from COMP20007-2024 Workshop Week 10

1	Floyd's Algorithm for APSP (*S*D search in weighted graphs) Q10.1, Q10.2
2	Transitive Closure of di-graphs, Warshall's algorithm, Q10.3
3	Hashing
LAB	Assignment 2: Q&A and/or implementing W10.3, W10.4

Floyd Algorithm – APSP (APSP == *S*D)

Floyd Algorithm aka. Floyd-Warshall Algorithm

The Task:

- Given a weighted graph G=(V,E,w(E))
- Find shortest path (path with min weight) *between all pairs of vertices. (*S*D)*

?:

- can we use Dijkstra's Algorithm for the task?
- why Floyd-Warshall's ?

Floyd Algorithm: DP for APSP

parameters & sub-problems:

- Problem: shortest path between all pairs (i,j)
- Value to optimize: a matrix D of V rows, V cols
- DP[k]= D^k: matrix of shortest path values after using k nodes 1..k as the transit points

Recurrence:

D0 = ?

Dk = ?

Floyd Algorithm: DP for APSP

parameters & sub-problems:

- Problem: shortest path between all pairs (i,j)
- Value to optimize: a matrix D of V rows, V cols
- DP[k]= D^k: matrix of shortest path values, using k nodes 1..k as the transit points

Recurrence:

$$D_{ij}^{0} := W_{ij}, \qquad D_{ij}^{k} := \min\left\{D_{ij}^{k-1}, D_{ik}^{k-1} + D_{kj}^{k-1}\right\}$$

Note: D^k only depends on D^{k-1} → just transform D, no need to have array of D

```
D := W      // W is the weigjht matrix of the graph
for each node k in 1..V:
  // use k as a stepstones
  for each pair (i,j):  // for (i=...) for (j=...
        if D_ik + D_kj <  D_ij
              D_ij := D_ik + D_kj
```

Floyd Algorithm

$$D^0_{ij} := W_{ij}, \qquad D^k_{ij} := \min\left\{D^{k-1}_{ij}, D^{k-1}_{ik} + D^{k-1}_{kj}\right\}$$

Conditions:

- directed or undirected ?
- weighted or unweighted?
- can be applied to negative weights?

Data structures / Graph representation = adjacency matrix

Complexity = O(?) or Θ (?)

Comparing with repeating Dijkstra (DA) for the same task:

- Applicability
- Complexity

Floyd Algorithm

$$D^0_{ij} := W_{ij}, \qquad D^k_{ij} := \min\left\{D^{k-1}_{ij}, D^{k-1}_{ik} + D^{k-1}_{kj}\right\}$$

Conditions:

- directed or undirected ? both
- weighted or unweighted: weighted, for unweighted: set edge weight to 1
- negative weights possible, but NOT negative cycles

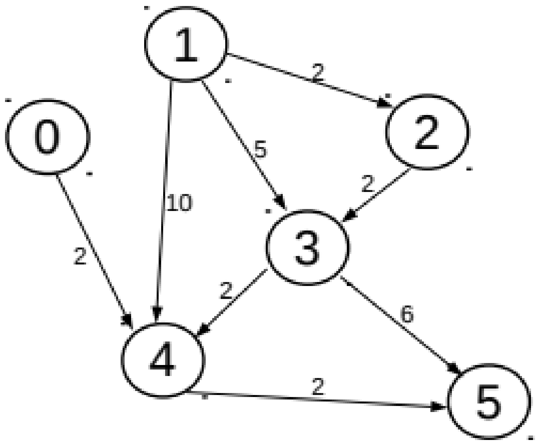
Data structures / Graph representation = `adjacency matrix`

Complexity = $\Theta(V^3)$

Comparing with repeating Dijkstra:

- Both are used for shortest paths only [NOT for longest path]
- Floyd allows negative weights, Dijkstra not
- Complexity of repeating DA is O(V(V+E)logV)

Step-by-step Example: Tracing Floyd for a graph



Trace the Floyd algorithm.

Step i= 0, 1, 2, 3, 4, 5

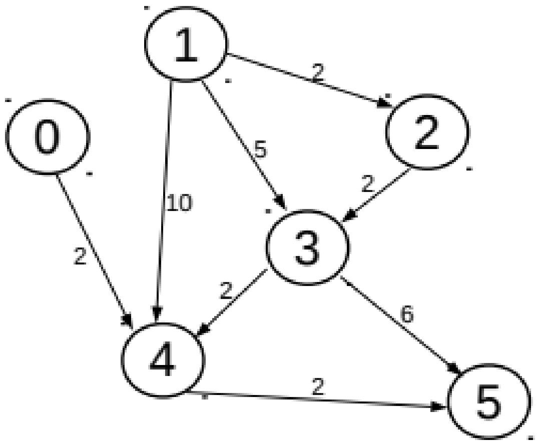
empty cell for ∞
(note `A[s][s]`
should be zero)

--------------------TO (`t`) -----------------------

FROM (`s`)

	0	1	2	3	4	5
0	0				2	
1		0	2	5	10	
2			0	2		
3				0	2	6
4					0	2
5						0

Tracing Floyd Algorithm



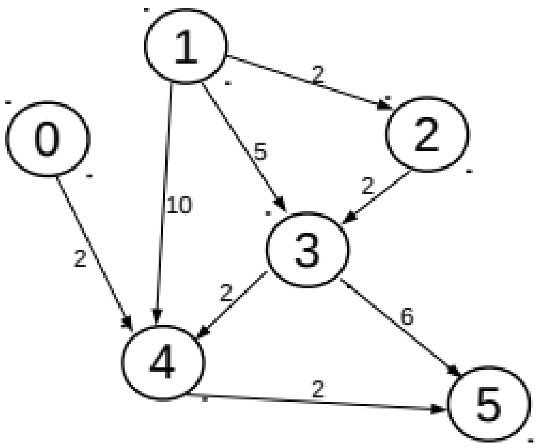
Notes:

- when 0, 1, or 5 is used as an intermediate, no change is possible (why?)

	0	1	2	3	4	5
0	0				2	
1		0	2	5	10	
2			0	2		
3				0	2	6
4					0	2
5						0

Tracing Floyd Algorithm

k= **2** as the stepstone: select rows **2** and column **2** as references



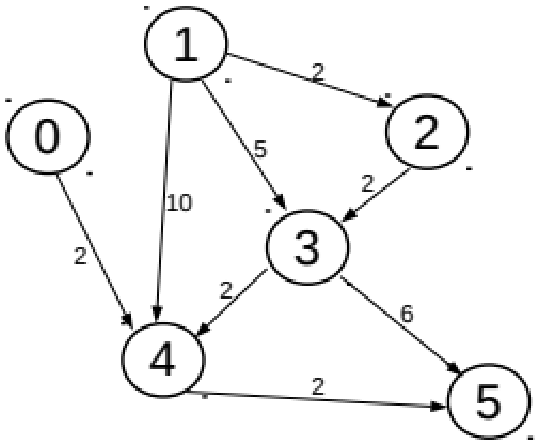
column **2** gives paths "from s to **2**"

Only this cell need to be considered. Why?

row **2** gives paths "from **2** to t"

	0	1	**2**	3	4	5
0	0				2	
1		0	2		10	
2			0	2		
3				0	2	6
4					0	2
5						0

Tracing Floyd Algorithm



k= **2** as the stepstone

	0	**1**	**2**	**3**	**4**	**5**
0	0				2	
1		0	2	4	10	
2			0	2		
3				0	2	6
4					0	2
5						0

Tracing Floyd Algorithm

No change at k= 0 and k= 1

k= **2** as the stepstone

	0	1	2	3	4	5
0	0				2	
1		0	2	4	10	
2			0	2		
3				0	2	6
4					0	2
5						0

k= **3** as the stepstone

	0	1	2	3	4	5
0	0				2	
1		0	2	4	6	10
2			0	2	4	8
3				0	2	6
4					0	2
5						0

k= **4** as the stepstone, done (no change at k= 5)

	0	1	2	3	4	5
0	0				2	4
1		0	2	4	6	8
2			0	2	4	6
3				0	2	4
4					0	2
5						0

Q10.3: manual exec of Floyd's

Perform Floyd's algorithm on the graph given by the following weights matrix:

$$W = \begin{bmatrix} 0 & 3 & \infty & 4 \\ \infty & 0 & 5 & \infty \\ 2 & \infty & 0 & \infty \\ \infty & \infty & 1 & 0 \end{bmatrix}.$$

$$D^0_{ij} := W_{ij}, \qquad D^k_{ij} := \min\left\{D^{k-1}_{ij}, D^{k-1}_{ik} + D^{k-1}_{kj}\right\}$$

Q 10.3: Check your answer

$$D^0_{ij} := W_{ij}, \qquad D^k_{ij} := \min\left\{D^{k-1}_{ij}, D^{k-1}_{ik} + D^{k-1}_{kj}\right\}$$

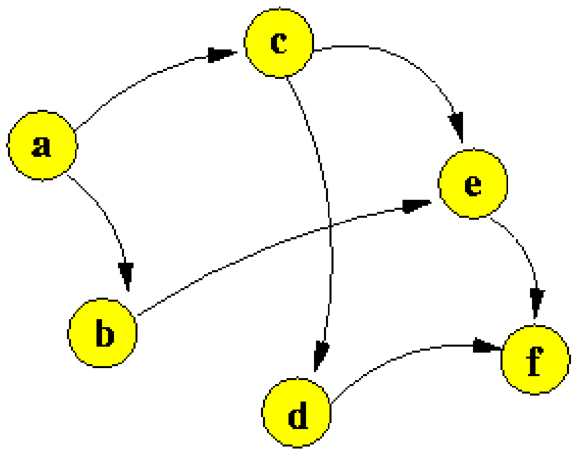
$$D^0 = \begin{bmatrix} 0 & 3 & \infty & 4 \\ \infty & 0 & 5 & \infty \\ 2 & \infty & 0 & \infty \\ \infty & \infty & 1 & 0 \end{bmatrix} \qquad D^1 = \begin{bmatrix} 0 & 3 & \infty & 4 \\ \infty & 0 & 5 & \infty \\ 2 & \mathbf{5} & 0 & \mathbf{6} \\ \infty & \infty & 1 & 0 \end{bmatrix}$$

$$D^2 = \begin{bmatrix} 0 & 3 & \mathbf{8} & 4 \\ \infty & 0 & 5 & \infty \\ 2 & 5 & 0 & 6 \\ \infty & \infty & 1 & 0 \end{bmatrix} \qquad D^3 = \begin{bmatrix} 0 & 3 & 8 & 4 \\ \mathbf{7} & 0 & 5 & \mathbf{11} \\ 2 & 5 & 0 & 6 \\ \mathbf{3} & \mathbf{6} & 1 & 0 \end{bmatrix} \qquad D := D^4 = \begin{bmatrix} 0 & 3 & \mathbf{5} & 4 \\ 7 & 0 & 5 & 11 \\ 2 & 5 & 0 & 6 \\ 3 & 6 & 1 & 0 \end{bmatrix}$$

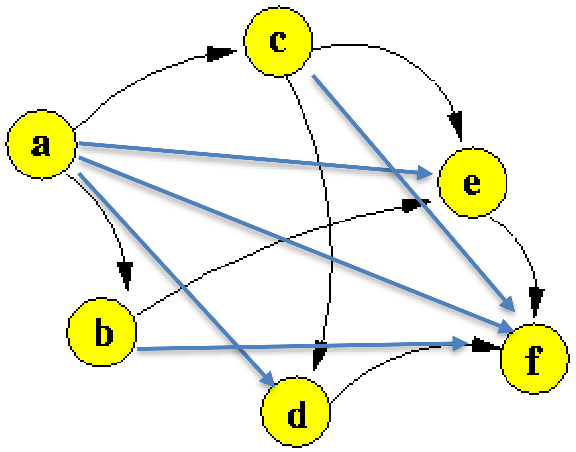
Transitive Closure of digraphs

Understanding:

- Transitive Closure of a di-graph
- Why not for undirected graph?



Directed graph G



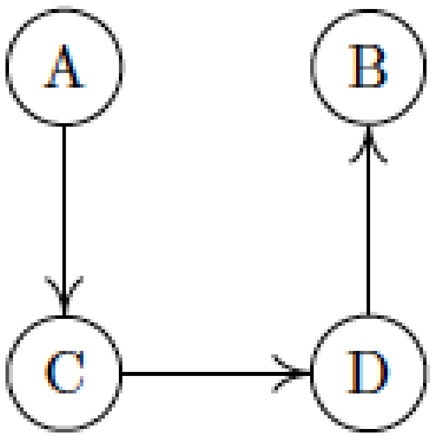
G' = Transitive Closure of G

Edge u→v in G' means v is reachable from u in G

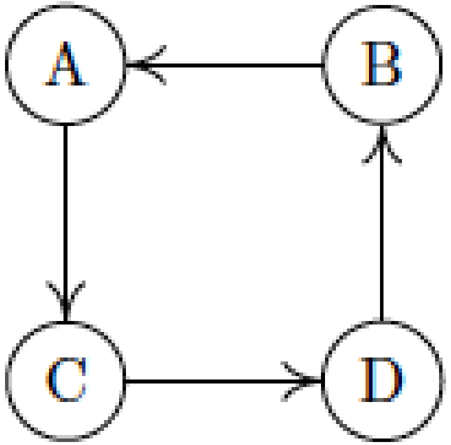
Q10.1: Transitive Closure of digraphs

Draw the transitive closure of the following two graphs:

(a)



(b)



Warshall's Algorithm: a DP algo for Transitive Closure

Input:
- adjacent matrix A[1..n][1..n] of an input graph G

Output:
- adjacent matrix of the transitive closure of G

Main argument:
- transitiveness: if there are paths $i \rightarrow k$ and $k \rightarrow j$, then there is path $i \rightarrow j$ which uses k as an intermediate node ($i, j, k \in 1..n$)

Warshall's Algorithm: Solve the problem using DP, similar to APSP. First decide:
- parameters & sub-problems:

  R^k = matrix of connectivity after using nodes 1..k as intermediates
- recurrence:

$$R^0_{ij} := A_{ij}, \qquad R^k_{ij} := R^{k-1}_{ij} \text{ or } \left(R^{k-1}_{ik} \text{ and } R^{k-1}_{kj}\right)$$

Q 10.2: Manually Tracing the Warshall's Algorithm

$$R^0_{ij} := A_{ij}, \qquad R^k_{ij} := R^{k-1}_{ij} \text{ or } \left(R^{k-1}_{ik} \text{ and } R^{k-1}_{kj}\right)$$

only cell with both non-zero references can change

$$R^0 = A = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Set R= A, R is R^0
transition from 0 to 1 by:
- looking at for all possible ij
- it can be done by using column 1 and row 1 as references

$$\begin{array}{cc} \text{row } k=1 & \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & \quad & \quad \\ 0 & 0 & 0 & 0 \end{bmatrix} \\ & \text{column } k=1 \end{array}$$

similarly for any transition from R^{k-1} to R^k: use row k and column k as references.

$$\begin{array}{cc} \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} & \text{row } k \\ \text{column } k & \end{array}$$

hashing: introductory exercise

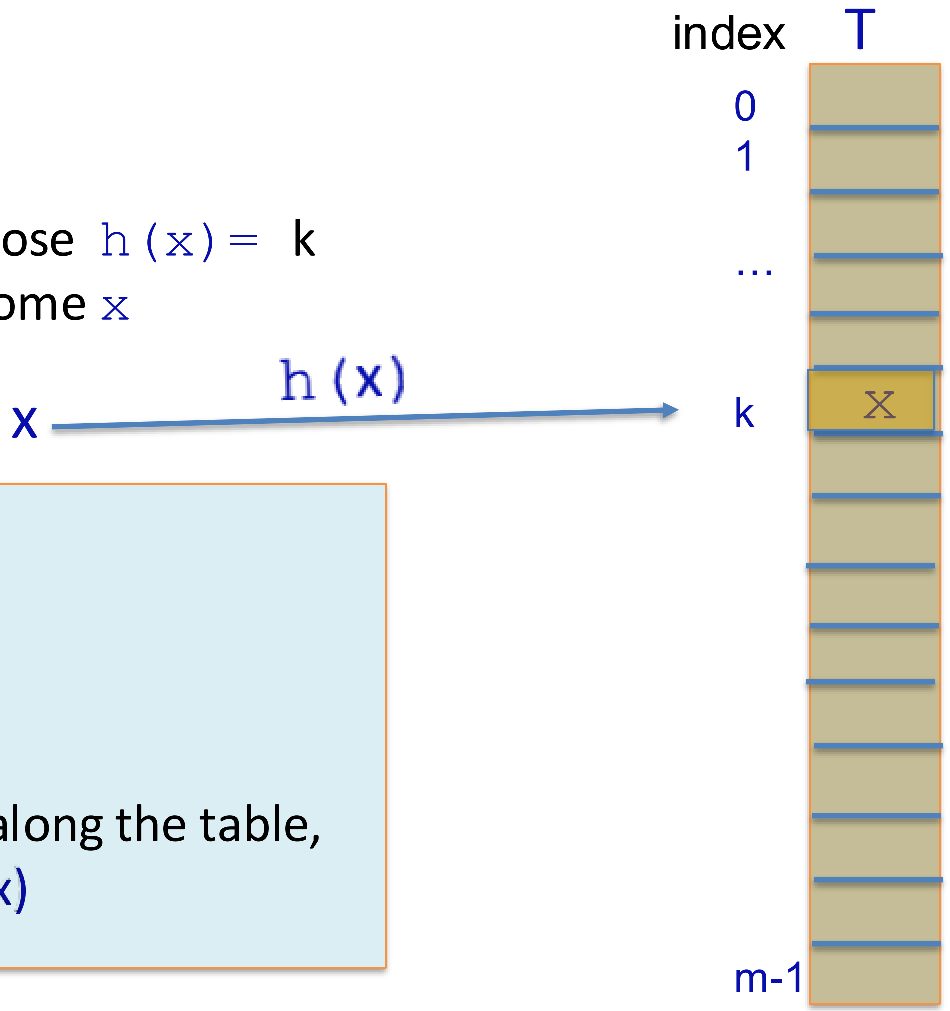
Using array A[0..99], how to keep n<100 distinct integer keys for efficient search and insert, if :

a) the keys are in the range 0..99
b) the keys are in the range 200..299
c) the keys are in the range 0..299

Hash Table: dictionary with average O(1) search/insert

Hashing= hash table T + *hash function* h(x) : store key x at T[h(x)]

suppose h(x) = k
for some x



- T[0..m-1]: a *hash table of size* m
- T[i] : a *hash slot*, or *hash bucket*
- h(x) : hash function, that
    - $0 \leq h(x) < m$
    - distributes keys evenly (uniformly) along the table,
    - is efficient: ($\theta(1)$) for computing h(x)

Collisions

Collision when `h(x1) = h(x2)` for some `x1≠ x2`.

Examples:

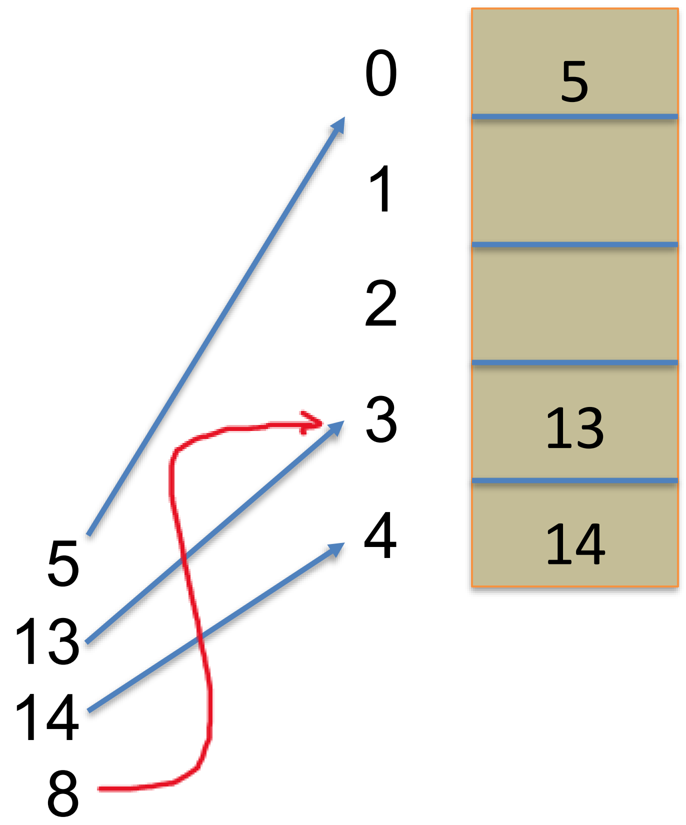
$h(x) = x \bmod 100$

insert $x1 = 1$, $x2 = 101$ → collision

Collisions are normally unavoidable.

Collisions

Example:
m=5, h(x)= x% m
Here: h(8) == h(5)



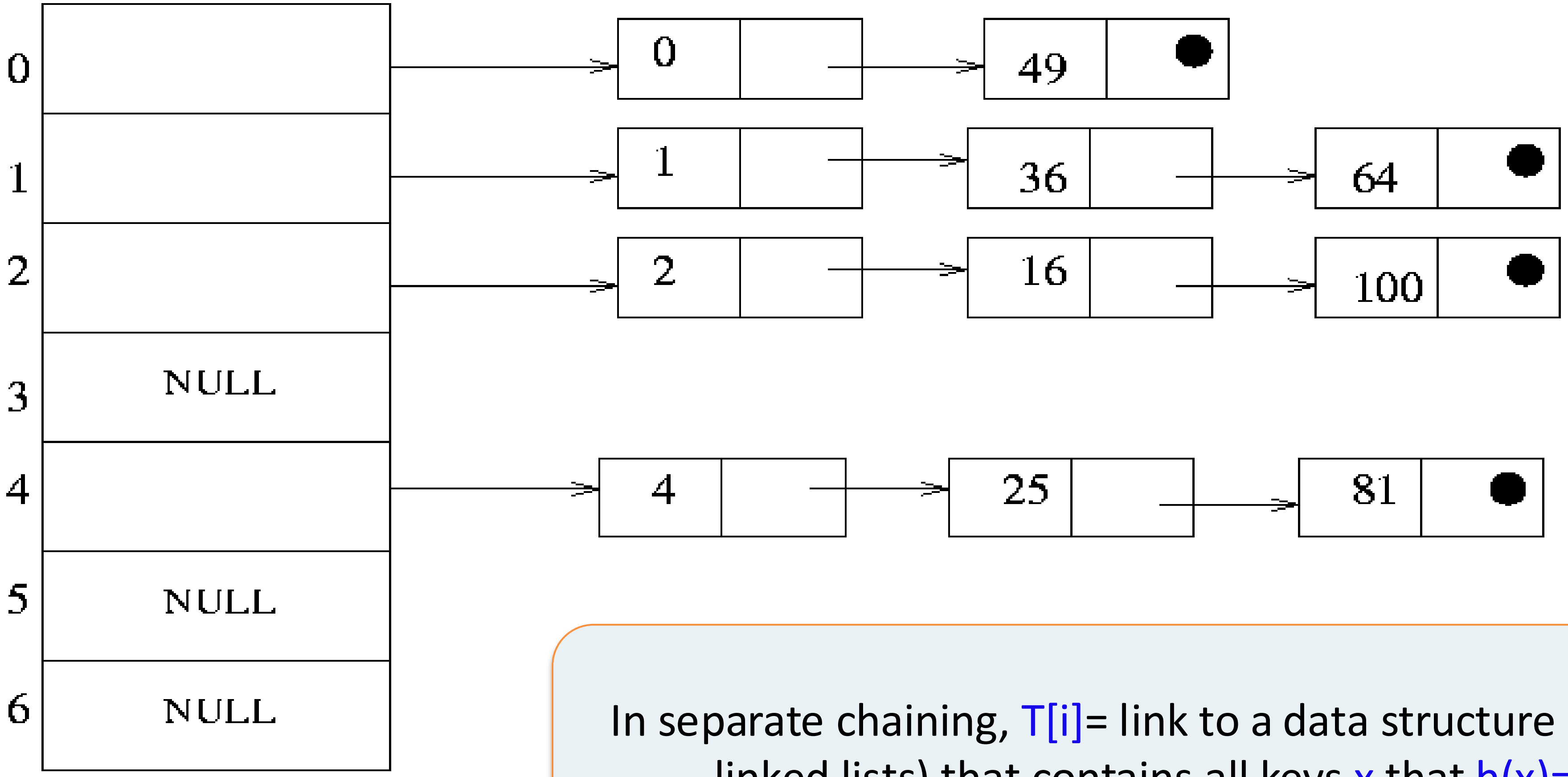
Methods *to reduce collisions:*

- using *a prime number* for hash table size m.
- making the table size m *big*

Even though, collisions might still happen

Collision Solution 1: Separate Chaining

`h(x) = x % 7,` keys entered:
`0, 1, 49, 2, 36, 4, 64, 16, 25, 100, 81`



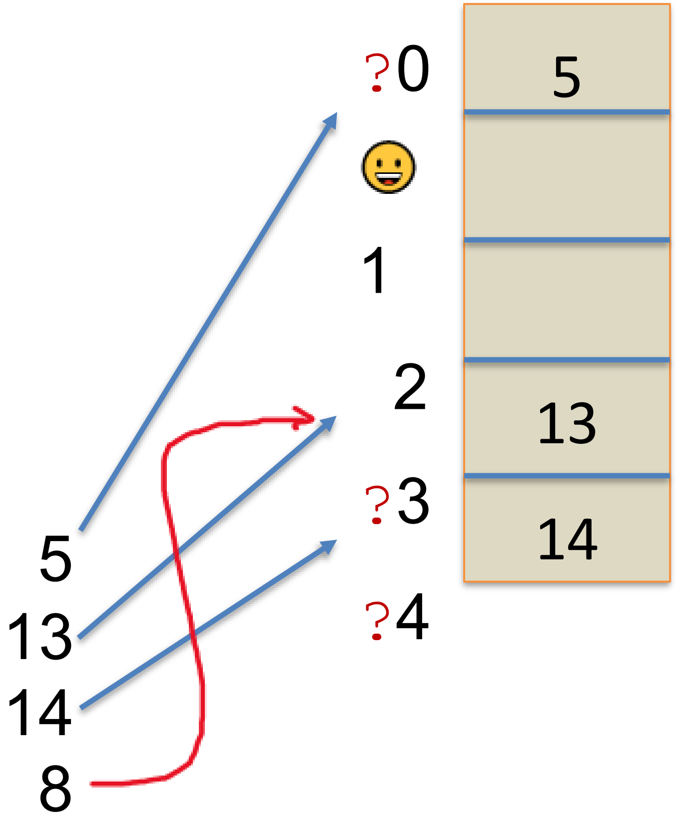
In separate chaining, T[i]= link to a data structure (such as linked lists) that contains all keys x that h(x)== i

Solution 2: Linear Probing (here, data are kept in the hash table)

linear probing= *when colliding, use the successive empty bucket*.

Example: m=5, h(x)= x mod m ,

keys inserted: 5, 13, 4, 8



Hashing with linear probing:

- When inserting we do some probes until getting a vacant slot:
  h(x) replaced by
  H(x, probe)= (h(x) + probe) mod m
  where probe is 0, 1, 2 …
  ie. *examining forward with step= 1 until reaching a vacant slot*
- The same procedure for search
- Deletion is problematic! (why?)

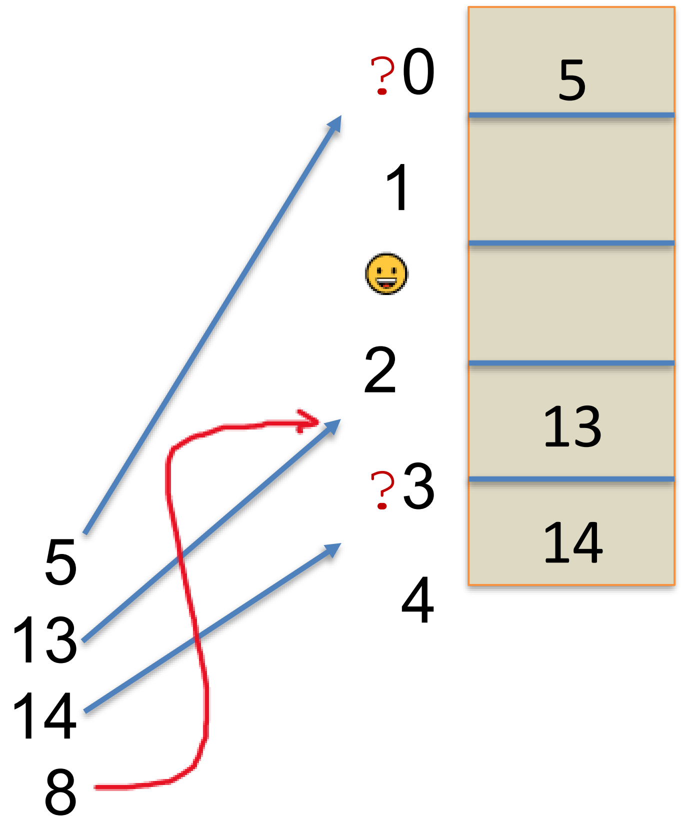
Double Hashing

When colliding, *look forward for empty cells at distance* `h2(x)`, *ie. examing forard with constant step=* `h2(x)`

Example: m=5, h(x)= x mod m ,

h2(x)= x mod 3 + 1 ,

keys inserted: 5, 13, 4, 8



Hashing with double hashing:

similar to *linear probing*, but employ a second hash function h2(x):

H(x,probe)= (h(x)+ probe***h2(x)**) mod m

where probe is 0, 1, 2, … (until reaching a vacant slot).

Note that:

- **h2(x)** ≠ 0 for all x, (why?)
- to be good, **h2(x)** should be co-prime with m, (how?)

Note: *linear probing* is just a special case of *double hashing* when **h2(x)**=1.

Both linear probing and double hashing are referred to as *Open Addressing methods*.

Q 10.4, 10.5, 10.6 [Group/Individual]

Q 10.4: [Separate chaining] Consider a hash table in which the elements inserted into each slot are stored in a linked list. The table has a fixed number of slots `L=2`. The hash function to be used is `h(k)=k mod L`.

a) Show the hash table after insertion of records with the keys
`17 6 11 21 12 33 5 23 1 8 9`

b) Can you think of a better data structure to use for storing the records in each slot?

Q 10.5 [Open Addressing]: Consider a hash table in which each slot can hold one record and additional records are stored elsewhere in the table using linear probing with steps of size `i=1`. The table has a fixed number of slots `L=8`. The hash function to be used is `h(k)=k mod L`.

a) Show the hash table after insertion of records with the keys
`17 7 11 33 12 18 9`

b) Repeat using linear probing with steps of size `i = 2`. What problem arises, and what constraints can we place on `i` and `L` to prevent it?

c) Can you think of a better way to find somewhere else in the table to store colliding keys?

Q 10.6 [double hashing]: Consider a hash table in which each slot can hold one record and additional records are stored elsewhere in the table using double hashing. The table has a fixed number of slots $L = 13$.
The hash function to be used is $h(k) = k \mod 13$. The second hash function to be used is $h2(k) = k \mod 5 + 1$.

Show the hash table after insertion of records with the keys $14 \quad 30 \quad 17 \quad 33 \quad 55 \quad 31 \quad 29 \quad 16$

Assignment 2: Q&A

Additional Slides

Floyd-Warshall Algorithm: How to retrieve the path between i-->j ?

In DP, in addition to matrix `dist[i][j]=` shortest path length from `i` to `j`, also maintain

matrix `stone[i][j]=` the decision made for the pair `(i,j)`

`=` the first stop on the path `i→j`

Algorithm:

```
for each node k in V:
    for each pair (i,j):  // for (i=...) for (j=...
      if  (dist[i][k]+dist[k][j] < dist[i][j]){  // if new path is shorter
          dist[i][j]= dist[i][k]+dist[k][j]       // ... take it, update dist
          stone[i][j]= k                          // ... and update stone
      }
```

Q10.2: Check your answer

$$R^0_{ij} := A_{ij}, \qquad R^k_{ij} := R^{k-1}_{ij} \text{ or } \left(R^{k-1}_{ik} \text{ and } R^{k-1}_{kj}\right)$$

$\Leftrightarrow$ if R_{ij}=0, change it to 1 iif R_{ik} and R_{kj} are both 1

$$R^0 = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$R^1 = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$R^2 = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

no change
col 2 is 0

$$R^3 = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$B := R^4 = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

no change
row 4 is 0

Check: Q 10.4: Separate chaining

Consider a hash table in which the elements inserted into each slot are stored in a linked list. The table has a fixed number of slots `L=2`. The hash function to be used is `h(k)=k mod L`.

a) Show the hash table after insertion of records with the keys
`17 6 11 21 12 33 5 23 1 8 9`

b) Can you think of a better data structure to use for storing the records in each slot?

Your solution & notes:

a)

$$\begin{array}{c|l} 0 & 6 \to 12 \to 8 \\ 1 & 17 \to 11 \to 21 \to 33 \to 5 \to 23 \to 1 \to 9 \end{array}$$

b) array/sorted array? balanced search trees such as AVL or 2-3 tree?

but why the above hashing is so bad? perhaps we should solve a more general problem: how to make that hashing better, including increasing table size (with a prime number), changing hash function.

Check: Q 10.5: Open addressing

Consider a hash table in which each slot can hold one record and additional records are stored elsewhere in the table using linear probing with steps of size `i=1`. The table has a fixed number of slots `L=8`. The hash function to be used is `h(k)=k mod L`.

a) Show the hash table after insertion of records with the keys
`17 7 11 33 12 18 9`

b) Repeat using linear probing with steps of size `i = 2`. What problem arises, and what constraints can we place on `i` and `L` to prevent it?

c) Can you think of a better way to find somewhere else in the table to store colliding keys?

Your solution & notes:

a)

0	1	2	3	4	5	6	7
	17	33	11	12	18	9	7

b)

0	1	2	3	4	5	6	7
	17	18	11	12	33		7
	9?		9?		9?		9?

this is double hashing with h2(x)=2, trouble because m and 2 are not co-prime, choose h2(x)= odd number such as 3 would help!

c) double hashing with h2(x)≠ 0 and h2(x) co-prime with m, such as: m prime and h2(x)<m, or m=2^k, h2(x) odd