GRAPH

- BFS → BFT (Queue)
- DFS → DFT (Stack)

Searching

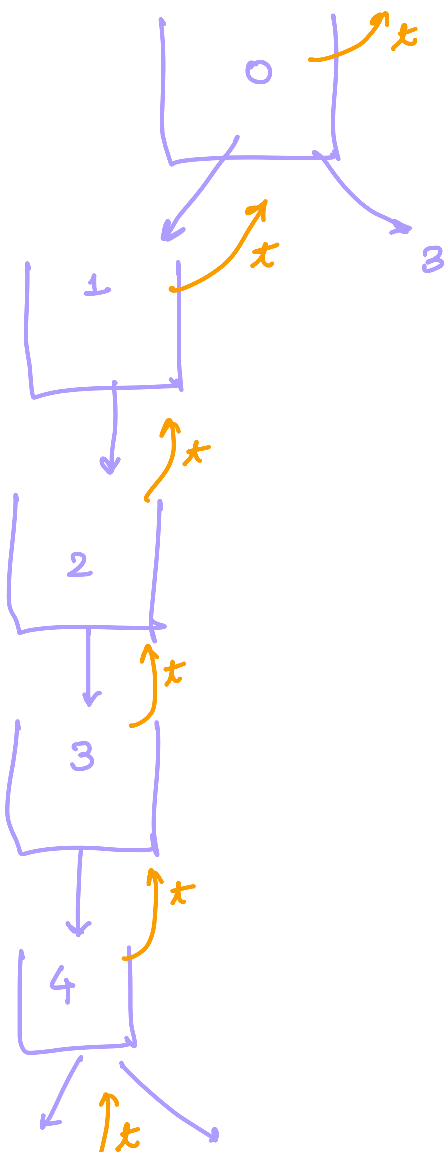
src → dst ?

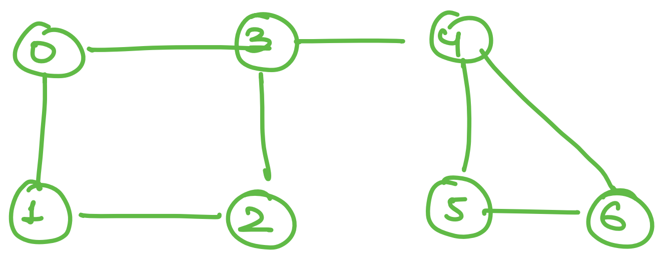
is Connected
isCyclic
is Tree
getCC

} Iteratively

DFS (Recursively)

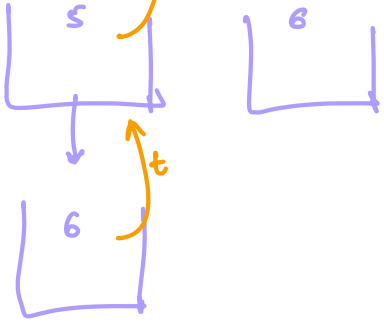
src = 0 dst = 6





0 ✓
1 ✓
2 ✓
3 ✓
4 ✓
5 ✓
6 ✓

- frame
- visited
- dst ?
- nbrs call | unvisited



DFT (Recursion)

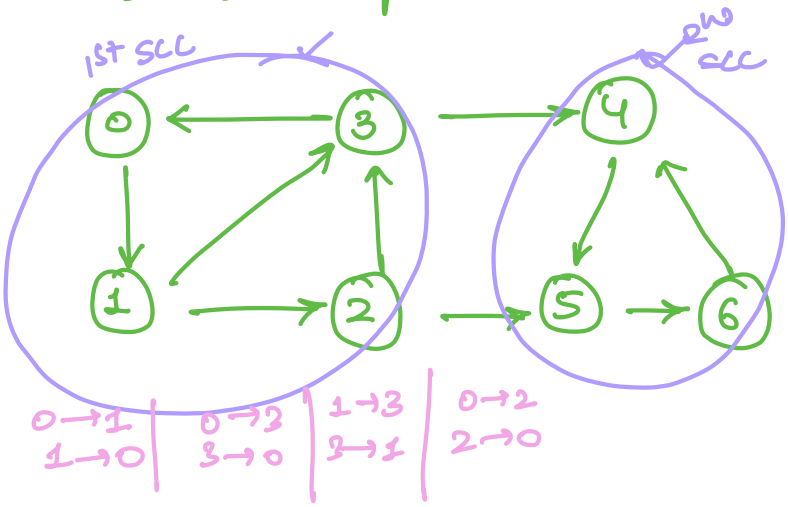
0 ——— V

——— if (V visited ignore)

DFS logic

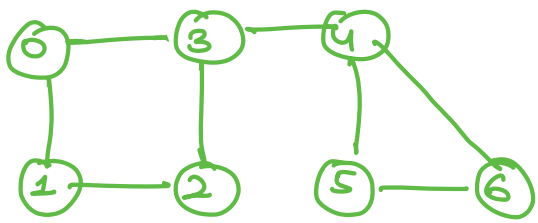
Strongly Connected Component

→ Directed Graph



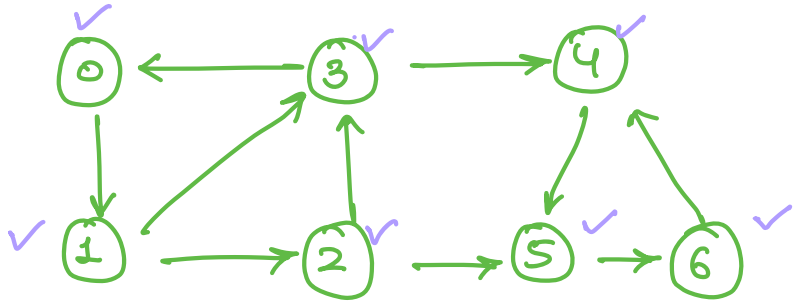
Connected Component

→ Undirected Graph



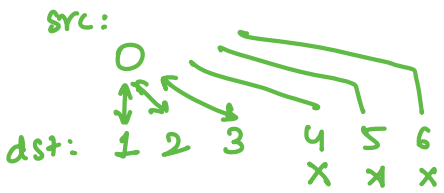
SCC:

A subgraph of a directed graph is called as SCC if and only if for every pair of vertices A and B there exist a path from A to B and path from B to A.



STC RST:

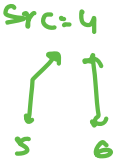
0 → 1 2 3 4 5 6



[0 1 2 3]

1 → 23456
2 → 3456
3 → 456
4 → 56
5 → 6

src: 1 ✓ scc part : ignore
src : 2 ignore
src : 3 ignore



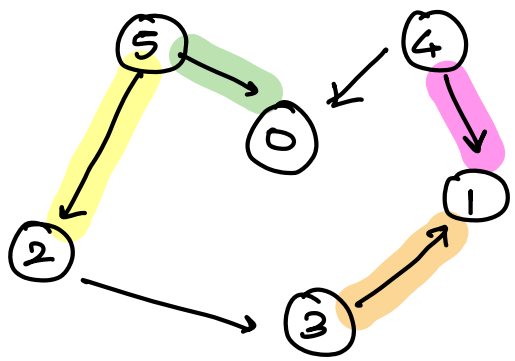
[4 5 6]

src: 5
src: 6
| ignore

GRAPH: dfs, bfs, dft, bft, isTree, isCyclic, getCC, isConnected & its application.

Topological Sorting:

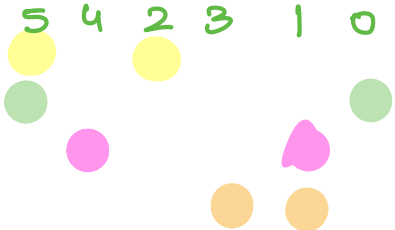
→ Directed Acyclic Graph (DAG)



u → v

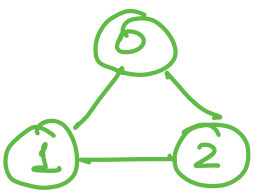
TS: u should come before v.

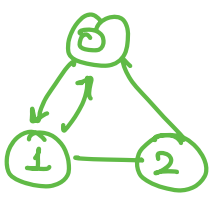
TS: 5 4 2 3 1 0



DFT ? 5 2 3 1 0 4 } but not TS

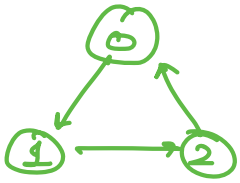
Undirected Graph ?



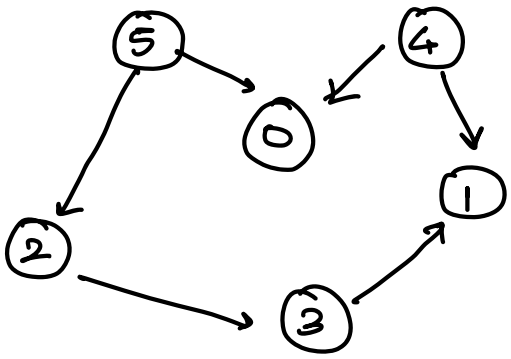


0 → 1 01
1 → 0 10

No Cycle ?



2 0 1 2



indegree : incoming edges

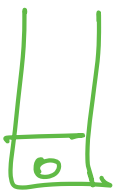
outdegree : outgoing edges

Vertex	Outdegree (sort)
0	0
1	0
2	1
3	1
4	2
5	2

indegree: 0
outdegree: map

Visited
0 ✓
1 ✓
2 ✓
3 ✓
4
5

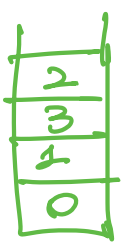
0 =



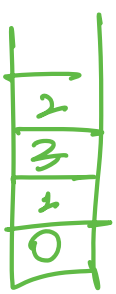
1 =

- visited
- visit (unvisited)
- dec path

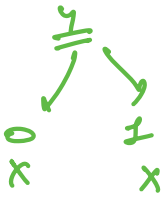
2 =
3

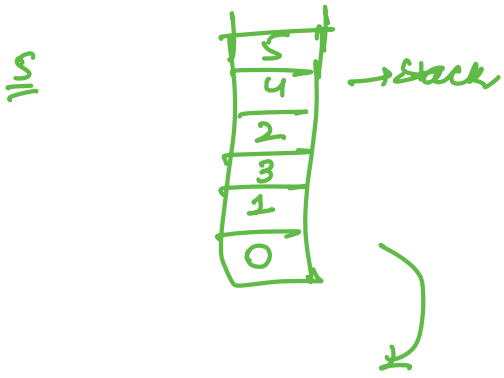


3 =
ignore



4 =





5 4 2 3 1 0

Hashing

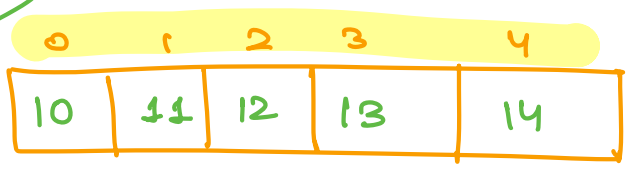
→ insertion, deletion, find } O(1)

vector/array : [10 20 30 40 80 15 20]

find(30) ? O(n)

find(15) ? O(n)

Keys: 10 11 12 13 14 hash fx^n: %5

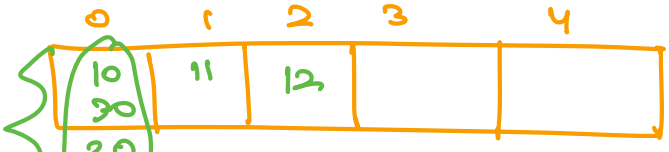


10%5: 0
11%5: 1
12%5: 2
13%5: 3
14%5: 4

Hash fx^n:

- keys % no of buckets
- (key * constant) % no of buckets

Keys: 10 11 12 30 20 %5



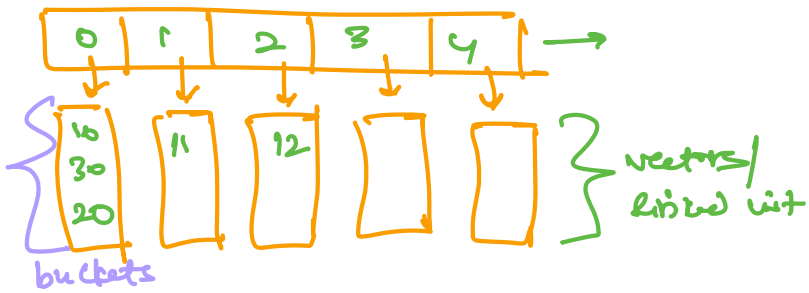
10 % 5 = 0

11 % 5 = 1

12 % 5 = 2

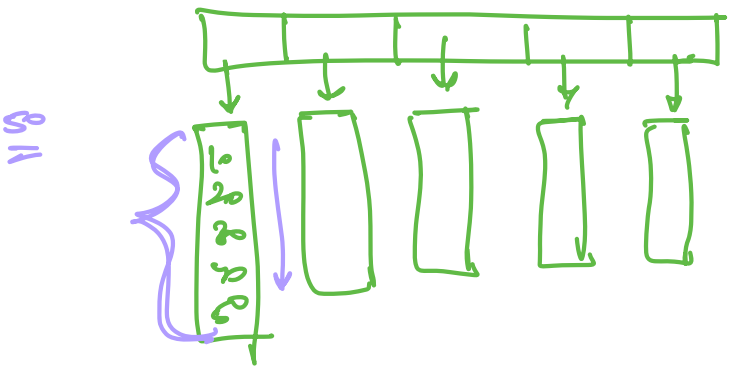
30 % 5 = 0

20 % 5 = 0



30 search?

→ 30 % 5 = 0 : search for 30 in 0th bucket only.



Collision Reduction?

1. Open Addressing → Linear Probing : Empty slot search sequentially
   → Quadratic Probing : Quadratic fashion

2. Closed Addressing
   ↓
   Chaining.

Linear Probing:

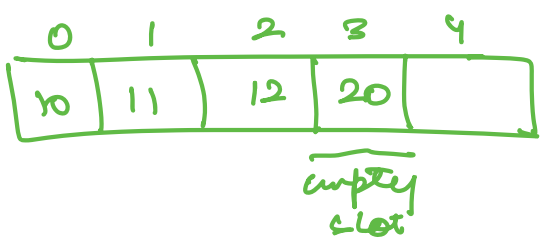
keys: 10 11 12 20 % 5

10 % 5 = 0

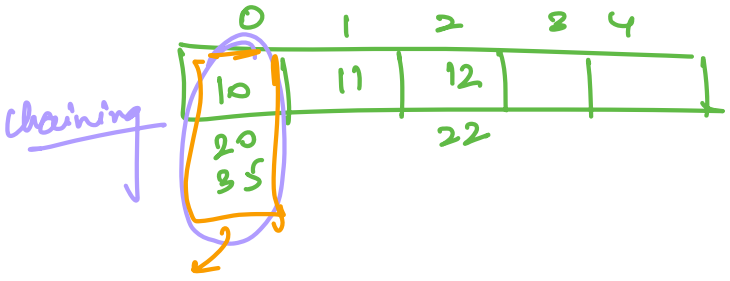
11 % 5 = 1

12 % 5 = 2

20 % 5 = 0



Chaining:



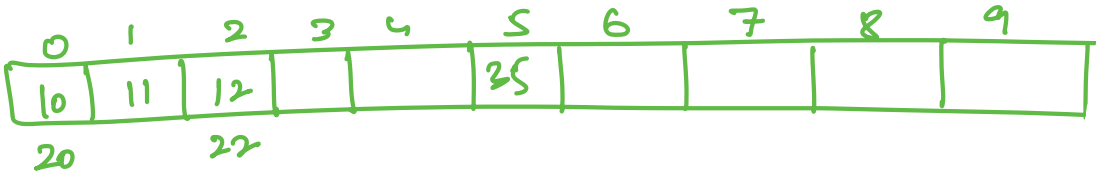
find(20) → 20 % 5 = 0 → 0th bucket.

bucket constant 1, 2, 3 —

Avg load = $\frac{6}{5}$

avg load > constant 1, 2, 3...

(rehashing)



Avg load = $\frac{6}{10}$ = 0.6 < 1 ✓

Java: hash map

C++: map