String matching :

* String matching :

- finding the substring (pattern) in main string (Text)
- patter (P) and Text (T)
- size of P = m
  size of T = n
- x] z : string x is suffix of z
  y [z : string y is prefix of z
- Algorithm of string matching
  1. Naive - string matching
  2. Rabin - karp
  3. KMP

[] Naive - String matching :

- Also called Brute force string matching
- Simple technique
- Complexity : $(n-m+1)(n)$
  $= O(nm)$

- Algorithm :

  T = Text
  P = Pattern

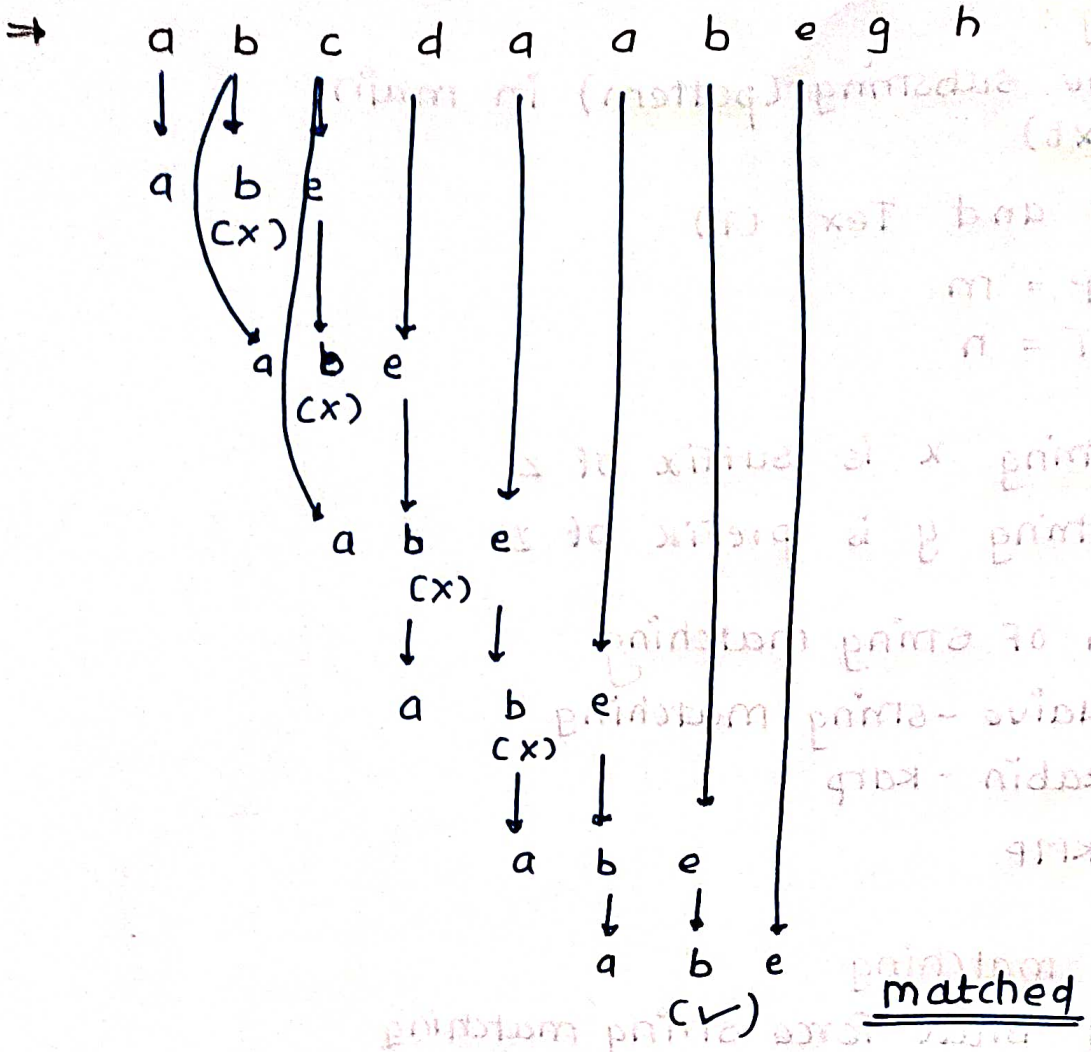
  then

```
Naive_string_matcher (T, P)
1  n = T.length
2  m = P.length
3  for i=0 to n-m
4        if P[1....m] = T [i+1 ..... i+m]
5              print "pattern match at ith shift"
```

Ex 1 : Average case

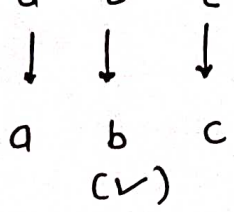
Text : a b c d a a b e g h

pattern : a b e



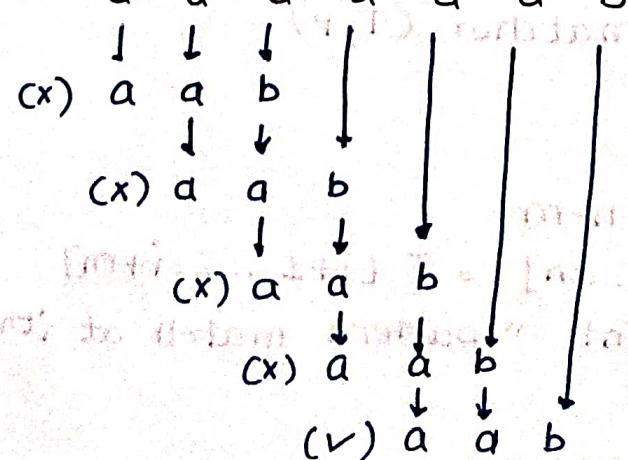
Ex 2 : Best case : pattern = a b c

Text : a b c d a b e a b



Ex 3 : Worst case : pattern = a a b

Text : a a a a a a b



② Rabin karp algorithm :

- String matching algorithm
- Uses hashing technique to find pattern in the text.
- Terms

  Hash function > converts a substring of the text into a hash value.

  Rolling hash function > updating hash value for each substring by considering the previous hash value.

- There are several way to calculate hash value

[i] way 1 :

Steps : 1) maps each character with no. in Text and pattern
(a = 1, b = 2, c = 3 ..... z = 26)

Hash function { 2) Add all digit assign to pattern (eg. abe)
(a = 1, b = 2, e = 5)
1 + 2 + 5 = (8) [Hash value]

3) Now start calculating hash value using same tech. for m-character subsequence of Text.

4) if hash value calculated matches with hash value calculated at step 2
then : pattern match

else
calculate hash value for next m-character sub sequence of Text by shifting 1 to right side.

- Complexity for avg. cases = O(n-m+1)
  complexity due to drawback = O(nm)

Drawback : Suprious hit occurs where hash value of pattern matches with hash value of m-char subsequence of Text but actual pattern is different.

i] Ex: Text : a b c d a b c e

pattern : a b e

→ Text : a b c d a b c e
(1) (2) (3) (4) (1) (2) (3) (5)

pattern : a b e
(1) (2) (5)

∴ Hash value of pattern = 1+2+5 = (8)

→ Text : a b c d a b e e
(1) (2) (3) (4) (1) (2) (3) (5)

× (6) 1 + 2 + 3

× (9) 2 + 3 + 4

✓ (8) 3 + 4 + 1

∴ Here hash value matches

but if we check actual pattern

c d a ≠ a b e

(m-char) — spurious hit

× (7) 4 + 1 + 2

(8) 1 + 2 + 5

∴ Here

a b e = a b e (matched)

[ii] way 2

i) map each char with no. (present in Text & pattern)

(a = 1, b = 2 .... z = 26)

ii) Hash function:

→ $P[1] \times 10^{(m-1)} + P[2] \times 10^{m-2} \ldots\ldots P[m] \times 10^{0}$

ex : a b c = 1 2 3

⇒ $1 * 10^{2} + 2 * 10^{1} + 3 \times 10^{0}$ = (123) [Hash value]

iii] Now in same fashion generate hash value for m-character subsequence of Text

if value match
- pattern match

else
- check for next m-char subsequence by shifting 1 to the left.

ex Text : c c a c c a d b a c

pattern : d b a

→ Hash value for pattern: d b a
(4) (2) (1)

$4 \times 10^2 + 2 \times 10^1 + 1 \times 10^0 = 421$

c	c	a	c	c	a	d	b	a	c
(3)	(3)	(1)	(3)	(3)	(1)	(4)	(2)	(1)	(3)

→ i) cca = $3 \times 10^2 + 3 \times 10^1 + 1 \times 10^0 = 331$
ii) cac = $3 \times 10^2 + 1 \times 10^1 + 3 \times 10^0 = 313$
iii) acc = 133
iv) cca = 331
v) cad = 314
vi) adb = 142
vii) dba = 421 ✓ match dba = dba
viii) bac =

∴ complexity = O(n+m)

iii] way 3 :

→ same as way 2 but instead of hash function use rolling hash function to generate hash values. (i.e use previous hash values)

ex : Text : c c a c c a d b a c

pattern : d b a

→ Hash value for pattern

= d b a
4 2 1

$= 4 \times 10^2 + 2 \times 10^1 + 1 \times 10^0 = 421$

c	c	a	c	c	a	d	b	a	c
(3)	(3)	(1)	(3)	(3)	(1)	(4)	(2)	(1)	(3)

1) ccq = $3 \times 10^2 + 3 \times 10^1 + 1 \times 10^0 = 331$
2) cqc = $(331 - 3 \times 10^2) \times 10 + 3 \times 10^0 = 313$
3) qcc = $(313 - 3 \times 10^2) \times 10 + 3 \times 10^0 = 133$
4) ccq = $(133 - 1 \times 10^2) \times 10 + 1 \times 10^0 = 331$
5) cqd = $(331 - 3 \times 10^2) \times 10 + 4 \times 10^0 = 314$
6) qdb = $(314 - 3 \times 10^2) \times 10 + 2 \times 10^0 = 142$
7) dbq = $(142 - 1 \times 10^2) \times 10 + 1 \times 10^0 = 421$ ✓ matched
8) bac =

Here rolling hash function is used.

iv] As we are taking power, so we may get very large values in some cases, so to avoid such cases we can use mod operation while hash value calculation.

③ KMP algorithm :

→ knuth-Morris-Pratt algorithm for string matching
→ It avoids unnecessary character comparisons by utilizing information from previous matches.
→ We need to calculate prefix table using prefix function
→ prefix function(π) calculates the length of the longest proper prefix that is also proper suffix of given substring

Prefix table example : using Prefix function (ΠΠ)

① string :

a	b	a	b	d
0	0	1	2	0

i) a x = a b x
ii) ab = a a ✓ ①
iii) aba = ab ba x

iv) abab = a d ✓
ab ab ✓ ②
aba bab x

v) ababd = a d x

②

a	b	a	b	a	c	d
0	0	1	2	3	0	1

1) a → x

2) ab → a b x

3) aba → a a ✓ ①
ab ba x

4) abab → a b x
ab ab ✓ ②
aba bab x

5) ababa → a a ✓
ab ba x
aba aba ✓ ③
abab baba x

6) ababac ⇒ a c x
ab ac x
aba bac x
abab abac ✓
ababa babac x

7) ababaca ⇒ a a ✓ ①
ab ca x
aba aca x
abab baca x
ababa abaca x
ababac babacax

Steps for KMP algo :

1] for given pattern calculate prefix table (Preprocessing)

2] use two pointers (initialization)

for Text (i) = 1

for pattern (j) = 0

3] Matching :

compare characters : while iterating through Text (T) and Pattern (P)

- If characters match
  increment both i & j
- If character mismatch
  update j using info from table.
  (it avoids unnecessary char comparison)

Example : a b a b c a b c a b a b a b d (Text)

a b a b d (pattern)

⇒ i) Prefix table :

a	b	a	b	d
0	0	1	2	0

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Text (T):	a	b	a	b	c	a	b	a	b	d	a	b	a	b	c

(i) = 1

Pattern (P):

(j) = 0

	a	b	a	b	d
π → 0	0	0	1	2	0

if T[i] == P[j+1]
then i++
j++
else if (j == 0)
j = π[j]
i++
else
j = π[j]

stop when j = m

⇒ T[1] = P[0+1]
a = a ✓
T[2] = P[1+1]
b = b ✓
T[3] = P[2+1]
a = a ✓
T[4] = P[3+1]
b = b ✓
T[5] = P[4+1]
c ≠ a ✗
∴ set j = π[j]
i.e. (j = 0)

T[5] = P[0+1]
c = a ✗ && j == 0
∴ set j = π[j]
(j = 0)
i = 6

T[6] = P[0+1]
a = a ✓
T[7] = P[1+1]
b = b ✓
T[8] = P[2+1]
a = a ✓
T[9] = P[3+1]
b = b ✓
T[10] = P[4+1]
d = d ✓

as here j = 5 & m = 5
∴ stop pattern matched ✓

∴ Complexity = O(n)

Convex polygon :

- If you connect any two points within a polygon, all the points on that line are also within the polygon.

Problem Statement :

Given set of points (n- with x and y co-ordinates), Find whether the polygon defined is convex?
- order of points is imp
- finding if the given point is within the polygon _?

Solution

1] i] Forming triangles :
- forming triangles from given set of n points
- there can be several triangles formed by connecting combinations of three points.

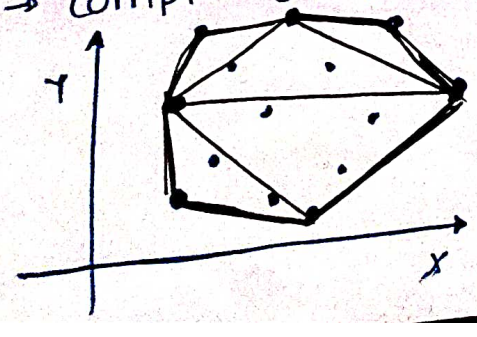
2] Convexity check :
- check triangles are convex or not?

3] Assumption :
- soln assume that checking whether a point lies within a triangle takes 1 unit of time

complexity :
- for forming all possible triangles $= O(n^3)$
- for checking if a point lies within it $= O(n)$

$\therefore$ Total : $O(n^4)$

2] Divide and Conquer (D and C) method :
- Given a set of points we need to find convex Hull [smallest convex polygon that contains all the points in the plane]
- complexity : $O(n \log n)$



Solution approach :

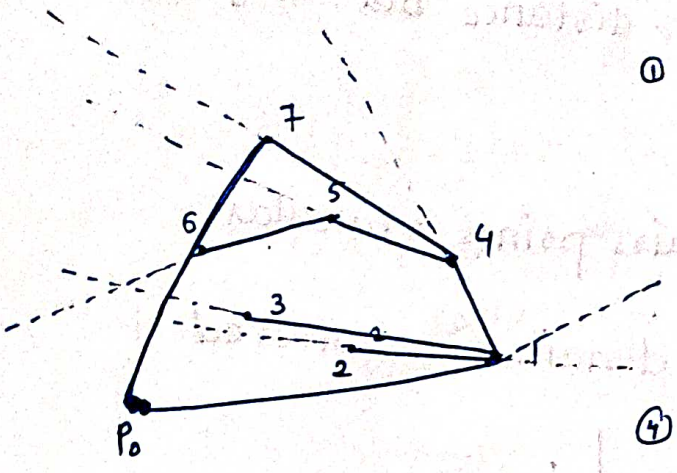
1) Select extreme points (left, right)
2) Join left and right point (so that line joining those points divides the space in lower/ Upper hull.).
3) Upper Hull :
   - Select triangle formed by these two points & third point (so that area will max.
   - Now consider two small hulls above the sides of triangle
   - Repeat till no point will be outside.
4) Repeat same for lower hull.

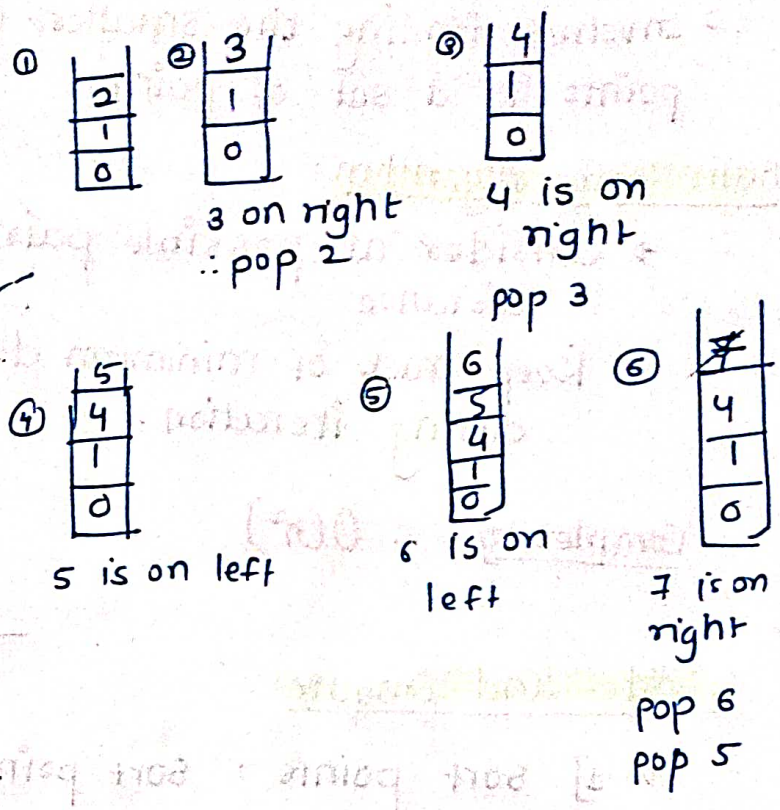
Other methods :

1] Grahm scan :

Algo :

1) Find (leftmost & rightmost point) P_0.
2) for remaining points, sort by polar angle in counter clockwise around P_0.
3) Let S is empty stack
4) push (P_0, S)
5) Push (P_1, S)
   push (P_2, S)
6) for i=3 to n — as 3 points are already done
   {
   IF (p_i is left of top of stack)
   then
   push (p_i, S)
   else
   pop (S)
7) push (p_i, S)
8) return S.



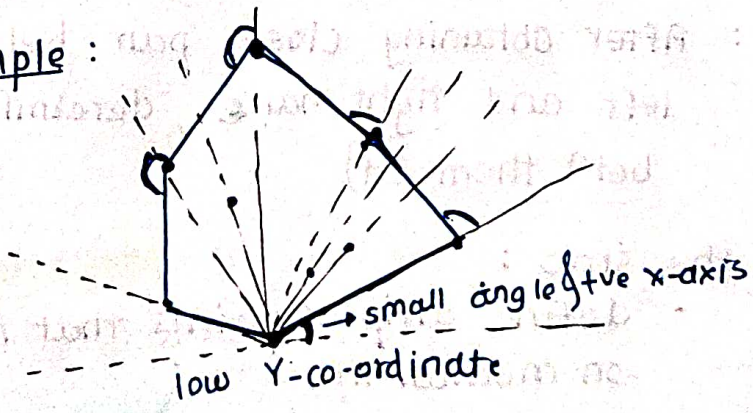


Complexity : $n \log n$

2] Jarvis march :

Algo : Note lowest & highest point
- start from lowest point with low Y-co-ordinate
- At every point search for point which makes lowest polar angle with +ve X-axis

Example :



Complexity $n * h$ — points defining convex polygon

no. of points.

Find closet pair of points :

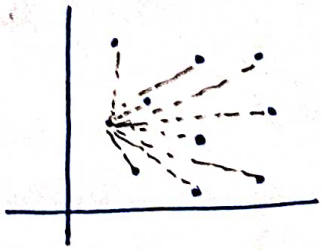
- Involves finding the smallest distance betn any two points in a set of points.

Brute force Algorithm :

→ Consider all possible pairs points : Calculate distance

→ Keep track of minimum distance encountered during iteration.

Complexity : $O(n^2)$



like this

Divide and Conqure :

→ 1] Sort points : Sort points based on their x- co-ordinates

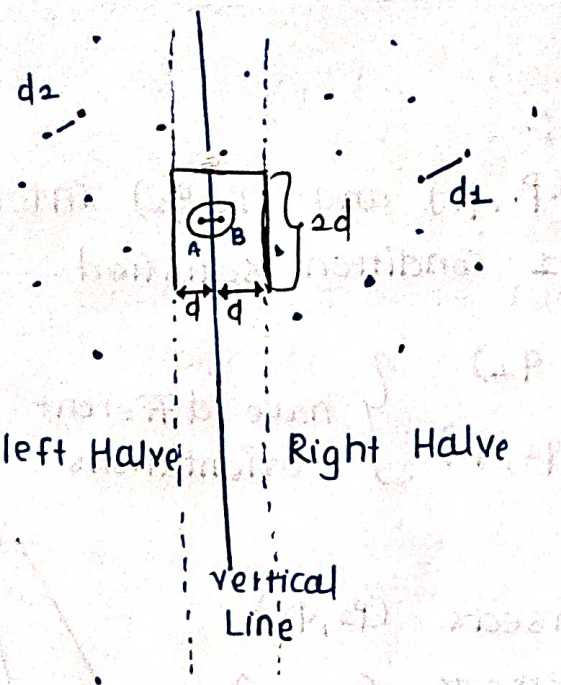
2] Divide : Divide set of points into <u>two halves</u> along the vertical line passing through x-co-ordinate.

3] Conqure : Find Recursively, the closet pair of points in each halve. (By apply same DQ tech)

4] Combine : After obtaining closet pair in left and right halve, determine minimum betn them. (d)

5] Define the strip :

- define strip of points that is centered on median line.
- It is vertical band that includes points with min distance found..
- check points that might form a closer pair (which one point lies in left & other in right halve)
- update min distance accordingly. (if new distance < d)



i) $d = \min(d_1, d_2)$

$\therefore$ AB → closet pair

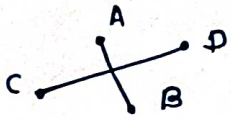
Complexity :

$O(n \log^2 n)$

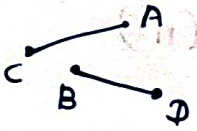
• pair of Intersecting segments :

Terminologies :

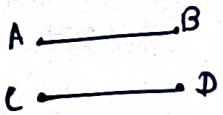
1] Intersecting :



2] Non intersecting :



3] Parallel :

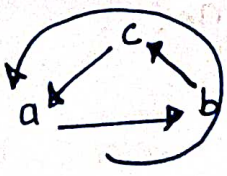


4] Colinear / Overlapping :

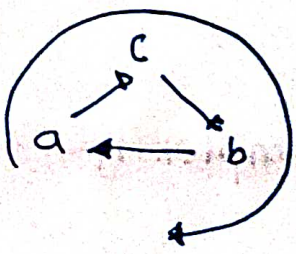
5] Colinear :

orientation

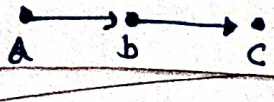
1] Counterclockwise



2] Clockwise



3] Colinear



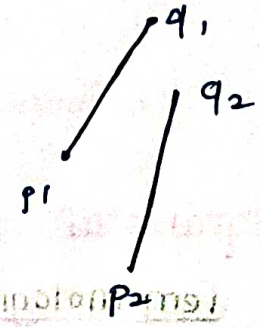
* So,

Intersection of segment :-

Two segments (p_1, q_1) and (p_2, q_2) intersect iff following any 1 condition satisfied.

(A) (p_1, q_1, p_2) & (p_1, q_1, q_2) } have different orientations
(p_2, q_2, p_1) & (p_2, q_2, q_1) }

(B) → All are collinear
→ x-projection intersects (p_2, q_2)
→ Y-projection intersects (p_2, q_2)



Approach

(1) Brute force Approach:

→ for every line segment check if they intersect. (using above conditions (A) & (B))

→ Complexity : $O(n^2)$

(2) Sweepline Algo :

→ Process events as a sweep line moves from L to R.

→ Events :

i) Insert : Add to active set, if left endpoint

ii) Remove / Delete : Remove from set, if Right end

iii) Intersect : check for intersection betn current segment & those are in active set.

→ Active set : maintains segments intersecting with sweep line.

→ Complexity : $O(n \log n)$

new distance < d)

Steps :

① Sort all endpoints of seg. L-R

② - Add seg to active set ⇒ left end
- Delete seg from active set ⇒ Right end
- Check intersection of current seg with seg present in active set