__/__/____

FIBONACCI HEAP

Characteristics

1. Collection of trees and trees may be in any order
2. Set of heap-ordered trees
3. Maintains a pointer to minimum element
4. Siblings are connected through a circular linked list
5. Each child points to its parent
6. Each parent points to any child
7. Set of marked nodes
8. Degree (x) - Number of children of root of tree

Inserting a node

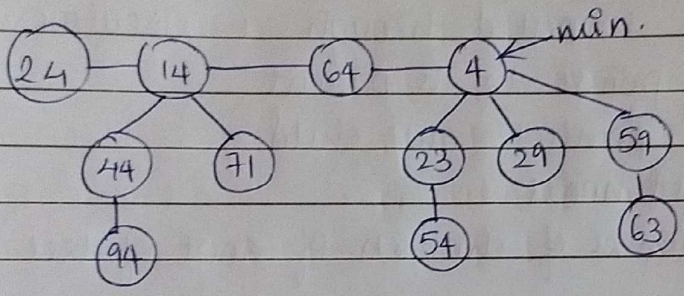
```
1  FIB-HEAP-INSERT (H, x)
2  degree [x] ← 0
3  p[x] ← NIL
4  child [x] ← NIL
5  left [x] ← x
6  right [x] ← x
7  mark [x] ← false
8  Concatenate root list containing x with root list H
9  if min [H] = NIL or key [x] < key [min [H]] then
10 min [H] ← x
11 n[H] ← n[H] + 1
```

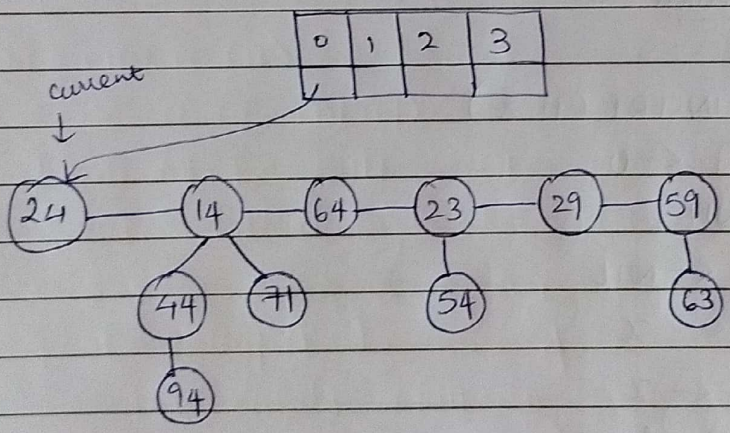
```
FIB-HEAP-INSERT [H, x]
1. Set all pointers and degree of x
2. Concatenate the root list containing x to root list H
3. if min [H] = NIL or key [x] < key [min [H]] then
       min [H] ← x.
```

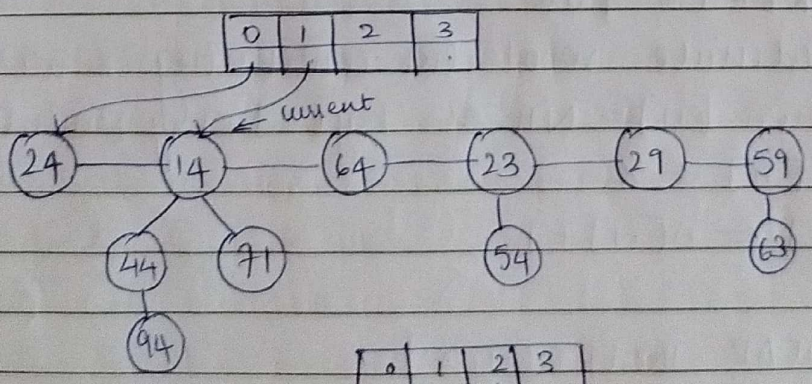
__/__/____

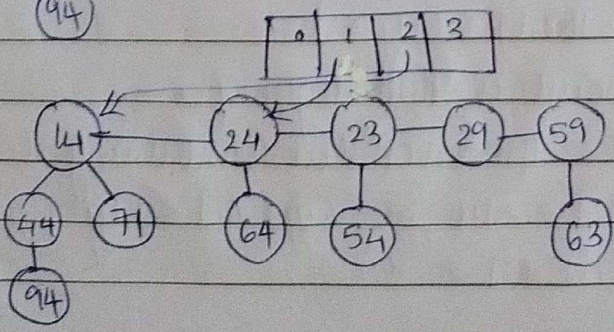
4. n[H] ← n[H]+1
   - Increase the size of fibonacci heap.

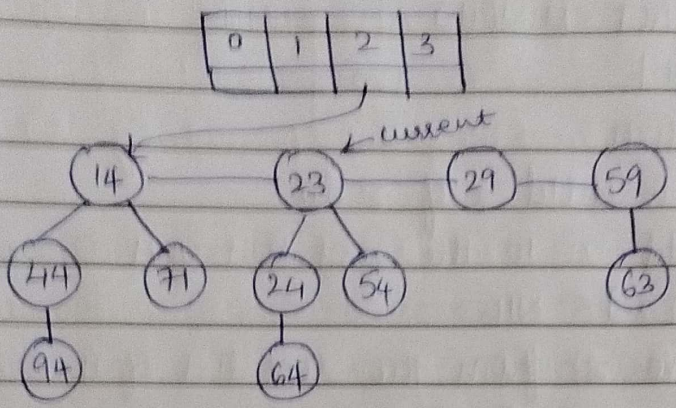
5. Illustrate extract min operation for the following fibonacci heap given below

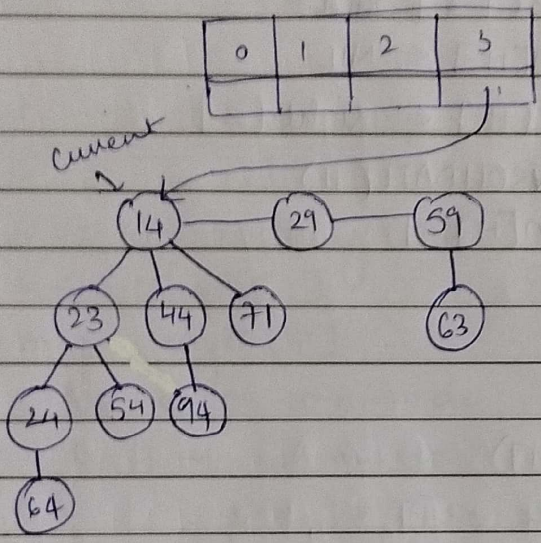


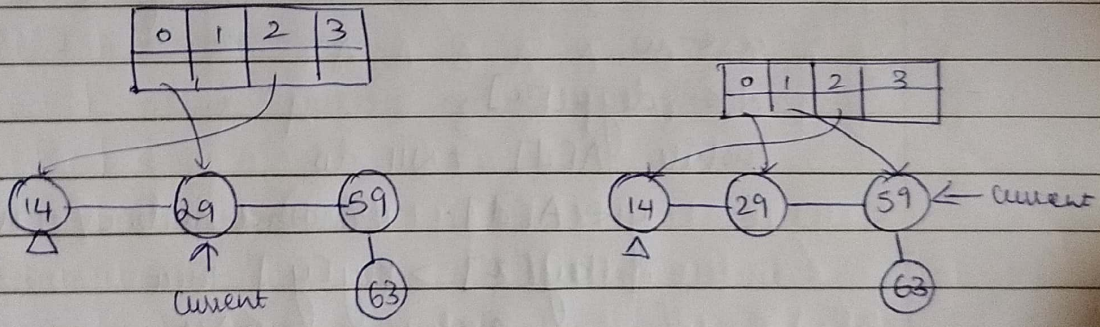







0
1
2
3
current
14
23
29
59
44
71
24
54
63
94
64


0
1
2
3
current
14
29
59
23
44
71
63
24
54
94
64


0
1
2
3
14
29
59
current
63
0
1
2
3
14
29
59
current
63

__/__/____

→ FIB-HEAP-EXTRACT-MIN(H)

```
z ← min[H]
if z ≠ NIL then
    for each child x of z do
        add x to the root list of H
        p[x] ← NIL
    remove z from the root list of H
    if z = right[z] then
        min[H] ← NIL
    else min[H] ← right[z]
        CONSOLIDATE(H)
    n[H] ← n[H] - 1
return z
```

→ CONSOLIDATE(H)

```
for i ← 0 to D(n[H])
    do A[i] ← NIL
for each node w in the root list of H do
    x ← w
    d ← degree[x]
    while A[d] ≠ NIL do
        y ← A[d]   // Another node w/ same degree as x
        if key[x] > key[y] then
            exchange x ↔ y
        FIB-HEAP-LINK(H, y, x)
        A[d] ← NIL
        d ← d+1
    A[d] ← x   // end of while
    min[H] ← NIL   // end of for.
for i ← 0 to D(n[H])
    do if A[i] ≠ NIL then
        add A[i] to the root list of H
```

__/__/____

if min[H] = NIL or
key[A[i]] < key[min[H]] then
min[H] ← A[i]

→ FIB-HEAP-LINK (H, y, x)
remove y from the root list of H
make y a child of x, incrementing degree[x]
mark[y] ← FALSE

NAIVE STRING MATCHING

```
naive-string matching (T,P)
    n ← length[T]
    m ← length[P]
    for s ← 0 to n-m do
    for j ← 0 to m-1 do
            if P[j] ≠ T[s+j] then
                break.
        if j == m then
            print "Pattern occurs with shift" s
```

Eg: S = abcab aabc abac
P = abaa
No of char comparisons = 9

__/__/____

KNUTH - MORRIS-PRAT (KMP) ALGORITHM

Find π value for the (longest prefix string = π) pattern given below

P: a b a b a c a

π: 0 0 1 2 3 0 1

↑

~~no of characters that match.~~

aba

a = a

ab ≠ ba

P: a a a a

0 1 2 3

aaaa aaa

a a a a

aa aa

aaaaa aa aa

P: M N O P Q

0 0 0 0 0

P: A A B A A C A A B A A

π 0 1 0 1 2 0 1 2 3 4 5

AA B AACAA B AA

^

AA

AAB

AABA

AABAA

AABAAC

AABAACA

AABAACAA

B

AB

AAB

CAAB

ACAAB

AACAAB

BAACAAB

ABAACAAB

P A A A C A A A A A C

0 1 2 0 1 2 3 3 3 4

COMPUTE-PREFIX-FUNCTION(P)

m ← length[P]

π[1] ← 0

k ← 0

for q ← 2 to m

do while k > 0 and P[k+1] ≠ P[q]

do k ← π[k]

if P[k+1] = P[q]

then k ← k+1

π[q] ← k

return π

__/__/__

Index	1	2	3	4	5	6	7	Initial
P	A	B	A	B	A	C	A	$k=0, \pi[1]=0$
LSP[](π)	0	0	1	2	3	0	1	$m=7$

Step	k	q	P[k+1]	p[q]	newk.	$\pi[q]$
1	0	2	A	B		$\pi[2]=0$
2	0	3	A	A	1	$\pi[3]=1$
3	1	4	B	B	2	$\pi[4]=2$
4	2	5	A	A	3	$\pi[5]=3$
5	3	6	B	C	$k=\pi[k]=1$	
6	1	6	B	C	$k=\pi[1]=0$	$\pi[6]=0$
7	0	7	A	A	1	$\pi[7]=1$

```
KMP-MATCHER(T, P)

n ← length[T]
m ← length[P]
π ← COMPUTE-PREFIX-FUNCTION(P)
q ← 0
for i ← to n
    do while q > 0 and p[q+1] ≠ T[i]
        do q ← π[q]
        if p[q+1] = T[i]
            then q ← q+1
        if q = m
            then print "pattern occurs w/ shift" i-m
            q ← π[q]
```

__/__/____

Q. Find the number of comparison to search for the given P in the T using Brute force approach & KMP

T: a b c a b c a b c x y c a b

P: b c a b c x y

Brute force number of comparisons = 16

P: b c a b c x y

π 0 0 0 1 2 0 0

1 2 3 4 5 6 7 8 9 10 11 12 13 14

a b c a b c a b c x y c a b

b c a b c x y

1 2 3 4 5 6 7

Q.

a b a c a a b a c c a b a c a b a a b b

a b a c a b

1 2 3 4 5 6

__/__/____

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
T:	a	b	a	c	a	a	b	a	c	a	b	a	c	a	b	a	a	b	b	

q:	1	2	3	4	5	6
p:	a	b	a	c	a	b
π:	0	0	1	0	1	2

Q. Apply KMP matcher algorithm for the text

T: A B C ✗ A B C D AB ✗ A B C D AB CD A B D E

P:	1	2	3	4	5	6	7
	A	B	C	D	A	B	D
	0	0	0	0	1	2	0

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
	A	B	C	✗	A	B	C	D	A	B	✗	A	B	C	D	A	B	C	D	A	B	D	E

	A	B	C	D	A	B	D
	1	2	3	4	5	6	7

A B C D A B D → i is shifted by 1

A B C D A B D

A B C D A B D

A B C D A B D

A B C D A B D

__/__/__

TYPE OF HASH FUNCTIONS

1. Division Technique

Collision: It occurs in hashing when two or more keys map to the same index (or location) in a hash table

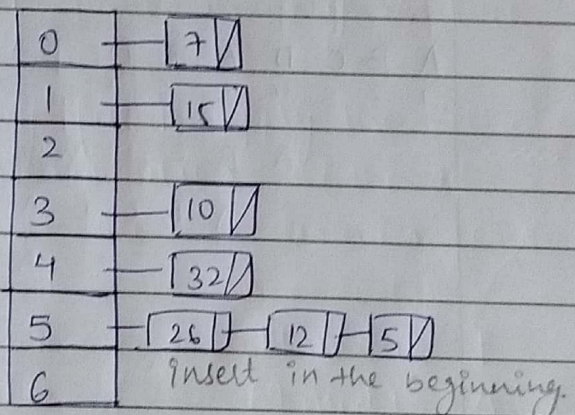
$h(k1) = h(k2)$ where $k1 \neq k2$

It means the hash func produces the same hash value ~~for~~

Load the keys 10, 20, 15, 7, 32, 5, 12, 26 into a hash table of size 7 using seperate chaining

hashfn = h(key) mod m

10 mod 7 = 3
20 mod 7 = 6
15 mod 7 = 1
7 mod 7 = 0
32 mod 7 = 4
5 mod 7 = 5
12 mod 7 = 5
26 mod 7 = 5



2. Open addressing

All keys are stored within the hash table itself. When a collision occurs, probing is used to find the next available slot.

3 probing techniques: Linear, Quadratic, Double

__/__/__

Q. Load the keys 23, 13, 21, 14, 7, 8, 15 in the same order into a hash table of the size 7 using open addressing (linear)

0	21
1	14
2	23
3	7
4	8
5	15
6	13

③ 15 mod 7 = 1
① 23 mod 7 = 2
② 21 mod 7 = 0
④ 14 mod 7 = 0 Collision occured.
⑤ 7 mod 7 = 0 Collision occured.
⑥ 8 mod 7 = 1
② 13 mod 7 = 6

we place it in the next available slot if collision occurs.

Quadratic probing

Define $h_0(K), h_1(K), h_2(K), h_3(K) \ldots$ where

$$h_i(K) = (hash(K) + i^2) \bmod size.$$

Q. Insert the keys 79, 69, 98, 72, 14, 50 into hashtable size 11 using quadratic probing

- $h(key) = key \bmod 11$

collision fun:

$h(key) = (hash\ val + i^2) \bmod$ table size

$(3+1^2) \bmod 11 = 4 \bmod 11 = 4$ 6
$6 \mid 1^2 \bmod 11 = 7 \bmod 11 = 7$ 7

0	-1	
1	-1	
2	-1	79
3	-1	69
4	-1	14
5	-1	
6	-1	72
7	-1	50
8	-1	
9	-1	
10	-1	98

79 mod 11 = 2
69 mod 11 = 3
98 mod 11 = 10
72 mod 11 = 6
14 mod 11 = 3
50 mod 11 = 6

__/__/____

Double Hashing

first hash → find initial pos.
2nd hash → if collision occurs, used to calculate step size for probing.

Q. Insert 79, 69, 98, 72, 14, 50 into hash table size 13
hash function: $h(k) = k \bmod 13$.
2nd hash function: $h_2(k) = 1 + (k \bmod 11)$

Steps.

1. Take the key you want to store on the hash-table
2. Apply the first hash func. h(key) over your key to get the location to store the key.
1. If the location is empty, place the key on that location
2. If the location is already filled, apply the second hash func h2(key) in combination of the 1st hash function h(key) to get the new location for the key

0	-1	
1	-1	79
2	-1	
3	-1	
4	-1	69
5	-1	14
6	-1	
7	-1	98
8	-1	72
9	-1	
10	-1	
11	-1	50
12	-1	

79 mod 13 = 1
69 mod 13 = 4
98 mod 13 = 7
72 mod 13 = 7 ✗ 8
14 mod 13 = 1
50 mod 13 = 11

collision
$h(k) = ((k \bmod 13) + i * (1 + k \bmod 11)) \bmod 13$
$(7 + 1 + 7) \bmod 13$
$(7 + (2 * 7)) \bmod 13$
$21 \bmod 13 = 8$

__/__/____

RABIN-KARP ALGORITHM.

Apply Rabin-Karp algorithm for the Text and the pattern given below.

T: 3 9 4 1 5 8 2 6 5 3 4
P: 2 6

hash func: p mod 11
hash value for pattern 26 = 26 mod 11 = 4

39 mod 11 = 6
94 mod 11 = 6
41 mod 11 = 8
15 mod 11 = 4. , spurious hit
58 mod 11 = 3
82 mod 11 = 5
26 mod 11 = 4 → match found.

How many false positives does the Rabin-Karp matches finds while matching strings c/ working model.

T: 3 1 4 1 5 9 2 6 5 3 5
p: 26
q: 11

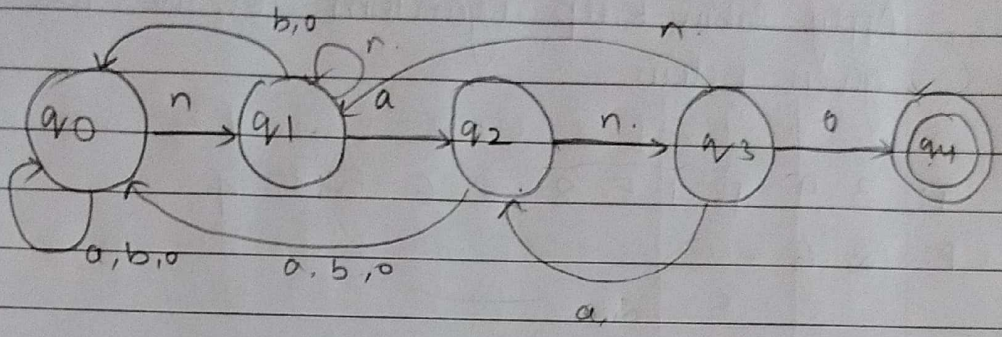
31 mod 11 = 9
14 mod 11 = 3
41 mod 11 = 8
15 mod 11 = 4 , SH
59 mod 11 = 4 , SH
92 mod 11 = 4 , SH
26 mod 11 = 4, match found.

False positives = 3
(Hash value matches but char does not)

__/__/____

FINITE AUTOMATA.

Q. Search for a pattern "nano" in a string "bananaona" using finite automata.



Transition Table.

State / Symbol	n	a	o
q_0	q_1	q_0	q_0
q_1	q_1	q_2	q_0
q_2	q_3	q_0	q_0
q_3	q_1	q_2	q_4
q_4	q_1	q_0	q_0

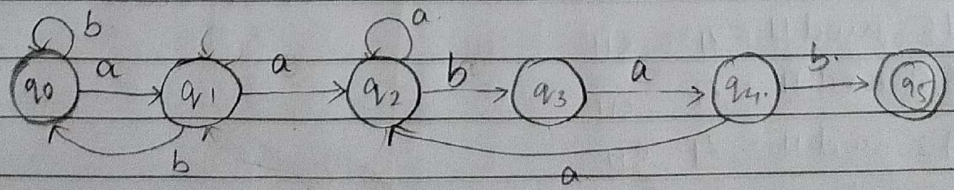
Text b a n a n a n o n a.

Initial State q_0 q_0 q_1 q_2 q_3 q_2 q_3 q_4 q_1 q_3 → match found.

Q. Apply finite automata model for the following text and patterns.

T: a a a b a b a a b a a b a b a a b

P: a a b a b.



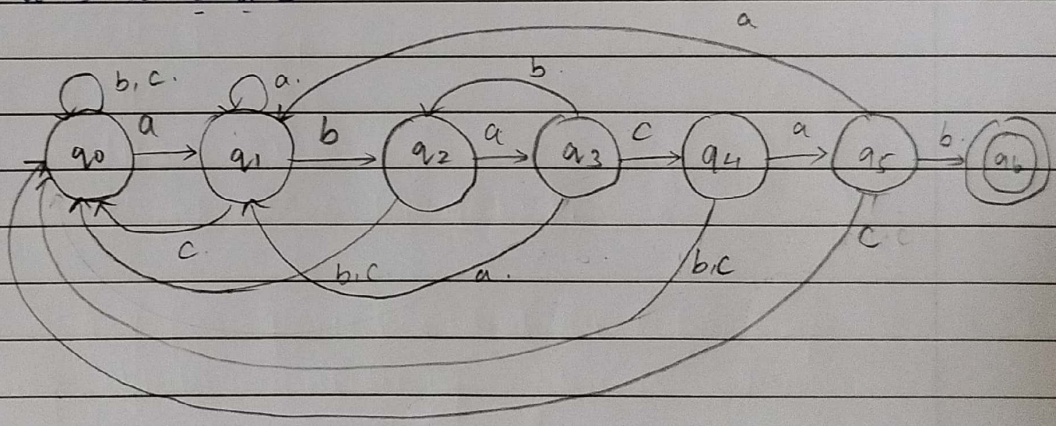
State/Symbol	a	b
q_0	q_1	q_0
q_1	q_2	q_0
q_2	q_2	q_3
q_3	q_4	q_0
q_4	q_2	q_5
q_5	q_1	q_0

T: a a a b a b a a b a a b a a b.

Initial state: q_0 q_1 q_2 q_2 q_3 q_4 (q_5) q_1 q_2 q_3 q_4 q_2 q_3 q_4 (q_5) q_1 q_2 q_3

Q. T: a b a c a a b a c c a b a c a b a a b b.

P: a b a c a b



State/Symbol	a	b	c
q_0	q_1	q_0	q_0
q_1	q_1	q_2	q_0
q_2	q_3	q_0	q_0
q_3	q_4	q_2	q_4
q_4	q_5	q_0	q_0
q_5	q_1	q_6	q_0
q_6	q_3	q_0	q_0

T: a b a c a a b a c c a b a c a b a a b b

q_0: q_1 q_2 q_2 q_4 q_5 q_1 q_2 q_3 q_4 q_0 q_1 q_2 q_3 q_4 q_5 (q_6) q_3 q_4 q_2 q_0