Ch 5 Hashing

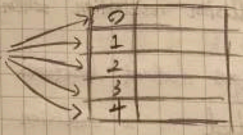
	Insertion	Deletion	Retrieval	Traversal
Unsorted Array Based	O(1)	O(n)	O(n)	O(n)
Unsorted Pointer Based	O(1)	O(n)	O(n)	O(n)
Sorted Array Based	O(n)	O(n)	O(logn)	O(n)
Sorted Pointer Based	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	O(logn)	O(logn)	O(logn)	O(n)
Hashing	O(1)	O(1)	O(1)	O(n)

Basic :

* Hash Table ≃ (automatic) Manger of lockers
  An array that contains the table Items, as assigned by a hash function (key → location mapping)

* Perfect hash function
  - Maps each Search key into a unique location.
  - Possible only if all the search keys are known.

key → Hash → location



ID? Birthday?

Callision

05-02

Def. Occurs when the hash function maps two or more items - all having different search keys - into the same array location.

Question: How many persons are assigned to 365 days such that there must be at least one callision?

Answer: Prob { 2 different birthdays } $= 1 \times \frac{364}{365}$

Prob { 3 ...... } $= 1 \times \frac{364}{365} \times \frac{363}{365}$

⋮

Prob { n ... } $= \frac{364 \times \cdots \times (366-n)}{365^{n-1}}$ ☆

Requirements

05-03

* Hash function - Assign each search key to a single location

1. Easy and Fast to compute
2. Place Items evenly throughout the hash table
3. Involves the entire search key
4. Use a prime base, if it uses modulo arithmetic

* Callision resolution schemes

Assigns distinct locations in the hash table to items involved in a callision.

05-04

Functions

Simple hash functions

1. Digit selection : Doesn't distribute items evenly
2. Folding : Involves the entire search key.
3. Modulo arithmetic : The table size should be prime

ex 表格 [0] - [12], 10027104 李XX, 資料 = 9

⇒ 資料 % 13, 15 % 13 = 2.

ex 非質數 :

items	% 12	% 24
140	[8]	[20]
92	[8]	[20]
91	[7]	[19]
87	[3]	[15]

Collision.

4. Converting character strings :

Using integers in the hash function instead of search strings.

char	ASCII
'0'	48
'1'	49
⋮	⋮

數字很大 ⇒ 提早取餘數

Collision Resolution

05-05

1. Open addressing.

Probe for an empty location in hash table.
As the hash table fills, collisions increase

→ Increse of Table size
(Need to hash the Items again)

2. Restructuring the hash table.

Allows the hash table to accommodate more than one item in the same location.

05-06

Linear probing : Search the first location and than the next

Primary clustering problem.

- Items need to cluster together and large clusters tend to get even larger
- Large clusters cause long probing sequence (sequence search)

Deletion Problem.

Empty location after deletions would incorrectly stop a probing sequence.

Quadratic Probing.

05-07

- Search the first location and then continue at the increments of 1^2, 2^2, 3^2 and so on.
- Probing sequence may not visit every location in the hash table.
- Secondary clustering problem.
  Different keys at the same location create the same probing sequence.
- Table size must be prime
  fewer alternate locations if it is not prime.

Open addressing

05-08

a. Linear Probing

b. Quadratic Probing

Both create key independent probing sequence.

c. double hashing

- It creates key dependent probing sequence.
- Use two hash functions h_1, h_2 to reduce clustering problem
- h_1 for the first location and h_2 for the step size

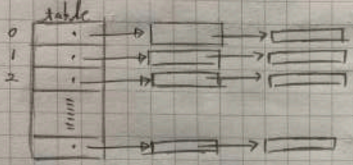
Restructuring the hash table

05-09

a. Buckets : Each location in the hash table is itself an array

b. Separate chaining :

- Each hash table location is a linked list
- Successfully resolves collisions
- The size of the hash table is dynamic



Buckets + Seperate chaining

- Memory management
- Worst case : $O(n)$

Efficiency

05-10

Average-case Analysis

- load factor α
  - Current number of items in the table / table size
  - Measure how full a hash table is

- Depends on whether the search is successful

  Unsuccessful searches generally require more time than successful searches.

05-11

Inefficient : operations on Hashing

- Traversal : Visit all the data in sorted order
- NN Search : Find the Item that has the smallest or the largest search key
- Range Query : Find the Item between two search keys

Summary :

1. A hash function should be extremely easy to compute and should scatter the search keys evenly throughout the hash table

2. A collision occurs when two different search keys hash into the same array location

3. Hashing doesn't efficiently support operations that require the items to be ordered

4. Simpler and faster than BST if

   ① traversals are not important
   ② Max num of items is known
   ③ Ample storage is available

Ch 7 Grapf Basic

Seven Bridges

06-01

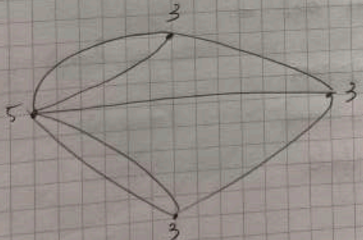
- Konigsberg in Prussia : Kaliningrad, Russia
  Pregel River
  Two large Islands
  Seven Bridges

- Leonhard Euler :
  - Find a work through the city that would cross each bridge once and exactly ones
  - The first paper in the history of grapf theory

- $G = \{V, E\}$
  - $V(G)$: vertex set
  - $E(G)$: edge set
  - degree (number of sets)

06-02

- Vertex type
  - odd or even degrees



点：陆地
线：桥

- Eulerian path (trial) / Euler walk :
  1. visit every edge exactly once
  2. 0 or 2 nodes with odd degrees

- Eulerian circuit (cycle) / Euler tour
  1. begin and end at the same vertex
  2. 0 node with odd degrees

Basic Terminologies

$\boxed{06-04}$

1. Undirected graph
2. Adjacent vertices
3. directed graph (digraph)
4. Edge is incident to vertices
5. Path : a sequence of edges
6. Cycle : begin & end at the same vertex
7. Simple path : a path that passes through any vertex only once
8. Simple cycle : a cycle that passes through the other vertices only ones

06-05

- Connected graph.
  - There is a path between any two vertices
  - Disconnected graph

- Complete graph $|E| = ?$
  - There is an edge between any two vertices

- Strong connected graph
  - For any two vertices on an digraph, there is a path from one vertex to the other

- Weighted graph
  - the edges have numeric labels

Graphs as ADTs

06-06

- 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 graph apply to vertices and edges

- Graphs can have traversal operations

Graph Representation

- Most common implementations of a graph
  1. adjacency matrix
  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 j.

Adjacency Matrix Example

06-07

- For an unweighted graph, matrix [i][j] is
  - 1, if an edge exists from vertex i to vertex j.
  - 0, 〃 no 〃

- For a weighted graph, matrix [i][j] is
  - The weight of the edge from vertex i to vertex j
  - ∞ (or 0) if no edge exists 〃

Adjacency List

06-08

Adjacency list for a directed graph that has n vertices numbered 0, 1, ..., n-1

- An array of n linked list
- The i^{th} linked list has a node for vertex j if and only if an edge exists from vertex i to j.
- The list's node can contain either:
  - Vertex j's values, if any.
  - An indication of vertex j's identity.

Graph Representation

* common operations:
  1. Determine whether there is an edge from vertex i to j.
  2. Find all verticies adjacent to a given vertex i

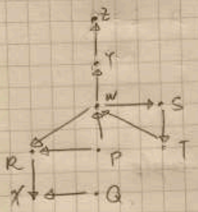
* Adjacency matrix
  1. Supports operation 1 more efficiency

* Adjacency list
  1. Supports operation 2 more efficiently
  2. Often requires less space than an adjacency matrix.

06-09

Sequential Representations

· Mapping from vertex tables to array indices



P	Q	R	S	T	W	X	Y	Z
0	1	2	3	4	5	6	7	8

nodes + edges

保留空間

[0]	[1]	[2]	[3]	⋯	[8]	[9]	[10]	⋯
10	12	13	14		20	20	2	⋯

* out-degree (P) = 12 − 10 = 2

06-10

Graph Traversal

DFS ·

1. Proceeds along a path from a vertex v to deeply into the graph as possible before backing up
2. A "last visited, first explored" strategy
3. Has a simple recursive form
4. Has an iterative form that uses a stack

BFS :

06-12

1. Visit every vertex adjacent to a vertex v before visiting any other vertex.

2. A " first in, first out " strategy.

3. An iterative form uses a queue

4. A recursive form is possible, but not simple

Summary :

- The most common implementations of a graph use either an adjacency matrix or adjacency list

- Graph Search :

$$\begin{cases} \text{DFS} - \text{stack} \\ \text{BFS} - \text{queue} \end{cases}$$

07-01

Topological Sort

Order :

- A list of verticies in a directed graph without cycles (DAG) such that vertex x preceds vertex y if there is a directed edge from x to y in the graph

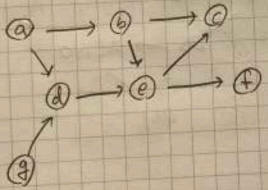
- Several topological orders are possible for a given graph

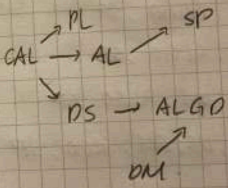
Sort :

- Arranging the vertices into a topological order.

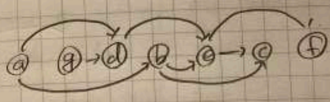
07-02

ex

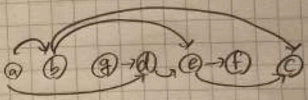




(a)



(b)



Algorithms .

07 - 03

STEP 1 : Find a vertex that has no successor

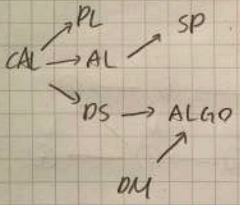
2 : Add the vertex to the begining of a list

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

Ex :

- SP
- ALGO , SP
- DM , ALGO , SP
- PL , DM , ALGO , SP
- DS , DM , ALGO , SP
- AL , DS , DM , ALGO , SP
- CAL , AL , DS , DM , ALGO , SP



Algorithms 2 .

07-04

- A modificiation of the DFS algorithms
- push all vertices that have no predecessor onto a stack
- Each time you pop a vertex from the stack, add it into the begining of a list of verticies
- When the traversal ends, the list of verticies will be in topological order

---

Spanning Tree Defination .

07-05

- A tree is an undirected connected graph without cycles
- 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
- To obtain a spanning tree from a connected undirected graph with cycles
  - Remove edges until there are no cycles

Properties

Detecting a cycle in an undirected connected graph

- DFS or BFS
- A connected undirected graph that has n vertices must have at least n-1 edges
- A connected undirected graph that has n verticies and exactly n-1 edges can't contain a cycle
- A connected undirected graph that has n verticies and more than n-1 edges must contain at least one cycle.

ex. 2 nodes $\rightarrow 2^{2-2} = 1$

3 — $\rightarrow 3^{3-2} = 3$

4 — $\rightarrow 4^{4-2} = 16$

Why n^{n-2} ?

$\Rightarrow$ Our proof is based on Prufer sequence

Graph isomorphism :

One of the NP problems

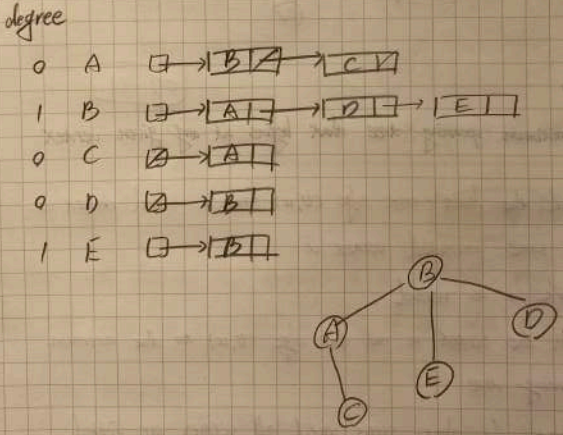
Prufer sequence

1. Each labeled tree with n vertices has a unique Prufer sequence of length n-2

   - Conversion algorithms
     - Leaf with the smallest key
     - keep the label of its parents

2. Each Prufer sequence of length n-2 has a unique labeled tree with n vertices

ex



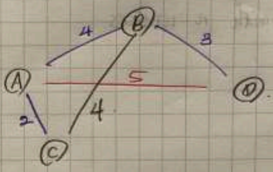
Minimum Spanning Tree :

Defination : $\boxed{07-10}$

1. Cost of spanning tree
   - Sum of the edge weights on a spanning tree.

2. A minimum spanning tree of a connected undirected graph has a minimum edge-weight sum.
   A particular graph could have several minimum spanning trees

$$\begin{cases} DFS : 4+4+3 = 11 \\ BFS : 4+2+5 = 11 \\ MST : 4+2+3 = 9 \end{cases}$$



Algorithms : $\boxed{07-11}$

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.
4. Repeat the above steps until all verties are visited.

Kruskal's Algorithms

07-12

STEP 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.

ex. assigned labels adjacency list

① - AC

② - BD

③ - AB

1

1 ~~2~~

1 ~~x~~

1 ~~2~~ ~~x~~

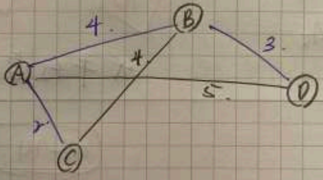
$A \to C_2 \to B_4 \to D_5$

$B \to D_3 \to A_4 \to C_4$.

$C \to A_2 \to B_4$

$D \to B_3 \to A_5$.

⇒



Sollin's Algorithms

07-13

STEP 1. Create a forest, where each vertex is a tree.

2. For each tree, do the following steps.

① Find the least-cost edge (v, u) where vertex v is in T and vertex u is outside T.

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.

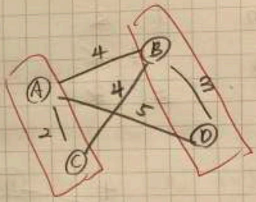
ex.

A → C_2 → B_4 → D_5

B → D_3 → A_4 → C_4

C → A_2 → B_4

D → B_3 → A_5



* 平行處理

Shortest Paths

07-14

Shortest path between two verticies in a weighted graph is the path that has the smallest sum of its edge weights.

* Problem definiation.

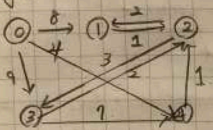
Find the shortest paths between a given origin and all other vertices.

* **Basic Idea**:
  - A set vertexset of selected vertices
  - An array weight, where weight [v] is the cheapest weight of the shortest path from vertex 0 (origin) to vertex v that pases through only the vertices in vertexSet.

07-15

Ex.

Origin



轉運站

question. How long is the shortest path from vertex 0 to 1 ?

step	v	vertexset	[0]	[1]	[2]	[3]	[4]
1	-	0 (全轉)	0	8	∞	9	4
2	4	0,4	0	8	5	9	4
3	2	0,4,2	0	7	5	8	4
4	1	0,4,2,1	0	7	5	8	4
5	3	0,4,2,1,3	0	7	5	8	4

n-1

0→4→2 5+2=7.

0→2 4+1=5.

加2後
check 從2走出的点

Single-Source All-Destination Shortest Paths (07-16)

Dijkstra's algorithm.

STEP 1. Initialize vertex & weight

2. Update weight for each vertex u not in vertexSet, which is adjacent to v.

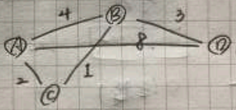
$$weight[u] = \min\{weight[u],\ weight[v] + edgeWeight[v,u]\}$$

3. Find the shortest path from v to u among every path that starts from v, passes vertices in vertexSet, and ends at a vertex not in vertexSet

if (weight[u] is min) vertexSet = vertexSet + {u}

4. Repeat steps 2, 3 until more vertex can be added

ex.



$\begin{cases} vertexSet_0 = \{\} \\ weight_0 = \{0, \infty, \infty, \infty\} \end{cases}$

$\begin{cases} vertexSet_1 = \{A\} \\ weight_1 = \{0, 4, 2, 8\} \end{cases}$

$\begin{cases} vertexSet_2 = \{A, C\} \\ weight_2 = \{0, 3, 2, 8\} \end{cases}$

$\begin{cases} vertexSet_3 = \{A, C, B\} \\ weight_3 = \{0, 3, 2, 6\} \end{cases}$

$A \to B = 4$

$A \to C \to B = 2 + 1 = 3$

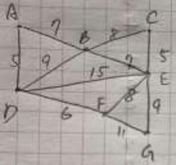
$A \to D = 8$

$A \to C \to B \to D = 3 + 3 = 6$

	A	B	C	D
A	0	4	2	8
B	4	0	1	3
C	2	1	0	∞
D	8	3	∞	0

Practice : Dijkstra's algorithm

(07-17)



step	v	vertexSet	A	B	C	D	E	F	G
1	–	A	0	7	∞	5	∞	∞	∞
2	D	A, D	0	7	∞	5	20	11	∞
3	B	A, D, B	0	7	15	5	14	11	∞
4	F	A, D, B, F	0	7	15	5	14	11	22
5	E	A, D, B, F, E							

(07-18)

Invariant : For each node $u \in S$, $d(u)$ is the length of the shortest s-u path

PF : by induction on $|S|$

Base case : $|S| = 1$ is trivial

Inductive hypothesis : Assume true for $|S| = k \ge 1$

① Let v be next node added to S, and let u-v be chosen edge

② the shortest s-u path plus (u,v) is an s-v path of length $\pi(v)$

③ Consider any s-v path P, We'll see that it's no shorter than $\pi(v)$

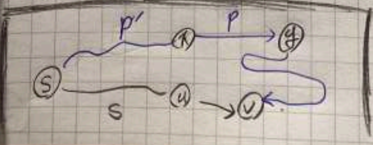
④ Let x-y be the first edge in P that leaves S, and let P' be the subpath to x

⑤ P is already too long as soon as it leave S.

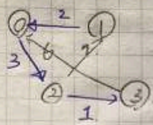
(07-19)

$$\boxed{\ell(P) \ge \ell(P') + \ell(x,y)} \ge \boxed{d(x) + \ell(x,y) \ge \pi(y)} \ge \pi(v)$$

nonnegative weights | inductive weights. | defn of $\pi(y)$ | Dijkstra chose v instead of y



Ex



0, 1	$0 \to 2 \to 1$	10
0, 2	$0 \to 2$	3
0, 3	$0 \to 2 \to 3$	4
1, 0	$1 \to 0$	2
1, 2	$1 \to 0 \to 2$	5
1, 3	$1 \to 0 \to 2 \to 3$	6
2, 0	$2 \to 3 \to 0$	7
2, 1	$2 \to 1$	7
2, 3	$2 \to 3$	1
3, 0	$3 \to 0$	6
3, 1	$3 \to 0 \to 2 \to 1$	16
3, 2	$3 \to 0 \to 2$	9

Floyd's Algorithm

07-20

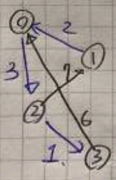
1. Initialize distance matrix D^{-1} = adjacency matrix
2. For $k = 0$ to $|V| - 1$

   $D^k \leftarrow D^{k-1}$; // add vertex k into vertexSet

   for $i = 0$ to $|V|-1$

   for $j = 0$ to $|V|-1$

   $D^0[1,2] = \min\{D^{-1}[1,2], D^{-1}[1,0] + D^{-1}[0,2]\}$



D^{-1}	0	1	2	3
0	0	∞	3	∞
1	2	0	∞	∞
2	∞	7	0	1
3	6	∞	∞	0

D^0	0	1	2	3
0	0	∞	3	∞
1	2	0	5	∞
2	∞	7	0	1
3	6	∞	∞	0

Directed Graph

D^{-1} : all-pairs shortest paths with no intermediate vertex

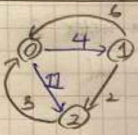
D^{0} : ---------------------------- intermediate vertex 0

D^{1} : -- 0, 1

D^{2} : -- 0, 1, 2

Anath example

07-21

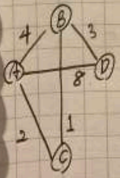


D^{-1}	0	1	2
0	0	4	11
1	6	0	2
2	3	∞	0

D^{0}	0	1	2
0	0	4	11
1	6	0	2
2	3	7	0

D^{2}	0	1	2
0	0	4	6
1	5	0	2
2	3	7	0

D^{1}	0	1	2
0	0	4	6
1	6	0	2
2	3	7	0



D^{-1}	A	B	C	D
A	0	4	2	8
B	4	0	1	3
C	2	1	0	∞
D	8	3	∞	0

D^{0}	A	B	C	D
A	0	4	2	8
B	4	0	1	3
C	2	1	0	10
D	8	3	10	0

D^1	A	B	C	D
A	0	4	2	7
B	4	0	1	3
C	2	1	0	4
D	7	3	4	0

D^2	A	B	C	D
A	0	3	2	6
B	3	0	1	3
C	2	1	0	4
D	6	3	4	0

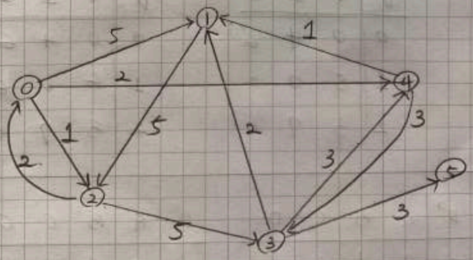
D^3	A	B	C	D
A	0	3	2	6
B	3	0	1	3
C	2	1	0	4
D	6	3	4	0

Self exercise6

07-22

1. Use Dijkstra's algorithm to find the shortest paths from vertex 0 to any other vertex. Show the content of vertexSet and weight obtained at the end of each round.

2. Chose the smallest label first if two or more verticies have the minimum weights



Summary

* Topological sorting produces a linear order of the vertices in a directed graph without cycles.

* Trees are connected undirected graphs without cycles.

* A spanning tree of a connected undirected graph is:
  - A subgraph that contains all the graph's vertices and enough of its edges to form a tree.

* A minimum spanning tree for a weighted undirected graph is
  - A spanning tree whose edge-weight sum is minimal.

* The shortest path between two vertices in a weighted directed graph is
  - The path that has the smallest sum of its edge weights.