* Pseudocode

```
Karatsuba(x,y):
if x<10 or y<10 return x*y
split x into a,b ; y into c,d
  p1 = Karatsuba (a,c)
  p2 = Karatsuba (b,d)
  p3 = Karatsuba (a+b, c+d)
 return p1* 10^n + (p3-p1-p2)* 10^(n/2) + p2
```

* Input

x = 1234
y = 5678

* output

z = 7006652.

48 Dynamic programming

Aim:
To find the number of ways to obtain a given target sum by throwing a specified number of dice with given number of sides using dynamic programming

* Pseudocode

```
dp[0][0] = 1
for i=1 dice
for s = 1.. target
for f=1.. sides
 if (s-f >=0) dp[i][s] += dp[i-1][s-f]
 print dp[dice][target]
```

* Input

Number of sides = 6
Number of dice = 2
Target sum = 7

* output:

Number of ways to reach sum 7: 6

49 Aim:
To ~~determine~~ find the minimum time required to pro a product through two assembly lines considering station time, transfer time, entry time and exit tim

* Pseudocode:

```
f1 = e1+a1[0];
f2 = e2+a2[0];
for i=1 to n-1
  f1 = min (f1+a1[i], f2 +t2[i-1] +a1[i]);
  f2 = min (f2+a2[i], f1 +t1[i-1]+a2[i]);
answer = min (f1+x1, f2+x2);
```

* Sample Input
n=4
a1 = {4,5,3,2}
a2 = {2,10,1,4}
t1 = {7,4,5}
t2 = {9,2,8}
e1 = 10
e2 = 12
x1 = 18
x2 = 7
* output
Minimum time required = 35

(24) Aim: To minimize the total production time by optimally scheduling tasks across three assembly lines considering station times, transfer times and tasks dependencies

* pseudocode
for l=0 to 2
dp[0][l] = time[l][0];
for i=1 to 2
for l = 0 to 2
dp[i][l] = time[l][i] + min(dp[i-1][0] + T[0][l],
dp[i-1][1] + T[1][l],
dp[i-1][2] + T[2][l])
answer = min (dp[2][0], dp[2][1], dp[2][2]);

* Input:
No. of. Stations = 3
Station times:
Line 1: 5 9 3
Line 2: 6 8 4
Line 3: 7 6 5
Transfer times:
0 2 3
2 0 4
3 4 0
Dependencies = (0 -> 1 -> 2)
* output
minimum production time = 17

(21) Aim: To find the minimum path distance that visits all nodes exactly once and returns to the starting node using a distance matrix

* pseudocode:

```
mincost = INF;
permute(cities);
for each permutation p
cost = sum(dist[p[i]][p[i+1]] + dist[p[last]][p
mincost = min(mincost, cost);
```

* Input:

Distance matrix:

0	10	15	20
10	0	35	25
15	35	0	30
20	25	30	0

* output:

minimum path distance = 80

(22) (TSP)

Aim: To find the shortest possible route that visits all 5 cities exactly once and returns to the starting city using the TSP approach

* pseudocode:

```
mincost = INF;
permute(cities[1..n-1]);
for each permutation p
cost = dist[0][p[0]] + sum(dist[p[i]][p[i+1]] + d
[p[last]][0];
mincost = min(mincost, cost);
```

* Input

Cities: A, B, C, D, E

Distance matrix:

	A	B	C	D	E
A	0	10	15	20	25
B	10	0	35	25	30
C	15	35	0	30	20
D	20	25	30	0	15
E	25	30	20	15	0

* output

Shortest Route: A→B→D→E→C→A

minimum Distance = 85

(33)

Aim,
To find the longest substring of a given string that reads the same forward and backward (palindrome)

* pseudocode:
maxlen = 0;
for i = 0 to n-1
expandAroundCenter(s, i, i);
expandAroundCenter(s, i, i+1);
return substring(start, start + maxlen);
while left >= 0 and right < n and s[left] == s[right]
if (right - left + 1) > maxlen
maxlen = right - left + 1
start = left
left --, right ++

* Input
s = "babad"

* output
"bab" // "aba"

(34) Aim:
To find the length of the longest substring of a given string s that contains no repeating characters

* pseudocode:
maxlen = 0;
start = 0;
map[256] = {-1};
for end = 0 to n-1
if map[s[end]] >= start
start = map[s[end]] + 1
map[s[end]] = end
maxlen = max(maxlen, end - start + 1)
return maxlen;

* Input
s = "abcabcbb"

* output
3

(35) Word Break

Aim: To determine whether a given string can be segmented into a sequence of dictionary words, allowing reuse of dictionary words

* pseudocode
dp[0..n] = {false};
dp[0] = true

for i=1 to n
for j=0 to i-1
if dp[j] and s[j..i-1] in wordDict
dp[i] = true
return dp[n] ;

* Input:
s = "Leetcode"
wordDict = ["Leet", "code"]

* output
true

(56)

Aim:-
To determine whether a given string can be segmented into a sequence of dictionary words and print "yes" if possible

* pseudocode:
dp[0..n] = {false};
dp[0] = true;
for i=1 to n
for j = 0 to i-1
if dp[j] and s[j..i-1] in dict
dp[i] = true
if dp[n] then print "yes"
else print "No"

* Input:
Dictionary = {i, Like, sam, sung, samsung, mobile, ice cream, man go, mango}
Input string: "iLike"

* output
yes

(57) Aim: To format an array of words into lines of exactly maxwidth characters, fully justifying the text by distributing spaces evenly and Left - justifying the Last Line

* pseudocode
i = 0;
while i<n:
j = i; len = 0;
while j<n and Len + words[j].Length + (j-i) <= max width;
Len += words[j].Length; j++;
spaceslots = maxwidth - len;

if j-i==1 or j==n:
Line = join (words[i..j-1], " ");
else:
Spaces = spaceSlots / (j-i-1),
extra = spaceSlots % (j-i-1)
Line = distribute spaces between words
Left slots get extra i=j;
output.push(Line);

* Input
words = ["this", "is", "an", "example", "of", "text", "justification"]
maxwidth = 16

* output
["this is an", "example of text", "justification."]

(28)

Aim: To design a special dictionary that finds a word index based on a given prefix and suffix returning the Largest index if multiple matches exist

* pseudocode
init (words[])
Store words with index
f(pref, suff):
ans = -1
for i=0 to n-1;
if startsWith (words[i], pref) && endswith (words[i], suff):
ans=i
return ans

* Input
words = ["apple"]
pref = "a"
suff = "e"

(29)

Floyd's Algorithm

Aim.
To find the shortest path between all pairs of cities (nodes) in a weighted graph using dynamic programming

* pseudocode

for k=0 to n-1
for i=0 to n-1
for j=0 to n-1
dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);

* problem ①

n=4
edges = [[0,1,3],[1,2,1],[1,3,4],[2,3,1]]
distanceThreshold = 4

* Distance Matrix (Before floyd)

	0	1	2	3
0	0	3	∞	∞
1	3	0	1	4
2	∞	1	0	1
3	∞	4	1	0

* Distance matrix (After floyd)

	0	1	2	3
0	0	3	4	5
1	3	0	1	2
2	4	1	0	1
3	5	2	1	0

* output

3

⑥

Aim: To Implement floyd's Algorithm to compute the shortest paths between all pairs of routers Simulate a Link failure b/w Router B and Router

* pseudocode.

init dist[][].
for k, i, j:
dist[i][j] = min(dist[i][j], dist[i][k] + dist[k
fail (B,D) = INF
repeat floyd

* Input

Routers = {A, B, C, D, E, F}
A-B=1
A-C=5
B-C=2
B-D=1
C-E=3
D-E=1
D-F=6
E-F=
Link failure: B-D

* output

Router A to Router F = 5

(61) Aim:
To implement Floyd's Algorithm to compute the shortest path between all pairs of cities

* Pseudocode
init dist[][]
for k, i, j:
dist[i][j] = min (dist[i][j], dist[i][k] + dist[k][j])
* Input
n = 5
edges = [[0,1,2], [0,4,8], [1,2,3], [1,4,2], [2,3,1], [3,4,[illegible]
distance threshold = 2
* output
0

(62): Aim:
To construct OBST
* pseudocode:
for i = 1 to n:
cost [i][i] = freq[i], root [i][i] = i
for len = 2 to n:
for i = 1 to n - len + 1:
j = i + len - 1
cost[i][j] = INF
for r = i to j:
t = cost [i][r-1] + cost[r+1][j] + sum (freq, i, j)
if (t < cost[i][j]) cost[i][j] = t, root[i][j] = r
* Input:
N = 4
Keys = {A, B, C, D}
Frequencies = {0.1, 0.2, 0.4, 0.3}
* output
optimal cost = 1.7

(63) Aim: To construct an optimal binary search tree using given keys and frequencies
* pseudocode
for i = 1..n: cost[i][i] = freq[i], root [i][i] = i
for l = 2..n:
for i = 1..n - l + 1 :
j = i + l - 1, cost [i][j] = INF
for r = i..j:
t = cost[i][r-1] + cost [r+1][j] + sum(freq, i, j)
if (t < cost[i][j]) cost[i][j] = t, root[i][j] = r
* Input:
N = 4
Key = {10, 12, 16, 21}
frequencies = {4, 2, 6, 3}
* output
optimal cost = 26

(24) Aim: To determine the winner (mouse, cat or Draw) in a graph-based game using optimal play str

* pseudocode.

```
if (mouse == 0) return 1;
if (mouse == cat) return 2;
dp[m][c][t] = Draw;
for (each move)
dp[m][c][t] = bestResult();
```

* input:

graph = [[2,5],[3],[0,4,5],[1,4,5],[2,3],[0,2,3]]

* output

0

(25) Aim:

To find the maximum probability path between two nodes in an undirected weighted graph

* pseudocode:

```
prob[start] = 1;
while (pq not empty) {
  u = extractMax();
  for (each v of u)
prob[v] = max(prob[v], prob[u] * w(u,v));
  }
```

* Input:

n=3

edges = [[0,1],[1,2],[0,2]]

Succ prob = [0.5, 0.5, 0.2]

Start=0

end=2

* output

0.25000

(26) Aim: To calculate the number of unique pat robot can take to reach the bottom-right an m×n grid.

* pseudocode

```
for i=0..m-1:
for j=0..n-1:
if (i==0||j==0) dp[i][j] = 1;
else dp[i][j] = dp[i-1][j] + dp[i][j-1];
```

* Input

m=3, n=7

* output

28