Hashing

	Insertion	Deletion	Retrieval	Traversal
Hashing	O(1)	O(1)	O(1)	O(n)

↳ relatively constant regardless of their locations

(key) → [Hash] → location

[Search key] ← [hash function] → Assign each search key to a single location

▲最完美的 hash function
- 每筆資料皆可放入，且有唯一位置

1. Easy and fast to compute
2. Places items evenly throughout the hash table
3. Involves the entire search key
4. Uses a prime base, if it uses modulo arithmetic
   質數

碰撞
▼Collision 解決
- Assigns distinct locations in the hash table to items involved in a collision

1. Open addressing
   - 尋找其他空位
   - 都滿了，碰撞增加
   ↳ 增加 table 大小（重新放入資料）
   1. Linear probing 線性
   2. Quadratic probing 平方
   3. Double probing 雙重碰撞

2. Restructure the hash table
   - Allows the hash table to accommodate more than one item in the same location.
   1. Buckets
   2. Separate Chaining

- Linear probing
- Quadratic probing

] Both create key independent probing sequences

- Double hashing — create key dependent probing sequences
  use two hash functions h_1, h_2, to reduce clustering problems
  h_1 for the first location and h_2 for the step size

▼ simple hash functions
1. Digit selection
   - Does not distribute items evenly
2. Folding
   - Involves the entire search key
3. Modulo arithmetic
   - The table size should be prime
4. Converting character strings
   - Use Integers in the hash function instead of search strings
     ex: ASCII

- Buckets
  - Each location in the hash table is itself an array

- Separate chaining
  - Each hash table location is a link list.
  - successfully resolves collisions
  - The size of the hash table is dynamic

▲ Summary
- a hash function should be extremely easy to compute and should scatter the search key evenly throughout the hash table
- a collision occurs when two different search keys hash into the same array location
- Hashing does not efficiently support operations the require the items to be order
- Simpler and faster than balanced search tree if
  1. Traversals are not important
  2. Maximum number of items is known
  3. Ample storage is available

Graph Basics

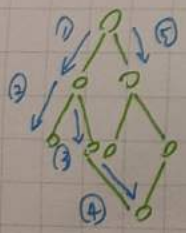
基本術語

- Undirected graph 無向圖
- ★ Directed graph (digraph) 有向圖
- Adjacent vertices 相鄰的
- Edge is incident to vertices
- Path: a sequence of edges
- Cycle: begin & end at the same vertex
- simple path: a path that passes through any vertex only once 每點又多到一次
- simple cycle: a cycle that passes through the other vertices only once 每點只經過一次，起點同終點
- Connected graph
  - There is a path between any two vertices
  - Disconnected graph
- Complete graph 圓=?
  - There is an edge between any two vertices
- Strong connected graph
  - For any two vertices on a digraph, there is a path from one vertex to the other
- weighted graph (權重圖)
  - the edges have numeric labels

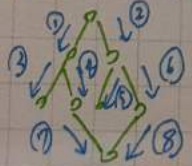
Graph 走訪

- visits all the vertices that it can reach
- visits all vertices of the graph if and only if the graph is connected
  - A connected component
    - The subset of vertices visited during a traversal that begins at a given vertex
- To prevent indefinite loops (break the cycles)
  - Mark each vertex during a visit, and
  - Never visit a vertex more than once

- Depth-First Search (DFS) Traversal
  深度優先走訪



- Breadth-First Search (BFS) Traversal
  寬度優先走訪



Graph As ADTs

- Variations of an ADT graph are possible
  - vertices may or may not contain values
    - Many Problems have no need for vertex values
    - Relationships among vertices is what is important
  - Either directed or undirected edges
  - Either weighted or unweighted edges
- Insertion and deletion operations for graphs apply to vertices and edges
- Graphs can have traversal operations

Graph Representations

- Most common implementations of graph
  1. Adjacency matrix (相鄰矩陣法)
  2. Adjacency list
- Adjacency matrix for a graph that has n vertices numbered 0, 1, ..., n-1
  - An n by n array matrix such that matrix[i][j] indicates whether an edge exists from vertex i to vertex j

Summary

- The most common implementations of a graph use either an adjacency matrix or adjacency list
- Graph searching
  - Depth-first search goes as deep into the graph as it can before backtracking
    - Uses a stack
  - Bread-first search visits all possible adjacent vertices before traversing further into the graph
    - Uses a queue

Graph App.

Topological sort:
- Topological order
  - A list of vertices in a directed graph without cycles (Acyclic Digraph or Directed Acyclic Graph, DAG) such that vertex x precedes vertex y if there is a directed edge from x to y in the graph
  - several topological orders are possible for a given graph
- Topological sorting
  - Arranging the vertices into a topological order

topSort1
1. Find a vertex that has no successor (out-degree = 0)
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
- A modification of the iterativeDFS algorithm
- Push all vertices that have no predecessor onto a stack
- Each time you pop a vertex from the stack, add it to the beginning of a list of vertices
- When the traversal ends, the list of vertices will be in topological order.

Activity-on-vertex (AOV) Network (由頂點形成，無權重)

每個固定開頭，可能有多種答案

Spanning Tree 生成樹 → to obtain a spanning tree from a connected undirected graph with cycles
- 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 form a tree
  - Application example: communication network

  ↳ Remove edges until there are no cycles

Properties
- Detecting a cycle in an undirected connected graph
  - A connected undirected graph that has n vertices must have at least n-1 edges
  - A connected undirected graph that has n vertices and exactly n-1 edges cannot contain a cycle
  - A connected undirected graph that has n vertices and more than n-1 edges must contain at least one cycle

- DFS / BFS for Spanning Trees
  - To create a spanning tree
    - Traverse the graph using either depth-first search (DFS) or breadth-first search (BFS) and mark the edges the you follow
    - After the traversal is complete, the graph's vertices and marked edges form a spanning tree

- Prüfer Sequence 普呂弗序列
  1. Each labeled tree with n vertices has a unique Prüfer sequence of length n-2
     - Conversion algorithms
       - Leaf with the smallest label
       - Keep the label of its parent
  2. Each prüfer sequence of length n-2 has a unique labeled tree with n vertices

Minimum Spanning Tree

- Cost of spanning tree
  - sum of the edge weights on a spanning tree
- A minimum spanning tree of a connected undirected graph has a minimal edge-weight sum
  - A particular graph could have several minimum spanning trees
- Find a minimum spanning tree that begins at any given vertex

① Prim's algorithm

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
4. Repeat the above steps until all vertices are visited

② Kruskal's algorithm

1. Create 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
4. Repeat the above steps until $|V|-1$ edges

③

1. Create a forest, where each vertex is a tree
2. For each tree T, do the following steps:
   - 2.1 Find the least-cost edge (v, u) where vertex v is in T and vertex u is outside T
   - 2.2 Merge the trees of vertex v and vertex u, and add the edge (v, u) to the minimum spanning tree
3. Repeat step 2 until only one tree is left