資結期末複習

* 題型：選擇，簡答，設計

不同 Data Structure

* Array
* Linked List
* Stack & Queue
* Tree / Heap
* Graph
* Hash Table
* 不同的的特性仍會依附在底層實作的種類

Array

* 1D、2D、3D...
* 需配置連續的記憶體空間
* 位置固定（已知）⇒ 可以 index 來 random access（O(1)）
* 宣告方式：static，dynamic

採作

- access / update：O(1)
- insert / delete：O(n)
- search：unsorted → O(n)
  sorted → O(logn)
- sort：根據演算法決定 O(n²)，O(nlogn)...

Linked list

* 在 pointer 已知的情況下，insert/delete：O(1)
* traverse 時：sequencial search：O(n)

Stack & Queue

* Stack
  - FILO，單一出入口
  - 可用 Array 或 Linked list 實作，
* Queue
  - FIFO，
  - 可以 Array，Linked list 實作，類型有 Linear Queue，Circular queue，Deque

Hash

* 實作方式

1. Chaining：Array + Linked list
   - 每個 bucket 裝一個 linked list
     ↓
     用來存相同 Hash value 元素的容器
2. Open Addressing：Only Array
   - Random Access
   - Collision 時依規則找下個位置
     - Linear probing：容易有 primary clustering
     - Quadratic probing：避免 primary clustering，但仍有 secondary clustering
     - Double hashing

* Primary clustering vs. Secondary clustering
  - Primary clustering：∵同個方向逐格探查，使得形成大條連續區塊
  - Secondary clustering：∵同 h(k) → 相同固定的探查序列，使得有多個分散但固定的小型聚集

* Loading factor
  - 空間利用率：$\alpha = \frac{n}{m}$，衡量 Hash table 的滿度
  - 性能影響：α 高表 { Open Addressing probing 速度越慢 / chaining 中每個 bucket 的 chain 越長

Graph

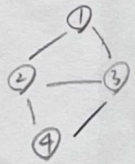
* G = (V, E)
* 表示法

1. Adjacency Matrix
   - Space complexity / Traverse：$O(V^2)$
   - Edge lookup、add：O(1)
2. Adjacency List
   - Space complexity / Traverse：O(V+E)
   - Edge lookup：O(deg(v))；add：O(1)

☆ Graph Traversal

- 為避免 infinite loop，重複 visit
  → 用 visited [] 記錄已訪問 node
- BFS：使用 queue 按層次訪問
- DFS：使用 stack 來記錄路徑
  與回溯點
- 標記為灰色
  - visited 判斷的位置
    1. Enqueue / push 之前：
       避免浪費 memory 存重複的節點
    2. dequeue / pop 之後：
       可能導致 stack overflow，但 DFS
       中 pop 後 mark 為標準遍歷 DFS

ex.



DFS simulation

1. push 前 mark ：traversal order：1, 2, 4, 3

visited：[1]
stack：[1] pop → print 1 → push 3, 2
mark 3, 2

visited：[1, 3, 2]
stack：[3, 2] pop → print 2 → push 4
mark 4
visited：[1, 3, 2, 4]
stack：[3, 4] pop → print 4 → 皆 mark
stack：[3] → print 3

2. pop 後 ：traversal order：1, 2, 3, 4

stack：[1] pop → print 1
visited：[1] push 3, 2

stack：[3, 2] pop → print 2
visited [1, 2] push 4, 3

stack：[3, 4, 3] pop → print 3 → push 4
visited [1, 2, 3]

stack：[3, 4, 4] pop → print 4 → 皆 visited
visited [1, 2, 3, 4]

總結：
push 前 mark：節點順序為第一次被看見的順序
pop 後 mark：順序為最後一次被發現的順序

② Tree

☆ Graph 的特例，Hierachical 結構，無 cycle

☆ 種類

general tree
- 結構條件 →
  1. complete binary tree → 除最後一層，其他全滿，由左至右填入
  2. full binary tree → 每個節點不能有 0 or 2 個子節點
  3. perfect binary tree → 所有節點皆有 2 個子節點且同一深度
- 規則條件 →
  1. Binary Search Tree → 平均 O(log n) 但可能退化成 O(n)
  2. AVL Tree → 左右子樹的高度差不超過 1
  3. Red Black Tree → 保證 O(log n)
  4. Heap → 同時滿足 complete Binary Tree

☆ Tree traversal