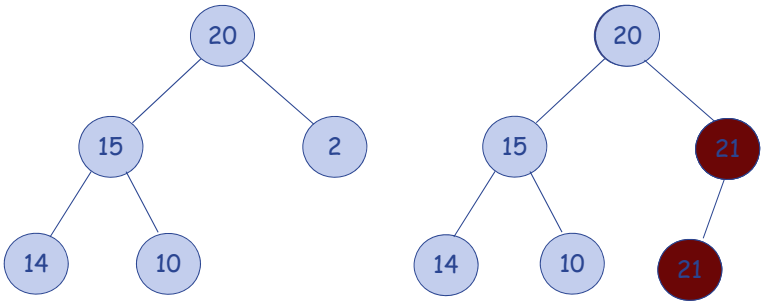
Insert new node with 1

Insertion Into A Max Heap (2)



Insert new node with 5

Insertion Into A Max Heap (3)



Insert new node with 21

Basic operations for Max / Min heap

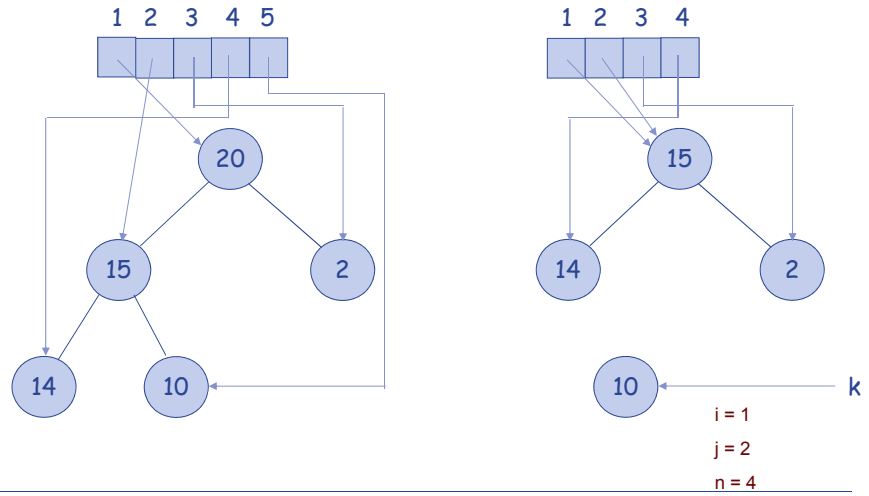
- ADT5.2

```
template <class T>
class MaxPQ {
public:
    virtual ~MaxPQ () {}
        // 虛擬解構子
    virtual bool IsEmpty () const = 0;
        // 回傳true 若且惟若優先權佇列是空的
    virtual const T& Top () const = 0;
        // 回傳指向最大元素的參照
    virtual void Push(const T&) = 0;
        // 加一個元素到優先權佇列中
    virtual void Pop () = 0;
        // 刪除最大優先權的元素
};
```

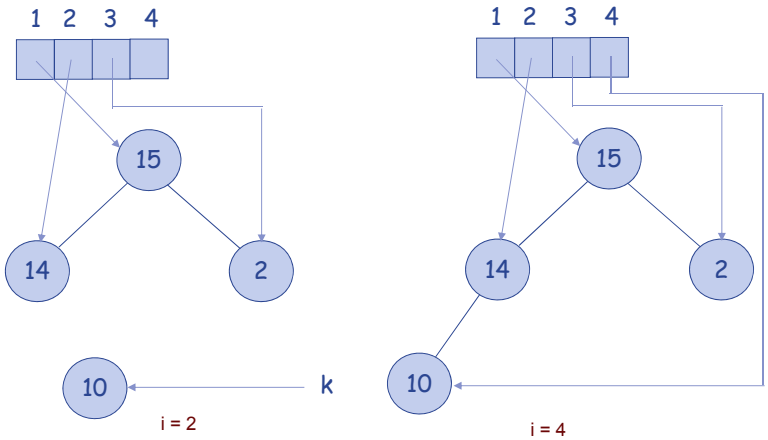
Insertion to A Max Heap

```
Template <class T>
void MaxHeap<T>::Push(const T& e)
{ // insert element e to a Max Heap ◦
   if (heapSize == capacity) { // Extend the capacity
      ChangeSize 1D(heap, capacity, 2*capacity);
      capacity *= 2;
   }
   int currentNode = + +heapSize;
   while (currentNode != 1 && heap[currentNode / 2] < e)
   { // Bubble up
      heap[currentNode] = heap[currentNode / 2];
                        // Move the value in Parent Node to Current
      currentNode /= 2;
   }
   heap[currentNode] = e;
}
```


Deletion From A Max Heap (Cont.)



Deletion From A Max Heap (Cont.)



i = 2
j = 4
n = 4

i = 4
j = 6
n = 4

Deletion From A Max Heap

```
template <class Type>
Element <Type>* MaxHeap <Type>::DeleteMax(Element <Type>& x)
// Delete from the max heap
{
    if (!n) {HeapEmpty(); return 0;}
    x = heap[1]; Element <Type> k = heap[n]; n--;
    for (int i = 1, j =2; j < n;)
    {
        if (j < n) if (heap[j].key < heap[j+1].key) j++;
        // j points to the larger child
        if (k.key >= heap[j].key) break;
        heap[i] = heap[j];
        i = j; j *= 2;
    }
    heap[i] = k;
    return &x;
}
```

Analysis

- The complexity of function *Insert* is O(log *n*)
- The complexity of function *DeleteMax* is O(log *n*)
- The complexity of delete an arbitrary element is O(*n*)
- Heap needs O(n) to perform deletion of a non-priority queue. This may not be the best solution.

TSEG Tamkang Universty Software Engineering Group 淡江軟體工程實驗室 http://www.tkse.tku.edu.tw/

延伸學習

- Lecture
  - http://en.wikipedia.org/wiki/Heap_%28data_structure%29
  - mail.sju.edu.tw/cm/course/data/09**heaptree**.ppt
  - http://openhome.cc/Gossip/AlgorithmGossip/HeapSort.htm
  - https://www.youtube.com/watch?v=LhhRbRXhB40
- Programming
  - http://www.cprogramming.com/tutorial/computersciencetheory/heapcode.html/ (in C++)
  - http://codereview.stackexchange.com/questions/32606/implementation-of-heap-sort (in Java)

http://inhon.tkse.tku.edu.tw

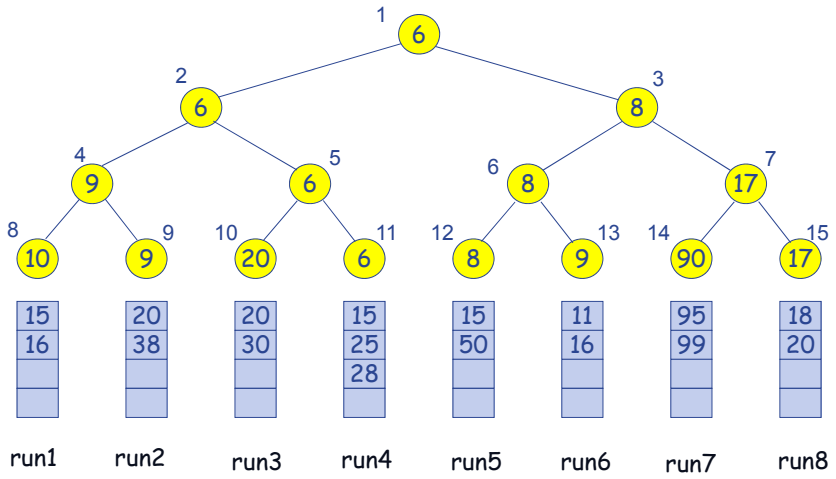
Selection Trees

- When trying to merge k ordered sequences (assume in non-decreasing order) into a single sequence, the most intuitive way is probably to perform k – 1 comparison each time to select the smallest one among the first number of each of the k ordered sequences. This goes on until all numbers in every sequences are visited.
- There should be a better way to do this.

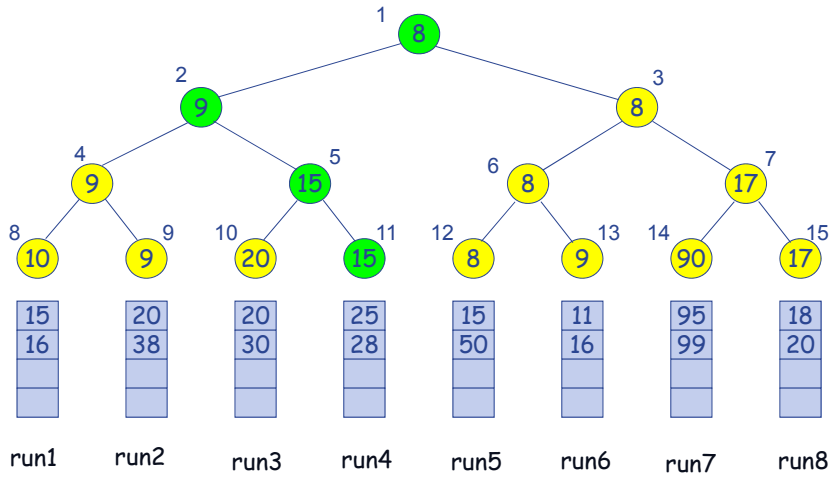
Winner Tree

- A winner tree is a complete binary tree in which each node represents the smaller of its two children. Thus the root represents the smallest node in the tree.
- Each leaf node represents the first record in the corresponding run.
- Each non-leaf node in the tree represents the winner of its right and left subtrees.

Winner Tree For k = 8



Winner Tree For k = 8 (Cont.)

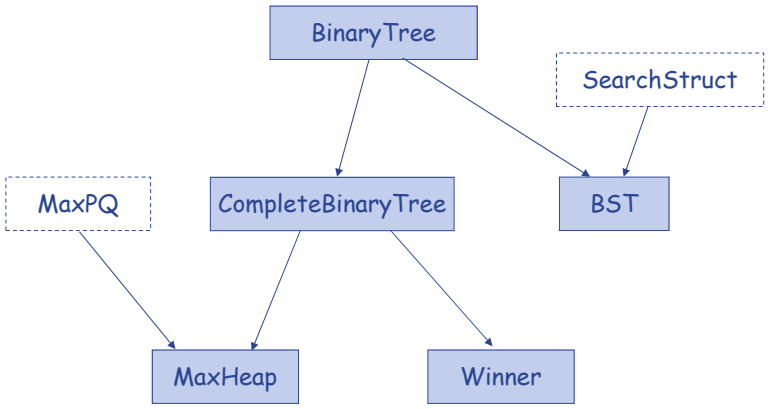


Analysis of Winner Tree

- The number of levels in the tree is
  - The time to restructure the winner tree is $O(log_2 k)$.
- Since the tree has to be restructured each time a number is output, the time to merge all n records is $O(n\ log_2 k)$.
- The time required to setup the selection tree for the first time is $O(k)$.
- Total time needed to merge the k runs is $O(n\ log_2 k)$.

TSEG Tamkang University Software Engineering Group 淡江軟體工程實驗室 http://www.tkse.tku.edu.tw/

An Object-Oriented System of Tree Data Structures

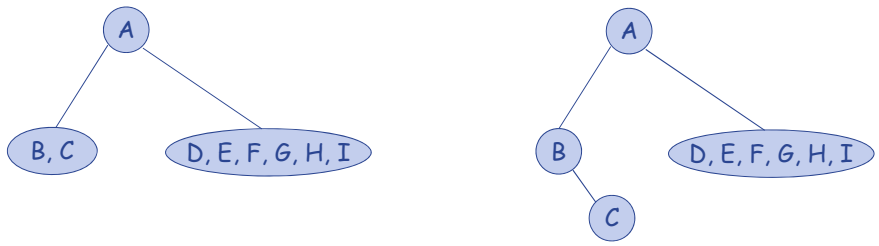


http://inhon.tkse.tku.edu.tw

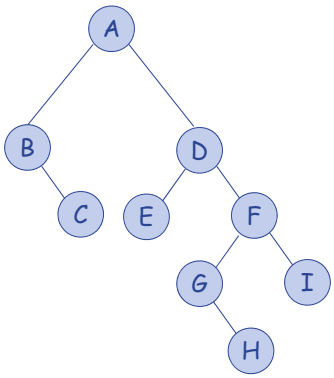
Uniqueness of A Binary Tree

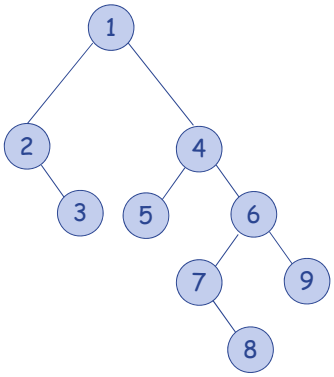
- In section 5.3 we introduced preorder, inorder, and postorder traversal of a binary tree. Now suppose we are given a sequence (e.g., inorder sequence BCAEDGHFI), does the sequence uniquely define a binary tree? Another way, can this sequence come from more than one binary tree?
- Thinking problem: What kind of provided sequences can construct one uniqueness Binary tree?

Constructing A Binary Tree From Its Inorder Sequence



Constructing A Binary Tree From Its Inorder Sequence (Cont.)

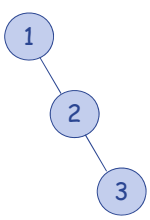




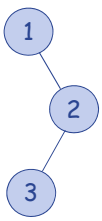
Preorder: 1, 2, 3, 4, 5, 6, 7, 8, 9

Inorder: 2, 3, 1, 5, 4, 7, 8, 6, 9

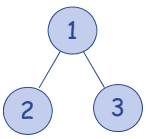
Distinct Binary Trees



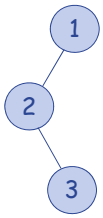
(1, 2, 3)



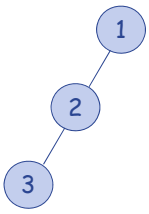
(1, 3, 2)



(2, 1, 3)



(2, 3, 1)



(3, 2, 1)

Distinct Binary Trees (Cont.)

- The number of distinct binary trees is equal to the number of distinct inorder permutations obtainable from binary trees having the preorder permutation, 1, 2, …, n.
- Computing the product of n matrices are related to the distinct binary tree problem.

  $M_1 * M_2 * \ldots * M_n$

  n = 3 $(M_1 * M_2) * M_3$ $M_1 * (M_2 * M_3)$

  n = 4 $((M_1 * M_2) * M_3) * M_4$

  $(M_1 * (M_2 * M_3)) * M_4$

  $M_1 * ((M_2 * M_3) * M_4)$

  $(M_1 * (M_2 * (M_3 * M_4)))$

  $((M_1 * M_2) * (M_3 * M_4))$

  Let bn be the number of different ways to compute the product of n matrices. $b_2 = 1$, $b_3 = 2$, and $b_4 = 5$.

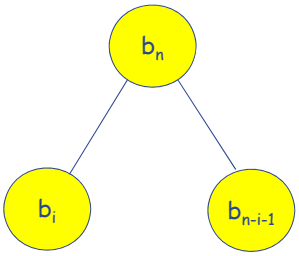
$$b_n = \sum_{i=1}^{n-1} b_i b_{n-i}, \quad n > 1$$

TSEG Tamkang University Software Engineering Group 淡江軟體工程實驗室 http://www.tkse.tku.edu.tw/

Distinct Binary Trees (Cont.)

- The number of distinct binary trees of n nodes is

$$b_n = \sum b_i b_{n-i-1}, \quad n \geq 1, and\ b_0 = 1$$



http://inhon.tkse.tku.edu.tw

Distinct Binary Trees (Cont.)

- Assume we let $B(x)=\sum_{i\geq 0} b_i x^i$ which is the generating function for the number of binary trees.
- By the recurrence relation we get

$$xB^2(x)=B(x)-1$$

$$B(x)=\frac{1-\sqrt{1-4x}}{2x}$$

$$B(x)=\frac{1}{2x}\left(1-\sum_{n\geq 0}\binom{1/2}{n}(-4x)^n\right)=\sum_{m\geq 0}\binom{1/2}{m+1}(-1)^m 2^{2m+1}x^m$$

$$b_n=\frac{1}{n+1}\binom{2n}{n} \quad \approx \quad b_n=O(4^n/n^{3/2})$$

TSEG

Tamkang University Software Engineering Group
淡 江 軟 體 工 程 實 驗 室
http://www.tkse.tku.edu.tw/

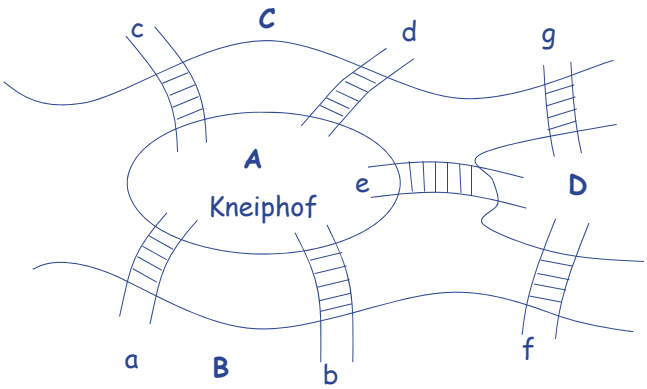
Graphs

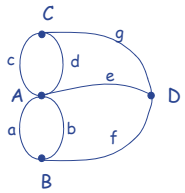
Presented by : Ying-Hong Wang
E-mail : inhon@mail.tku.edu.tw
Date :December 7, 2014

Konigsberg Bridge Problem

- A river Pregel flows around the island Keniphof and then divides into two.
- Four land areas A, B, C, D have this river on their borders.
- The four lands are connected by 7 bridges a – g.
- Determine whether it's possible to walk across all the bridges exactly once in returning back to the starting land area.

Konigsberg Bridge Problem (Cont.)





Euler's Graph

- Define the degree of a vertex to be the number of edges incident to it
- Euler showed that there is a walk starting at any vertex, going through each edge exactly once and terminating at the start vertex iff the degree of each vertex is even. This walk is called Eulerian.
- No Eulerian walk of the Konigsberg bridge problem since all four vertices are of odd edges.

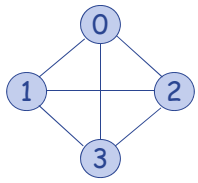
Application of Graphs

- Analysis of electrical circuits
- Finding shortest routes
- Project planning
- Identification of chemical compounds
- Statistical mechanics
- Genertics
- Cybernetics
- Linguistics
- Social Sciences, and so on …

Definition of A Graph

- A graph, G, consists tof two sets, V and E.
  - V is a finite, nonempty set of vertices.
  - E is set of pairs of vertices called edges.
- The vertices of a graph G can be represented as V(G).
- Likewise, the edges of a graph, G, can be represented as E(G).
- Graphs can be either undirected graphs or directed graphs.
- For a undirected graph, a pair of vertices (u, v) or (v, u) represent the same edge.
- For a directed graph, a directed pair <u, v> has u as the tail and the v as the head. Therefore, <u, v> and <v, u> represent different edges.

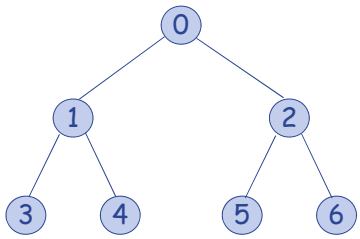
Three Sample Graphs



$V(G_1)$ = {0, 1, 2, 3}

$E(G_1)$ = {(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)}

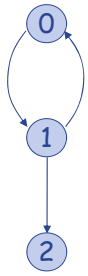
(a) G_1



$V(G_2)$ = {0, 1, 2, 3, 4, 5, 6}

$E(G_2)$ = {(0, 1), (0, 2), (1, 3), (1, 4), (2, 5), (2, 6)}

(b) G_2



$V(G_3)$ = {0, 1, 2}

$E(G_3)$ = {<0, 1>, <1, 0>, <1, 2>}

(c) G_3

Graph Restrictions

- A graph may not have an edge from a vertex back to itself.
  - (v, v) or <v, v> are called self edge or self loop. If a graph with self edges, it is called **a graph with self edges**.
- A graph man not have multiple occurrences of the same edge.
  - If without this restriction, it is called a **multigraph**.

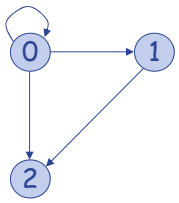
TSEG Tamkang Universty Software Engineering Group 淡江軟體工程實驗室 http://www.tkse.tku.edu.tw/

Complete Graph

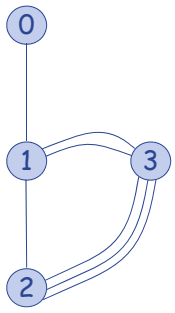
- The number of distinct unordered pairs (u, v) with u≠v in a graph with n vertices is n(n-1)/2.
- A *complete* unordered graph is an unordered graph with exactly n(n-1)/2 edges.
- A *complete* directed graph is a directed graph with exactly n(n-1) edges.

http://inhon.tkse.tku.edu.tw

Examples of Graphlike Structures



(a) Graph with a self edge



(b) Multigraph

Graph Edges

- If (u, v) is an edge in E(G), vertices u and v are adjacent and the edge (u, v) is the incident on vertices u and v.
- For a directed graph, <u, v> indicates u is adjacent to v and v is adjacent from u.