資料結構筆記 5-7 章

資訊二甲 10927159 林玟君

單元五、雜湊 Hashing

- □ Hash table 雜湊表 ≅ (automatic) Manger of lockers
  key → location mapping.
- □ Collision 碰撞 → 雜湊必須解決的問題
  - 不同的key 對應到同一個 location
- □ Functions.
  - Digit selection : Does not distribute item evenly.
  - Folding : Involves the entire search key.
  - Modulo arithmetic 取餘數 : 用質數 prime 可避免太多 collision.
  - Converting character strings : 轉換成數字.
- □ Collision Resolution
  1、Open Address　　*hash table 最好用質數!!!

key independent

a、**Linear probing** : Search the first location and the next.
- Promble : 鳩佔鵲巢，且當資料黏在一起，要搜尋很多筆數
- Delete : 刪掉某筆資料，下次搜尋遇到空隔，不代表不用繼續找

b、**Quadratic probing** : Search the first location and continue at the increments of 1^2、2^2、3^2 (二次方)
- Problem : 空間問題 → overflow area (可能有空間，但找不到)
- 也有群聚問題，但沒 Linear probing 嚴重

key dependent

c、**Double hashing** : 用2個 hash function 來減少 collision.
($h_1 \neq h_2$ 且 $h_2(key) \neq 0$) → step size
- table size v.s. step size 要互質，才能用到每個空間
  (table size：質數)

2、Restructuring the hash table：允許多個item存在同一格.

a. Buckets 空間換時間.

- Each location in the hash table is array.
- 若還有碰撞，則前面3種方式擇一.

b. Separate chaining

- Each hash table location keep a linked list.
- The size of the hash table is dynamic.

c. Buckets + Separate chaining. 記憶體維護.

- Memory management（把空掉的空間拿起來）
- Worst case：O(n) 缺：刪除時，空間使用率差

若集中在同一串，則變成要一個一個看

□ Inefficient Operations on Hashing. 不適合用雜湊

- Traversal：sorted order
- NN search：Find the smallest or largest item ex：Heap...
- Range Query：Find the item between two search key. ex：tree...

□ Application 1：Hash Join.

- Join is an important operation in database.
（在不同群的資料中，找查某筆特定資料）

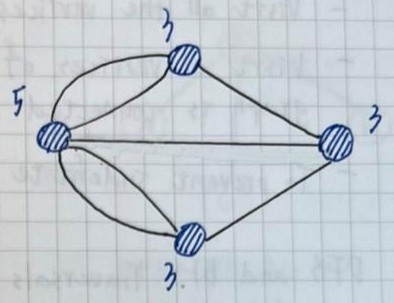
□ Application 2：Count-Min Sketch

- 在資源有限的情況下，紀錄下某些資料.（有可能有一些錯誤）.
- 應用：Hot IPs / users / files / words（熱門→超過某些門檻）

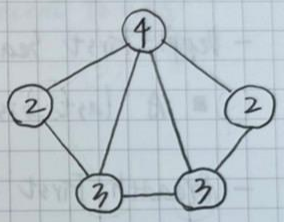
第六章 圖形概論

- Seven Bridge of Königsberg
  - V(G) = vertex set
  - E(G) = edge set
  - degree = Number of edges

    $\sum degree(V_i) = |E(G)| \times 2$

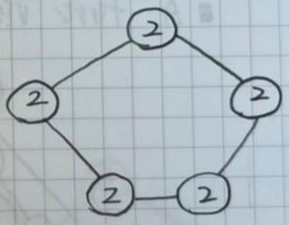


- Eulerian path (trial) / Euler walk
  - visits every edge exactly once
  - 0 or 2 node with odd degrees



- Eulerian circuit (cycle) / Euler tour
  - begin and end at the same vertex
  - 0 node with odd degrees

    {Euler Walks} ⊇ {Euler tours}



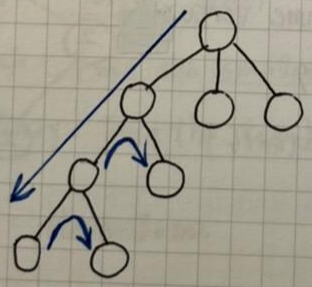
- Graphs as ADTs
  - Variations of an ADT graph are possible
    - Vertices may or may not contain value
    - 有無方向性
    - 有無權重
  - Insertion and deletion operation for graphs apply to vertices and edges
  - Graph can have traversal operations

□ Graph Traversals

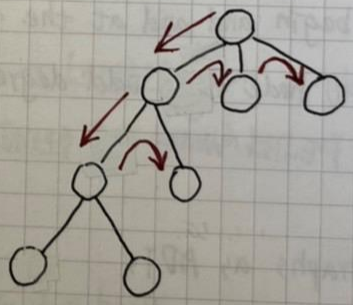
- Visit all the vertices that it can reach.
- Visit all vertices of the graph if and only if the graph is connected.
- To prevent indefinite loops (break the cycles) // 無窮迴圈

□ DFS and BFS Traversals

- Depth-First Search (DFS) Traversal 深度優先
  - A "last visited, first explored" strategy. ex: 堆疊 (返迴)
- Breadth-First Search (BFS) Traversal 寬度優先
  - A "first visited, first explored" strategy ex: queue



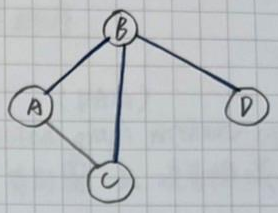
DFS



BFS

□ DFS in iterative from (stack)

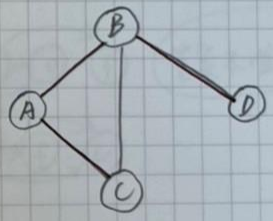
```
iterativeDFS (Vertex v) {
    s.createStack();
    s.push(v);
    Mark v as visited;
    while ( !s.empty() ) {
        u = s.getTop();
        if ( unvisited vertex w is adjacent to u) {
            s.push(w);
            Mark w as visited;
        } // if
        else s.pop();
    } // while
} // iterativeDFS()
```



DFS: AB BC BD

□ BFS in iterative from (queue)

```
iterativeBFS (Vertex v) {
    q.creatQueue();
    q.enqueue(v);
    Mark v as visited;
    while ( !q.isEmpty() ) {
        q.dequeue(u);
        for ( each unvisited vertex w adjacent to u) {
            Mark w as visited;
            q.enqueue(w);
        } // for
    } // while
} // iterativeBFS()
```



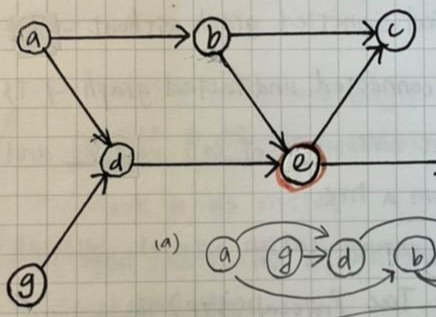
BFS: AB AC BD

第七章 圖形應用

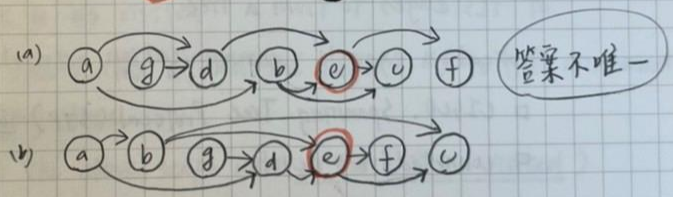
□ Topological Sort: Definition 拓樸排序

- Topological order
  a. A list of vertices in a directed graph without cycles (DAG)
     有向圖 且 沒有循環
  b. several topological orders are possible for a given graph.
- Topological sorting: Arranging the vertices into a topological order.

□ Topological: Example



edge (AOE) 邊
Activity-on-Vertex (AOV) 點
Network



□ Topological Sort: Algorithm

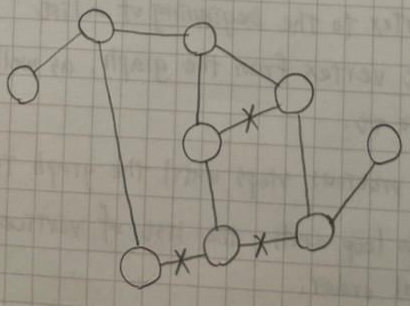
- topSort! (從無到有)
  1. Find a vertex that has no successor (out-degree = 0) 沒有指出去的邊.
  2. Add the vertex to the beginning of a list
  3. Remove that vertex from the graph, as well as all edges that lead to it.
  4. Repeat the previous steps until the graph is empty.
     - When the loop ends, the list of vertices will be in topological order.

- topsort2 (利用深度優先走訪)
  1. A modification of the iterative DFS algorithm.
  2. Push all vertices that have no predecessor onto a stack
     表示為末端，無指出去的邊
  3. Each time you pop a vertex from the stack, add it to the beginning of a list of vertices.
  4. When the traversal ends, the list of vertices will be in topological order.
  ▲ pop的順序即為要輸出的順序(拓撲排序)

□ Spanning Tree: Definition 生成樹

- A tree is an undirected connected graph without cycles (acyclic)
- A spanning tree of a connected undirected graph G is
  - A subgraph of G that contains all of G's vertices and enough of its edges to from a tree
  - Application example: communication network 網路規劃
    - CISCO、Spanning Tree Protocal (STP) 通訊協定
    - connected ↔ acyclic

- To obtain a spanning tree from a connected undirected graph with cycles
  - Remove edges until there are no cycle

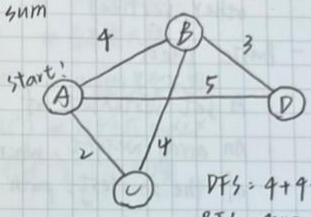


▲ 邊數 = 點數 - 1

▲ 有n個點, spanning tree 數量為 n^{n-2} 個

□ Minimum Spanning Tree = Definition 最小生成樹.

- Cost of spanning tree = sum of the edge weight on a spanning tree
- A minimum spanning tree of a connected undirected graph has a minimal edge-weight sum
- Other variations
  - (minimum) Steiner tree
  - k-minimum spanning tree



DFS: 4+4+3=11
BFS: 4+2+5=11
MST: 4+2+3=9

□ Minimum Spanning Trees = Prim's Algorithm

- Find a minimum spanning tree that begins at any given vertex.

1. Find the least-cost edge (v, u) from a visited vertex v to some unvisited vertex u.
2. Mark u as visited
3. Add the vertex u and the edge(v, u) to the minimum spanning tree.

(Repeat the above steps until all vertices are visited)

□ Minimum Spanning Trees = Kruskal's Algorithm

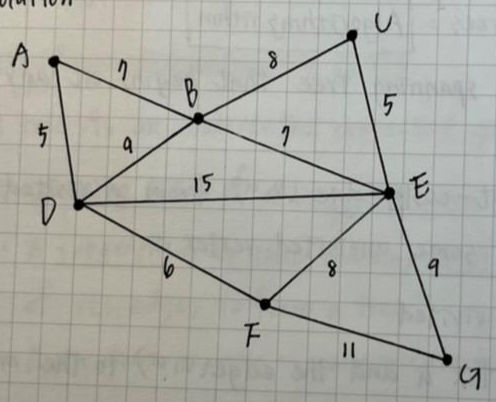
1. Creat a forest, where each vertex is a tree.
2. Find the least-cost edge (v, u) where vertex v and vertex u are from two different trees.
3. Merge the trees of vertex v and vertex u, and add the edge(v, u) to the minimum spanning tree.

(Repeat the above steps until |V|-1 edges)

☐ Shortest Paths - Dijkstra's Algorithm

- Problem definition
  - Find the shortest paths between a given origin and all other vertices
- Basic idea
  - A set vertexSet of selected vertices
  - An array weight, where weight[v] is the cheapest weight of the shortage path from vertex 0 to vertex

☐ Solution



vertexSet = {A, D, B, F, E, C}

weight = {0, 7, 15, 5, 14, 11, 22}