Algorithms

授課老師：張景堯

Course introduction

上課資訊

- 班級：資管三四甲乙(非研究所課程，但為資管碩先修)
- 地點：逸仙樓5樓資管系電腦教室。(以實體上課為原則)
- 上課時間：每週三上午9點10分～12點。
- 商談地點：下課時間或來信另約時間地點。
- 電子郵件：jychang@nccu.edu.tw
- 教科書：Steven S. Skiena. *The Algorithm Design Manual. 3rd ed.* Springer-Verlag, 2020. ISBN: 978-3030542559 www.algorist.com
- 參考書：
  - Adnan Aziz, Tsung-Hsien Lee, Amit Prakash. *Elements of Programming Interviews: The Insiders' Guide. 2nd ed*. CreateSpace, 2012. ISBN: 9781479274833 (C++,Python Available)
  - Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms. 3rd ed*. MIT Press, 2009. ISBN: 9780262033848.

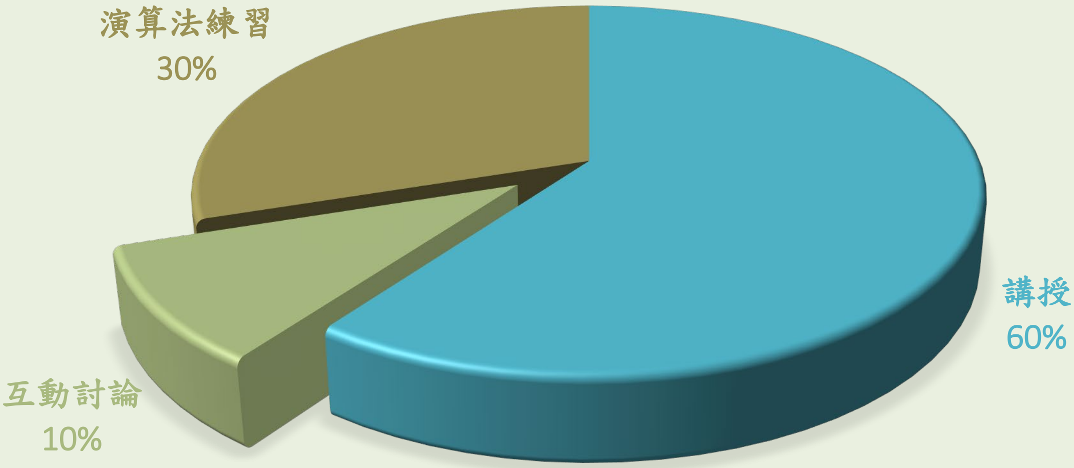
課程進度預估(上)

學期週次(日期)	課程內容	作業
01(2/19)	[Introduction] Course introduction, Introduction to algorithms (1-27) (**Exhaustive Search**)	Zuvio IRS setup & Find counterexamples
02(2/26)	[Preliminaries/Algorithm Analysis] Python Basic; Asymptotic notation(31-48), Logarithms and more(48-54)	Program Exercises of Python, Estimation, Big oh exercises, Reverse digits & The 3n+1 problem
03(3/5)	[Data Structures I] Elementary data structures(69-75)~Array	How many seats are available & Jolly jumper
04(3/12)	[Data Structures II] Linked List~, Dictionary(75-80), Binary Search Tree(81-87), Hashing (93-98)	Valid Parentheses, Test for Cyclicity & Sudoku validation
05(3/19)	[Sorting I] (**Divide and Conquer**) Priority Queues/Heapsort(87-89,115-125), Applications of Sorting(109-115)	Two Sum & Find the Nearest Repeated Entries
06(3/26)	[Sorting II] Mergesort/ Quicksort(127-138), Binsearch(148-150)	Merge two sorted arrays & Exact Sum
07(4/2)	Intercollegiate Activities (校際活動週)	
08(4/9)	Midterm Exam 期中考	

課程進度預估(下)

學期週次(日期)	課程內容	作業
09(4/16)	[Graph Traversal] Data structures for graphs(197-213), Breadth-first search (213-221)	Convert a Graph from Adjacent List to Adjacent Matrix & Making Wired Connections (Bipartite)
10(4/23)	Depth-First Search/Topological sort/Connectivity (221-235) [Weighted Graph] (**Greedy**) Minimum spanning trees(243-248)	Search a maze & Traffic Flow
11(4/30)	More MST(248-257), Shortest paths(257-267), Network flows and Bipartite matching(267-276)	Sending Packet & Road Network
12(5/7)	[Combinatorial search] **Backtracking** (281-295), Program optimization(295-302)	Permutation of a Multiset, N-Queens Problem & Implement a Sudoku solver
13(5/14)	[**Dynamic Programming**] Introduction to dynamic programming(307-314), Edit distance/Customizing edit distance(314-326)	Count The Number Of Ways To Traverse A 2D Array, Interleaving String & Find the Minimum Weight Path in a Triangle
14(5/21)	Athletic contests (校慶運動會)	
15(5/28)	Examples of dynamic programming(329-339), Limitations of dynamic programming(339-342) [Reduction] Reductions(335-357), Easy reductions(358-361), Harder reductions(361-373)	Maximum Monotone Subsequence, The Knapsack Problem & Subset Sum
16(6/4)	Final Exam 期末考	
17(6/11)	Work on Programming Assignments	Assignments 1-5
18(6/18)	Work on Programming Assignments	Assignments 6-10

授課方式



最重要的事

- Program Exercises 30%
- Midterm Exam 20%(期中考)
- Final Exam 30%(期末考)
- Participation 10%
  - Zuvio 出席率*75%+答對率*25% (滿分15，超過部分為Bonus)
  - [(全作答)X1+(部分作答)X0.5]/資料夾(課堂)總數，全作答視同出席、部分作答視同出席半堂課、無作答視同曠課
- Programming Assignments 10%
- Bonus 5%(95加滿，總分達95以上加分遞減)
  - 公式：95之前加分+(100-超出95)*95以上加分/10
    $\min(\max((95-Score),0),Bonus)+\min((100-Score),5) * (Bonus-\min(\max((95-Score),0),Bonus))/10$

加簽規定

- 本課程有開放加簽在第一次上課就會決定名單
- 碩班補先修為保障名額，本系(含輔系雙修)大四以上可優先登記
- 加簽人數以電腦教室容量為上限，超出上限則由電腦現場抽籤決定
- 若還有多餘名額才供其他身分抽籤。

Introduction to algorithms

Chapter.1-Introduction to algorithm Design

What Is An Algorithm?

- Algorithms are the ideas behind computer programs.
- An algorithm is the thing which stays the same whether the program is in assembly language running on a supercomputer in New York or running on a cell phone in Kathmandu in Python!(無關語言、執行平台)
- To be interesting, an algorithm has to solve a general, specified problem.
- An algorithmic problem is specified by describing the set of instances it must work on, and what desired properties the output must have.(明定輸入與輸出)

Example Problem: Sorting

- Input: A sequence of N numbers $a_1 \ldots a_n$
- Output: The permutation (reordering) of the input sequence such as $a'_1 \leq a'_2 \ldots \leq a'_n$.

- We seek algorithms which are *correct* and *efficient*, while being *easy to implement*.(正確性、有效率及容易建置)
- A faster algorithm running on a slower computer will *always* win for sufficiently large instances, as we shall see.
- Usually, problems don't have to get that large before the faster algorithm wins.

[補充]

- 當我們使用某種技巧解決一個問題時，會產生一種<u>**逐步執行的程序**</u>(step-by-step procedure)來解決問題，該種逐步執行的程序即稱為解決這個問題的**演算法**。
- Def:
  - 完成特定功能之有限個指令之集合。
  - 需滿足下列5個性質：
    - ***Input***: *外界至少提供≥0個輸入*
    - ***Output***: *Algorithm至少產生≥1個輸出結果*
    - ***Definiteness*** *(明確): 每個指令必須是Clear and Unambiguous*
    - ***Finiteness*** *(有限性): Algorithm在執行有限個步驟後，必定終止*
    - ***Effectiveness*** *(有效性): Algorithm執行的過程可追蹤及結果是否是想要的*

Correctness(正確性)

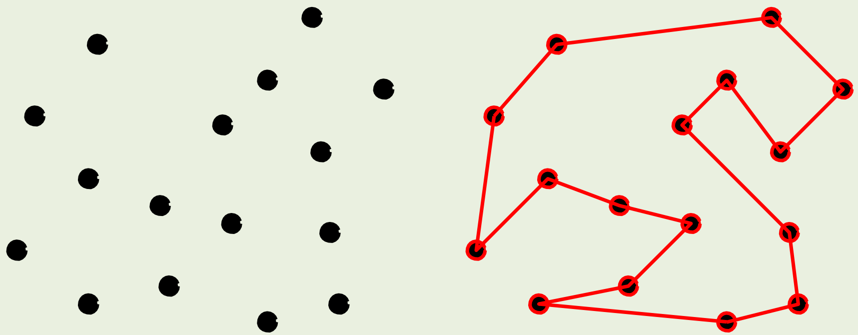
- For any algorithm, we must prove that it *always* returns the desired output for all legal instances of the problem.
- For sorting, this means even if (1) the input is already sorted, or (2) it contains repeated elements.
- Algorithm **correctness** is not obvious in many optimization problems!
- Algorithms *problems* must be carefully specified to allow a provably correct algorithm to exist. We can find the "shortest tour" but not the "best tour".(問題本身也要問的正確)

Robot Tour Optimization

- Suppose you have a robot arm equipped with a tool, say a soldering iron. To enable the robot arm to do a soldering job, we must construct an ordering of the contact points, so the robot visits (and solders) the points in order.(焊接機器手臂)
- We seek the order which minimizes the testing time (i.e. travel distance) it takes to assemble the circuit board.

Find the Shortest Robot Tour

- Input: A set S of n points in the plane.
- Output: What is the shortest cycle tour that visits each point in the set S?



- You are given the job to program the robot arm. Give me an algorithm to find the most efficient tour!

Nearest Neighbor Tour(最近鄰居)

- A popular solution starts at some point p_0 and then walks to its **nearest neighbor p_1 first**, then repeats from p_1, etc. until done.

Pick and visit an initial point p_0

$p = p_0$

$i = 0$

While there are still unvisited points

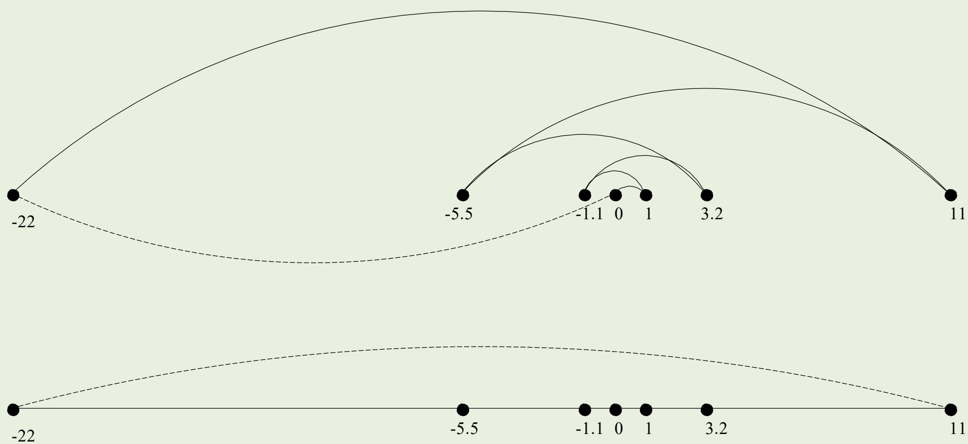
  $i = i + 1$

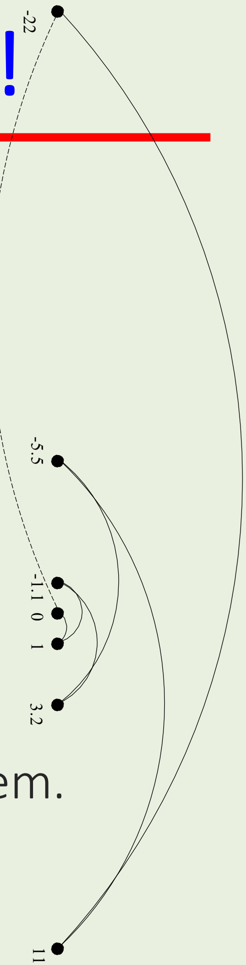
  Let p_i be the closest unvisited point to p_{i-1}

  Visit p_i

Return to p_0 from p_i

Nearest Neighbor Tour is Wrong!





- Starting from the *leftmost* point will not fix the problem.

Closest Pair Tour(最近一對)

- Another idea is to repeatedly **connect the closest pair of points** whose connection will not cause a cycle or a three-way branch, until all points are in one tour.

Let n be the number of points in the set

For $i = 1$ to $n - 1$ do

  $d = \infty$

  For each pair of endpoints (x, y) of partial paths

    If $dist(x, y) \leq d$ then

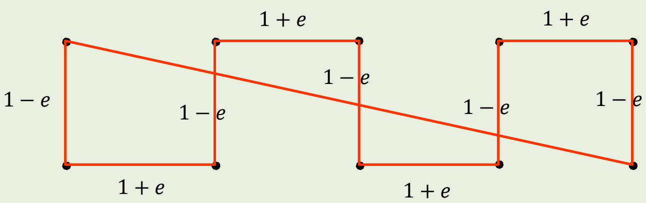
      $x_m = x, y_m = y, d = dist(x, y)$

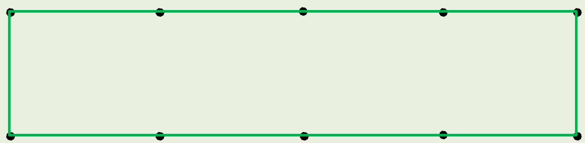
  Connect (x_m, y_m) by an edge

Connect the two endpoints by an edge.

Closest Pair Tour is Wrong!

- Although it works correctly on the previous example, other data causes trouble:





A Correct Algorithm: Exhaustive Search窮舉

- We could try **all** possible orderings of the points, then select the one which minimizes the total length:

$d = \infty$

For each of the $n!$ permutations Π_i of the n points
    If $(cost(\Pi_i) \leq d)$ then
        $d = cost(\Pi_i)$ and $P_{min} = \Pi_i$

Return P_{min}

- Since all possible orderings are considered, we are guaranteed to end up with the shortest possible tour.

Exhaustive Search is Slow!

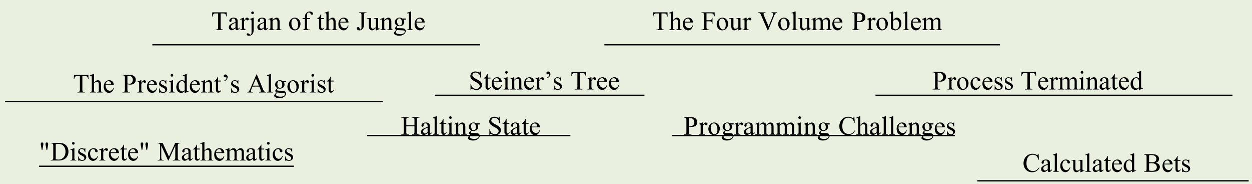
- Because it tries all $n!$ permutations, it is much too slow to use when there are more than 10-20 points.
  20! =2,432,902,008,176,640,000
- No efficient, correct algorithm exists for the *traveling salesman problem*(TSP), as we will see later.

Expressing Algorithms(表達演算法)

- We need some way to express the sequence of steps comprising an algorithm.
- In order of increasing **precision**, we have English or other natural language, pseudocode, and real programming languages. Unfortunately, **ease of expression** moves in the reverse order.
- I prefer to describe the ideas of an algorithm in English, moving to pseudocode to clarify sufficiently tricky details of the algorithm.

Selecting the Right Jobs

- A movie star wants to the select the maximum number of staring roles such that no two jobs require his presence at the same time.



The Movie Star Scheduling Problem

- Input: A set I of n intervals on the line.
- Output: What is the *largest* subset of mutually non-overlapping intervals which can be selected from I?

- Give an algorithm to solve the problem!

Earliest Job First(最早開始先做)

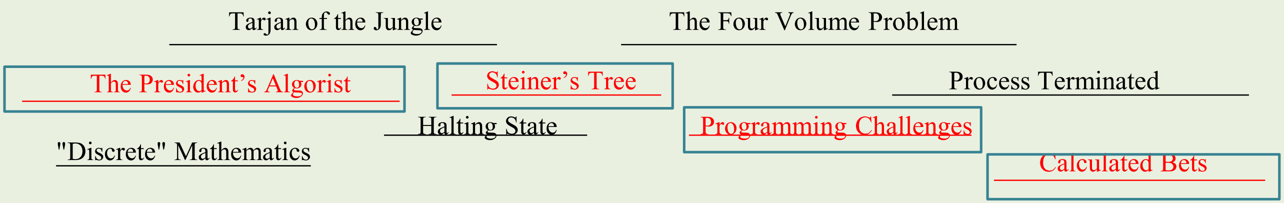
- Start working as soon as there is work available:

$EarliestJobFirst(I)$

Accept the earliest starting job j from I which does not overlap any previously accepted job, and repeat until no more such jobs remain.

Result of Earliest Job First

- Seems nice!



Earliest Job First is Wrong!

- The first job might be so long (War and Peace) that it prevents us from taking any other job.

Shortest Job First(最短工作先做)

- Always take the shortest possible job, so you spend the least time working (and thus unavailable).

$ShortestJobFirst(I)$

While ($I \neq \emptyset$) do

Accept the shortest possible job j from I.

Delete j, and intervals which intersect j from I.

Result of Shortest Job First

- Seems OK!

Tarjan of the Jungle

The Four Volume Problem

The President's Algorist

Steiner's Tree

Process Terminated

"Discrete" Mathematics

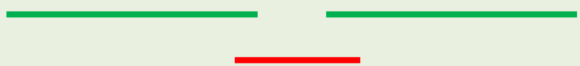
Halting State

Programming Challenges

Calculated Bets

Shortest Job First is Wrong!

- Taking the shortest job can prevent us from taking two longer jobs which barely overlap it.



Exhaustive Scheduling(窮舉法)

- Exhaustive approach will be certainly correct, however, which is enumerating the 2^n subsets of n things.

$ExhaustiveScheduling(I)$

$j = 0$

$S_{max} = \emptyset$

For each of the 2^n subsets S_i of intervals I

If (S_i is mutally non-overlapping) and (size(S_i) > j)

then j = size(S_i) and $S_{max} = S_i$.

Return S_{max}

First Job to Complete(最先結束的先做)

- Take the job with the *earliest completion date*:

$OptimalScheduling(I)$

While ($I \neq \emptyset$) do

Accept job j with the earliest completion date.

Delete j, and whatever intersects j from I.

First Job to Complete is Optimal!

- Proof: Other jobs may well have started before the first to complete (say, x), but all must at least partially overlap both x and each other.

- Thus we can select at most one from the group.
- The first these jobs to complete is x, so selecting any job but x would only block out more opportunities after x.
(因為所有在最先結束的工作x前啟動的，都多少會跟x與其他同時段的工作時間重疊，而x是影響最小的)

Demonstrating Incorrectness(找反例)

- Searching for counterexamples is the best way to disprove the correctness of a heuristic.
  - Think about all small examples.
  - Think about examples with ties on your decision criteria (e.g. pick the nearest point -> provide instances where everything is the same distance)
  - Think about examples with extremes of big and small. . .

Induction and Recursion(歸納與遞歸)

- Failure to find a counterexample to a given algorithm does not mean "it is obvious" that the algorithm is correct.
- Mathematical induction is a very useful method for proving the correctness of recursive algorithms.
- Recursion and induction are the same basic idea:
  - (1) basis case,
  - (2) general assumption which is true all the way to $n-1$,
  - (3) general case of n.

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Incremental Correctness(證明正確性)

- **Problem**: Prove the correctness of the following recursive algorithm for incrementing natural numbers, i.e. $y \to y+1$:

```
Increment(y)
    if y == 0 then return(1) else
        if (y mod 2) == 1 then
            return(2 · Increment(⌊y/2⌋))
        else return(y + 1)
```

- Solution:
  1. The basis case of y = 0, the value 1 is returned, and 0 + 1 = 1.
  2. Assume the function works correctly for the general case of $y = n-1$.
  3. If n is even numbers (For which (y mod 2) is 0), since $y+1$ is returned. the case of odd n (i.e. $n = 2m+1$ for some integer m) can be dealt with as:

$$2 \cdot Increment\left(\left\lfloor \frac{2m+1}{2} \right\rfloor\right) = 2 \cdot Increment\left(\left\lfloor m + \frac{1}{2} \right\rfloor\right)$$
$$= 2 \cdot Increment(m) = 2(m+1) = 2m+2 = n+1$$

Exercise

Problem of the Day

- The knapsack problem is as follows: given a set of integers $S = \{s_1, s_2, \ldots, s_n\}$, and a given target number T, find a subset of S which adds up exactly to T. For example, within $S = \{1, 9, 2, 10, 5\}$ there is a subset which adds up to $T = 22$ but not $T = 23$.
- Find counterexamples to each of the following algorithms for the knapsack problem. That is, give an S and T such that the subset is selected using the algorithm does not leave the knapsack completely full, even though such a solution exists. (找出選項中哪個是反例，使得演算法判斷該容量T的背包無法容納但卻是可以達成的，或是可以容納但其實是不行的)

Question

- Put the elements of S in the knapsack in left to right order if they fit, i.e. the first-fit algorithm?
- Put the elements of S in the knapsack from smallest to largest, i.e. the best-fit algorithm?
- Put the elements of S in the knapsack from largest to smallest?
- Put the elements of S in the knapsack which is the largest fitted one?

Go to Zuvio & Answer

- Web - https://irs.zuvio.com.tw/irs/login
- APP -

- 如果沒帳號請儘量用學校信箱註冊並用真實姓名，進入課程"演算法"