CHAPTER 5

IV

DECREASE-and-CONQUER

The Decrease and conquer is an approach solving a problem by

* changing an instance into smaller instance of the problem
* solve the smaller instance
* convert the solution of smaller instance into a solution for larger instance.

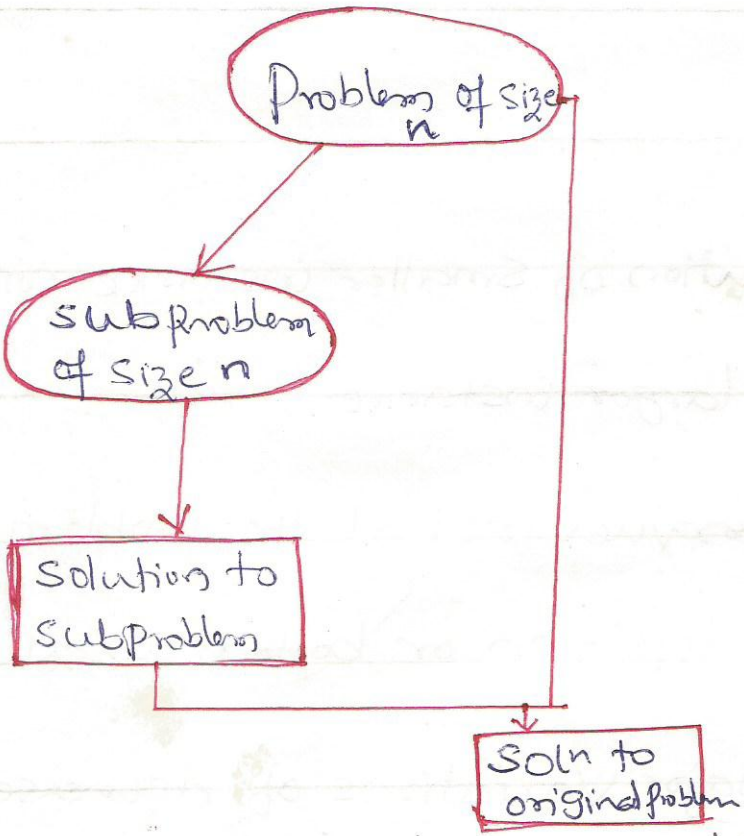
In decrease and conquer method the problem can be solved using topdown or bottom up solution.

The three major variations of decrease and conquer are

1. decrease by constant
2. decrease by constant factor
3. Variable size decrease

I. DECREASE BY CONSTANT

In this method the size of the instance is reduced by same constant on each iteration of the algorithm. Generally this constant is equal to one. The decrease by constant is illustrated in following fig.



Eg: Algorithm to Compute a^n.

In this algorithm the size of the problem, that is reduces by 1 in each recursive call

The algorithm is

```
algorithm power(a,n)
    // a and n are i/ps.
begin
    if (n=1) return a
    return power(n-1,a) * a
End
```

↑ n reduces by 1

Applications of decrease by constant and conquer method is used to solve following problems

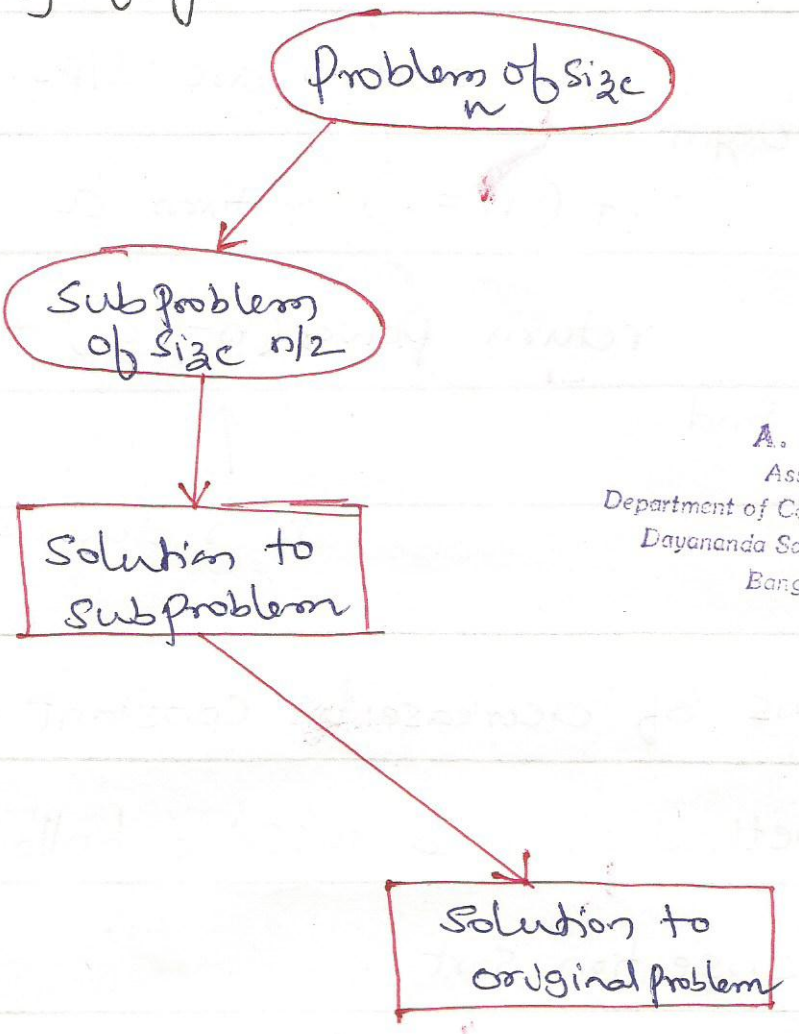
1. Insertion sort
2. Graph searching algorithm
   * Depth first Search
   * Breadth first Search
   * Topological Sorting.

II DECREASE BY A CONSTANT FACTOR

Decrease by a constant factor decreases the instant size by half or by some other fraction.

(4)

The decrease by constant factor is illustrated in following fig.



A. M. PRASAD
Assistant Professor
Department of Computer Science & Engineering
Dayananda Sagar College of Engineering
Bangalore - 560 078

Eg: Algorithm to compute a^n

1. $a^n = a$ if $n = 1$ Eg: $4^1 = 4$

2. $a^n = a^{n/2} * a^{n/2}$ if n is even

   Eg. if $n=8$, $4^8 = 4^4 * 4^4$

3. $a^n = a^{(n-1)/2} * a^{(n-1)/2} * a$ if n is odd

   if $n=9$, $4^9 = 4^4 * 4^4 * 4$

```
Algorithm  power(a, n)
    // a and n are inputs
    begin
        if (n=1) return a
        if (n>1 and n%2 =0) power(a, n/2)
    return power(a,(n-1)/2) * power(a,(n-1)/2) * a
    End.
```

n reduces by n/2

The applications of decrease by constant factor ~~one~~ binary search. The Problem instance reduces by half inorder to obtain solution.

A. M. PRASAD
Assistant Professor
Department of Computer Science & Engineering
Dayananda Sagar College of Engineering
Bangalore - 560 078

III Variable size decrease.

The Variable size decrease is one of the variations of decrease and conquer technique. In Variable size decrease method the size reduction pattren varies from one iteration of an algorithm to another.

Eg: The typical ~~variation~~ of this variation is

finding GCD of two numbers using Euclid's algorithm. The formula for finding GCD is

$$gcd(m, n) = gcd(n, m \bmod n).$$

Two numbers m and n goon varying until GCD value is found.

Insertion Sort:

Generally the insertion sort maintains a zone of sorted elements. If any unsorted element is obtained then it is compared with the elements in sorted zone and then it is inserted at the proper position in sorted zone.

Illustration of insertion sort: consider the elements

9
7
5
4
2

The number of elements are five, insertion sort needs (n-1) cycles to arrange them in order.

I cycle. bigger → V=7

9 7 7
7 ⇒ 9 9
5 5 5
4 4 4
2 2 2

Store second element in variable V, if first element is bigger than V, copy the first element in second position. Then copy V to first position

II cycle

7 bigger → V=5
9
5
4
2

7 bigger → V=5 7 5 ↙V
9 7 7
9 9 9
4 4 4
2 2 2

A. M. PRASAD
Assistant Professor
Department of Compuier Science & Engineering
Dayananda Sagar College of Engineering
Bangalore - 560 078

III cycle

5
7
9
4→V
2

5 bigger V=4
7
9
9
2

5 → bigger V=4 5 4 ↙V
7 7 5 5
9 7 7 7
9 9 9 9
2 2 2 2

IV cycle

4 9>2^V 7>2^V 5>2^V 4>2^V 2 ↙V
5 4 4 4 4 4
7 5 5 5 5 5
9 7 7 7 7 7
V→2 9 9 9 9 9

Algorithm for Insertion Sort given below

```
Algorithm InsertionSort(A[1....n])
// Input array A[1...n] of ordable elements
// output A[1...n] sorted in ascending order
begin
    for i ← 2 to n do
begin   v ← A[i]
        j ← i-1
    while (j ≥ 1 and A[j] > v) do
begin   A[j+1] ← A[j]
        j ← j-1
Endwhile
        A[j+1] ← v
End for
End.
```

Time Efficiency: Let T(n) be the time taken by insertion sort algorithm to arrange numbers in ascending order. The worst case input is all the 'n' numbers in decending order. The time complexity depends upon basic operation.

The basic operation is A[j]>V. and how many times it is executed depends upon loops.

$$T(n) = \sum_{i=2}^{n} \sum_{j=1}^{i-1} 1$$

```
for i<- 2 to n
V= a[i]
j= i-1
while(j>=1 and a[j]>V)
```

A. M. PRASAD
Assistant Professor
Department of Compuier Science & Engineering
Dayananda Sagar College of Engineering
Bangalore - 560 078

$$= \sum_{i=2}^{n} i-1-1+1$$

$$= \sum_{i=2}^{n} i-1$$

$$= 1 + 2 + 3 + \cdots\cdots\cdots + (n-1)$$

$$= \frac{n(n-1)}{2}$$

$$= \frac{n^2-n}{2}$$ note: ignore constant

$$= n^2-n$$ note: Consider higher order term

$= n^2$ Since it is worst case time efficiency use O notation.

So $T(n) = O(n^2)$

Advantages of Insertion sort

1. simple to implement
2. This method is efficient, when n is small
3. This is stable algorithm
4. It is in-place sorting algorithm.

Disadvantages are

1. not efficient compare to quick, mergesort
2. Not suitable when 'n' is large.

GRAPH TRAVERSALS

Graph Traversal means visiting nodes of a graph one after the other in a systematic way. Traversal can start from any arbitrary vertex. The two important graph traversal methods are 1) Deapth first Search (DFS)

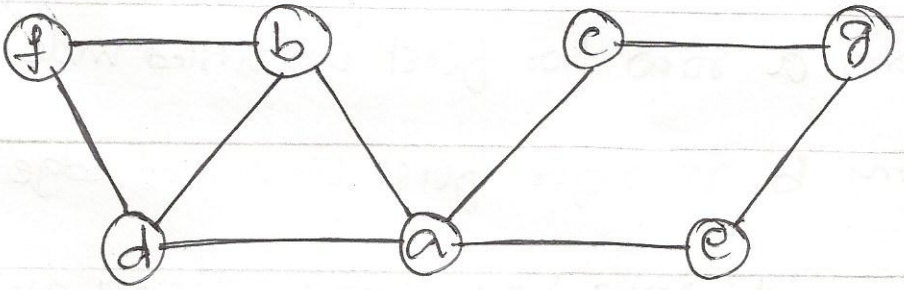
2) Breadth first Search (BFS)

Deapth first Search (DFS)

In DFS a arbitrary Vertex called source is selected and it is visited. Then from source find out the first immediate or adjacent edge and visit that node. Repeat the same process from the new node visited. The search will terminate when all the vertices have been examined or visited.

A. M. PRASAD
Assistant Professor
Department of Compuier Science & Engineering
Dayananda Sagar College of Engine ..g
Bangalore - 560 078

Consider the following graph



for the graph given, write adjacency matrix.

consider the source as a

(12)

	a	b	c	d	e	f	g
a	0	1	1	1	1	0	0
b	0	0	0	1	0	1	0
c	1	0	0	0	0	0	1
d	1	1	0	0	0	1	0
e	1	0	0	0	0	0	1
f	0	1	0	1	0	0	0
g	0	0	1	0	1	0	0

first node to be visited is source a

now scan 'a' row for first unvisited node b, visit

now scan 'b' row for first unvisited edge d. visit

now scan 'd' row for first unvisited edge f, visit

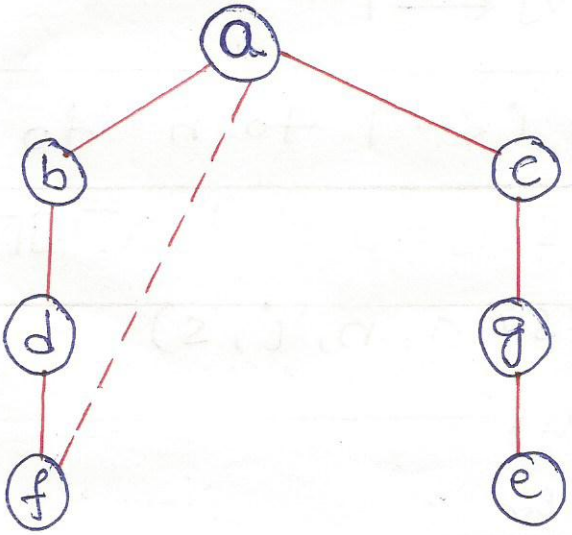
now scan 'f' row for first unvisited edge, two edges are present but they are visited. Now consider the vertices in reverse order one by one and search for unvisited edge. and visit that node

from f, d, b no vertex can be visited but from a, c can be visited, visit c. Now from vertex c, search first unvisited node, g can be visited. from g, vertex e can be visited. Now all vertices are visited, search stops.

The DFS order is

a, b, d, f, c, g, e

In this example, after visiting vertex f, no node can be visited from f, so unvisited node is searched from visited nodes. The from a vertex c is visited. They are called as Back Edges.



The data structure "Stack" is used to conduct DFS. All visited nodes are stored in stack.

When no unvisited edges are present from vertex pop the vertex on top of the stack and search for unvisited edge. This process is repeated till all nodes are visited.

A. M. PRASAD
Assistant Professor
Department of Compuier Science & Engineering
Dayananda Sagar College of Engineering
Bangalore - 560 078

Algorithm:

```
Algorithm Dfs(a, n, v, s)
    // a is adjacency matrix
    // n is number of nodes
    // v is source node
    // s is array indicates vertices visited or not
begin
    s[v] ← 1
    for i ← 1 to n do
    if (s[i]=0 and a[v][i]=1) then
     Dfs(a, n, i, s)
    Endif
    Endfor
End.
```

Analysis.

Every node is visited once, if adjacency matrix is used, it is $n \times n$ matrix is used

$$T(n) = O(|V^2|) \quad V \text{ is Vertices.}$$

if adjacency list is used then

$$T(n) = O(|V| + |E|)$$

Applications of DFS

1. To find graph is connected or not
2. To check graph is acyclic or not
3. To find Spanning Tree
4. To solve Topological Sorting.

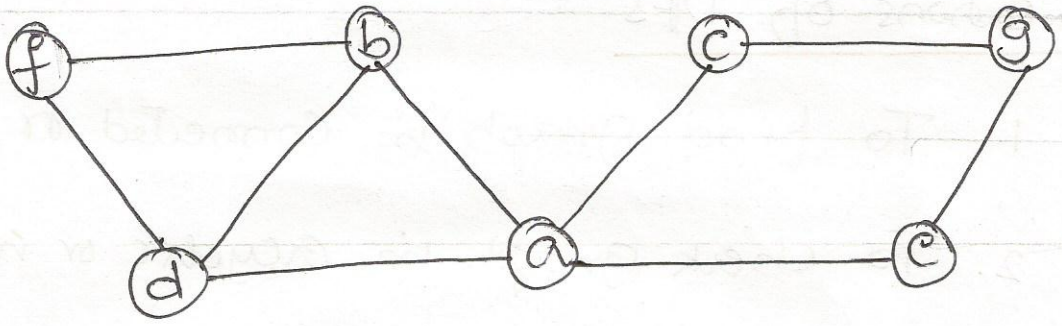
Breadth first Search (BFS)

BFS is method of traversing the Graph by visiting each node of the Graph. In BFS arbitrary node called source node is selected and that is the first node to be visited. From source

node visit all nodes adjacent to it. All nodes visited are stored in queue. The pop the Vertex from queue and visit all the nodes adjacent to it and store them in queue. Then pop the next vertex and repeat the same. till queue becomes empty.

A. M. PRASAD
Assistant Professor
Department of Compuier Science & Engineering
Dayananda Sagar College of Engine...g
Bangalore - 560 078

Consider the following Graph



The adjacency matrix is

	a	b	c	d	e	f	g
a	0	1	1	1	1	0	0
b	0	0	0	0	0	1	0
c	1	0	0	0	0	0	1
d	0	1	0	0	0	1	0
e	1	0	0	0	0	0	1
f	0	1	0	1	0	0	0
g	0	0	1	0	1	0	0

If source is a, visit a first, then from a which all the nodes can be visited, visited all of them ie

a, b, c, d, e

now pop the element from queue, b, visit all the nodes from b and add them to queue

a, b, c, d, e, f

Pop the next element c, visit all unvisited nodes

a, b, c, d, e, f, g

This process is repeated till queue becomes empty.

A. M. PRASAD
Assistant Professor
Department of Compuier Science & Engineering
Dayananda Sagar College of Engineering
Bangalore - 560 078

Algorithm: BFS(a, n, s, source)

// a is adjacency matrix, n is no of nodes
// source is starting node, s indicates node visited/not

```
begin   f ← r ← 01
        q[r] ← source
        s[source] ← 1,  Print source
        while (f <= r) do
begin   u ← q[f]
        f ← f+1
```

(18)

```
for i ← 1 to n do
begin if (s[i]=0 and a[u][i]=1) then
    begin
        r ← r+1
        q[r] ← i
        s[i] ← 1
        Print i
    Endif
Endfor
Endwhile
End.
```

Time Efficiency: The basic operation is if (s[i]=0 and a[u][i]=1). The basic operation execution depends upon loops for and while. Since a graph can have 'n' nodes, the while loop runs 'n' times. For loop runs 1 to n each time.

Let T(n) be the time taken to visit 'n' vertices using BFS method

$$T(n) = \sum_{j=1}^{n} \sum_{i=1}^{n} 1$$

A. M. PRASAD
Assistant Professor
Department of Compuier Science & Engineering
Dayananda Sagar College of Engineering
Bangalore - 560 078

$$T(n) = \sum_{j=1}^{n} n - 1 + 1$$

$$T(n) = \sum_{j=1}^{n} n$$

$$T(n) = n(n-1+1)$$

$$T(n) = n^2$$

Since this is time taken in average or worst case θ or O notations are used. So

$$T(n) = \theta(n^2) \text{ or } \theta(|V^2|)$$

The differences between BFS and DFS are

(20)

1. Data structures is Stack	Data structure is Queue
2. Tree edges and back edges are present	Tree edges and cross edges are present
3. Exploration of node is postponed as soon as a unvisited node is reached.	A node is fully explored before exploration of new node
4. Two vertex ordering is used	one Vertex ordering is used.

TOPOLOGICAL SORTING

The method of arranging vertices in some specific manner is called topological sort. The method is directed acyclic graph, each time select the vertex without indegrees, that is the node visited. The remove the node from graph, check the other nodes with indegree zero, select that node and remove the node from the graph and repeat the same process.

A graph which is cyclic doesnot have topological sequence. Topological sorting can be done using two methods

A. M. PRASAD
Assistant Professor
Department of Compuier Science & Engineering
Dayananda Sagar College of Engineering
Bangalore - 560 078

1. DFS method
2. Source removal method.

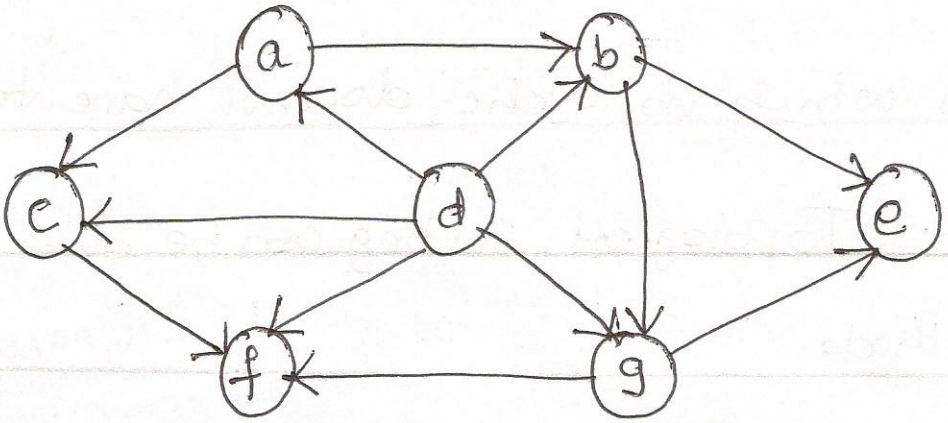
DFS method

The following steps are used to find topological order

1. Select any arbitrary Vertex
2. When Vertex visited first time it is pushed on to stack
3. When Vertex becomes a dead end, it is removed
4. repeat above two steps till all nodes are over.
5. Reverse the order of deleted nodes.

Consider the graph shown

(22)



Let Source be a

Stack	nodes to be visited	Node deleted
a	b	–
a, b	e	–
a, b, e	from e no nodes so delete e	e
a, b	g	–
a, b, g	f	–
a, b, g, f	from f no nodes so delete f	f
a, b, g	from g no nodes so delete g	g
a, b	from b no nodes so delete b	b
a	c	–
a, c	from c no nodes so delete c	c

a — from a no nodes so delete a — a

Now Stack is empty, any node left in graph push it into stack. The node left is d

d — no node can be visited, so pop — d

A. M. PRASAD
Assistant Professor
Department of Compuier Science & Engineering
Dayananda Sagar College of Engineering
Bangalore - 560 078

The nodes poped sequence is

e, f, g, b, c, a, d

The topological order is reverse of popped sequence

d, a, c, b, g, f, e

Algorithm:

```
algorithm topology(s,n, a)
// s is array indicates node visited/not
// a is adjacency matrix
// n is number of vertices.
Step 1: for i ← 01 to n do
        if (s[u]=0) call DFS(u, n, a)
        endfor
```

Step 2: for i ← n downto 1

Print result[i]

Step 3: End

Algorithm DFS(u, n, a)

begin

S[u] ← 1

for v ← 1 to n do

if (a[u][v]=1 and s[v]=0) then

DFS(v, n, a)

Endif

Endfor

result[j++] = u

End

Time Efficiency: Let T(n) be the time taken by algorithm to sort vertices. The basic or key operation is

if (a[u][v]=1 and S[v]=0) then
DFS(v, n, a)

The number of times it is executed on for loop

```
for V ← 1 to n do
if (a[u][v] = 1 and s[v] = 0) then
DFS (v, n, a)
```

A. M. PRASAD
Assistant Professor
Department of Compuier Science & Engineering
Dayananda Sagar College of Engineering
Bangalore - 560 078

$$T(n) = \sum_{v=1}^{n} \sum_{v=1}^{n} 1$$

$$T(n) = \sum_{v=1}^{n} n-1+1$$

$$T(n) = \sum_{v=1}^{n} n$$

$$T(n) = n(n-1+1)$$

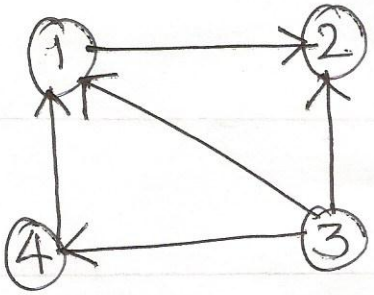
$$T(n) = n^2$$

So time efficiency is $T(n) = \Theta(n^2)$

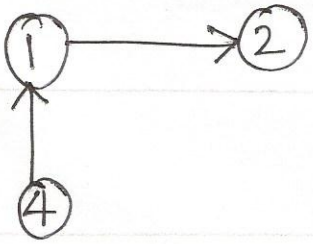
TOPOLOGICAL SORTING USING SOURCE REMOVAL

In this method, a vertex with no incoming edges is selected and deleted along without going edges. Then select the vertex with no indegrees and delete that edge along with outgoing edges. Repeat this method till all nodes in graph are deleted. The nodes deleted order is topological sequence

Eg: Consider the following graph



step 1: delete 3, because it has no indegrees

step 2:  delete node 4, no indegrees

step 3: 

delete node ①, no indegrees

step 4: ②

delete ②, no indegrees

The deleted sequence is 3, 4, 1, 2 and is topological order.

The logic is for given graph write adjacency matrix

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

Now, take sum of each column, the column with sum equal to zero is the first node to be

visited. Then remove that column and row and take the sum of elements in column. The column with 0 is the next node to be visited. This is repeated all nodes are over.

```
Algorithm Topology (a, n)
// a is adjacency matrix, n is number of nodes
begin
    for i ← 1 to n do
    for j ← 1 to n do
    if (a[i][j] = 1) sc[j] = sc[j] + 1
    End
    End
    for i ← 1 to n do
begin
    if (sc[i] = 0) then
begin
        Print i
        sc[i] = -1
        for j ← 1 to n do
begin
        if (a[i][j] = 1) then
        sc[j] = sc[j] - 1
        Endif
        i = 0
        End for
    Endif
    Endfor
End
```

A. M. PRASAD
Assistant Professor
Department of Compuier Science & Engineering
Dayananda Sagar College of Engineering
Bangalore - 560 078

SPACE AND TIME TRADE OFFS

In space and time trade off the idea is to preprocess the problem's input in whole or in a part and store the additional information obtained to accelerate solving the problem later. This is called input enhancement technique

The algorithms under this are

<1> Sorting by counting
<2> Horspool's algorithm
<3> Hashing.

SORTING BY COUNTING:

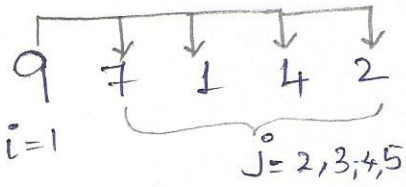
Sorting by counting is also called as comparison counting. It is one of the sorting approaches which uses the input enhancement technique to sort the data. It is used to sort the elements in ascending order. In this method, for each element of the list to be sorted total number of elements smaller than this element are recorded in a table. These recorded numbers would indicate the positions of the elements in the sorted list.

In this algorithm the i^{th} element is compared with the remaining 'j' elements. If incase the i^{th} element is greater than

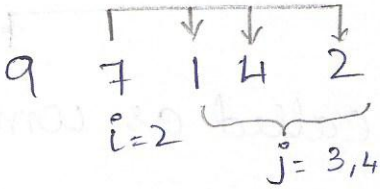
j^{th} element then the count[i] is incremented. Else if the j^{th} element happens to be greater than the i^{th} element the count[j] is incremented. Here i varies from 1 to (n-1) and j from i+1 to n. At the end of all the n-1 passes the count table elements indicates the position of each element of the array in the sorted list.

ILLUSTRATION OF ALGORITHM

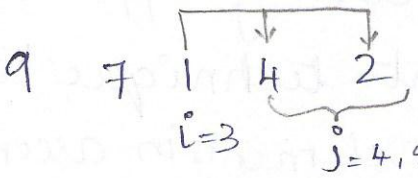
Consider the elements 9, 7, 1, 4, 2



count[1] = 1+1+1+1+1 = 5



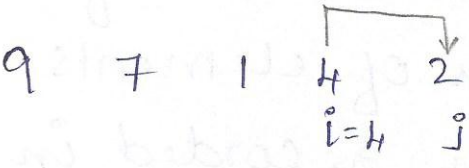
count[2] = 1+1+1+1 = 4



count [3] = 1

count [4] = 1+1 = 2

count [5] = 1+1 = 2



count[4] = 2+1 = 3

9 7 1 4 2
i

count[5] = 2+0 = 2

∴ Count table elements are

1	2	3	4	5
5	4	1	3	2

The sorted elements are

	1	2	3	4	5
b[]	1	2	4	7	9

In the above example, during the 1st pass, i=1, j = 2, 3, 4, 5. a[1] = 9. This element is compared with a[2 ... 5]. If a[1] > a[2 ... 5] then, count[1] is incremented. If a[1] < a[j] where j varies from 2 to 5 then, count[j] is incremented. Thus, we get count[1] = 5 which implies that the element 9 resides in the 5th position in the sorted array.

ALGORITHM FOR SORTING BY COUNTING:

```
Algorithm sorting_by_counting (a, n)
BEGIN
    for i ← 1 to n do
        count[i] = 1

    for i ← 1 to n-1 do
    BEGIN
        for j ← i+1 to n do
        BEGIN
            if (a[i] > a[j])
                count[i] ← count[i] + 1
            else
                count[j] ← count[j] + 1
        END FOR
    END FOR
    for i ← 1 to n do
    b[count[i]] = a[i]
END
```

TIME EFFICIENCY:

Let T(n) be the time required to sort an array using sorting by counting. Let the number of elements be n. Then, the time complexity is given depends on the two for loop $i \leftarrow 1$ to $n-1$ and $j \leftarrow i+1$ to n. The basic operation is if (a[i] > a[j]). The number of times it gets executed is given by.

$$T(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 1$$

$$= \sum_{i=1}^{n-1} [n-(i+1)+1]$$

$$= \sum_{i=1}^{n-1} n - i - 1 + 1$$

$$= \sum_{i=1}^{n-1} (n-i)$$

$$= (n-1) + (n-2) + \cdots + 3+2+1$$

$$= \frac{n(n-1)}{2}$$

$$= n^2 - n$$

NOTE: Constants are not considered in time efficiency analysis

$$= n^2$$

NOTE: Higher order term is considered

Since it is average case analysis

$$T(n) = \theta(n^2)$$

HORSPOOL'S ALGORITHM

Horspool's algorithm is one of the string matching algorithm that uses the technique called input enhancement to search for a pattern in a given text.

In this algorithm initially we create a Shift table using the formula $m-1-i$ where m is the length of the pattern string and i varies from 0 to $m-2$. This indicates the shift to be performed during searching process. The following 2 possibilities may occur during pattern searching.

(1) If the required character is not present in the pattern, then the pattern is shifted by its length

(2) If the character occurs in the text then the pattern is shifted based on the entry present in the shift table.

We first compare the last character of the pattern with the corresponding i^{th} character of the text. If the 2 characters are equal then the other characters of the pattern string are matched. Else the shift is performed as mention above.

The above process is performed til either the pattern is found or the text is exhausted.

ILLUSTRATION OF HORSPOOL'S ALGORITHM:

Consider the ~~text~~ string

TEXT: INDIA _ IS _ A _ DEMOCRATIC

PATTERN: DEMO

SHIFT TABLE

D ---> 3

E --> 2

M --> 1

OTHER --> 4

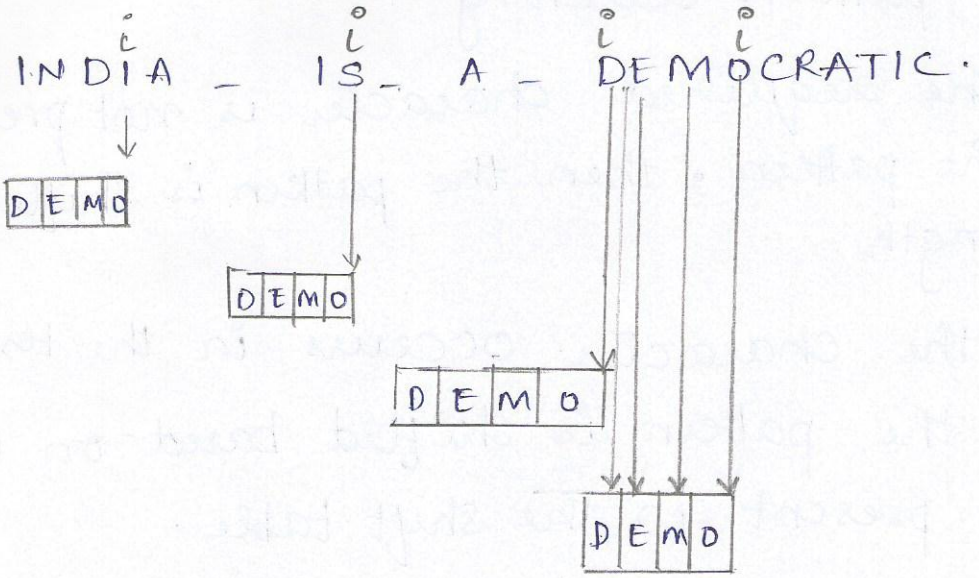


TABLE [I] --> 4
TABLE [N] --> 4
[D] --> ~~4~~ 3
[I] --> 4
[A] --> 4
[-] --> 4
[I] --> 4
[S] --> 4
[A] --> 4
[-] --> 4
[D] --> ~~4~~ 3
[E] --> ~~4~~ 2
[M] --> ~~4~~ 1
[O] --> 4
[C] --> 4
[R] --> 4
[A] --> 4
[T] --> 4
[I] --> 4
[C] --> 4

```
Algorithm  Horspool (T, P, m, n)

begin
    for i ← 0 to n-1 do
     table [T[i]] ← m
     for i ← 0 to m-2 do
     table[P[i]] ← m-1-i
       i = m-1
     while (i <= n-1) do
     begin
         j ← 0
       while (j < m and T[i-j] = P[m-1-j])
           j ← j+1
       if (j = m)
           return i-m+1
       else
           i ← i + table[T[i]]
    end
    return -1
end.
```

HASHING

Hashing is a searching technique using the principles of space and time trade offs.

Hashing is based on the idea of distributing keys among a one-dimensional array called the Hash table. The distribution is done by computing for each of the keys the value of some predefined function h called the hash function. This function assigns an integer between 0 and m-1 called the hash address to a key.

Eg: Let an array consists of elements 0, 2, 5, 9, 7
Let the choosen hash function be

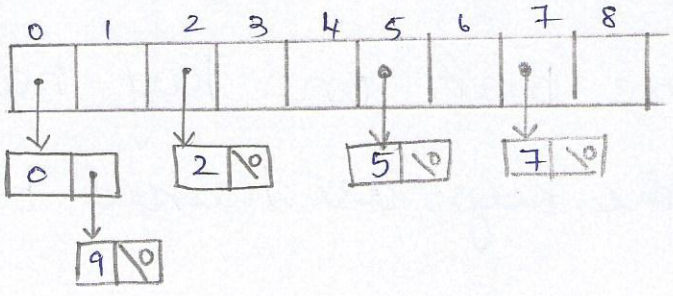
$$h(k) = k \bmod m$$

where k are the keys and ~~m~~ let $m = 9$.

Then,

$$h(0) = 0 \bmod 9 = 0$$
$$h(2) = 2 \bmod 9 = 2$$
$$h(5) = 5 \bmod 9 = 5$$
$$h(9) = 9 \bmod 9 = 0$$
$$h(7) = 7 \bmod 9 = 7$$

We construct the hash table as below



In order to search an element, the hash function is applied on the element and the hash address thus obtained is sequentially searched to find if the element is present.

In order to choose a good hash function the following conditions are to be satisfied.

(a) A hash function must be able to uniformly distribute all the keys in a hash table

(b) A hash function has to be easy to compute

Hashing can be performed in 2 ways:

(1) Open hashing

(2) Closed hashing.

OPEN HASHING:

It is a hashing technique in which keys are stored in linked lists attached to an array cell of hash table using an appropriate hash function. The hash value obtained is used to find the index of list to which the item is to be added. If more than one key has same hash value, then the keys are hashed to the same location.

When an item has to be searched

it is necessary to find the hash value of the item using the hash function. This hash value corresponds to an index which indicates the address of the key's location. If the address list is NULL then the item is not present. Else the list is searched sequentially.

ILLUSTRATION OF OPEN HASHING

Consider the following list of words

A FOOL AND HIS MONEY ARE SOON PARTED

In this example let us assign each character their position in the alphabetical order and compute the sum. Further the sum is divided by 13 and the remainder is taken as hash address. Hence, we get,

A = 1

FOOL = 6 + 15 + 15 + 12 = 48

AND = 1 + 14 + 4 = 19

HIS = 8 + 9 + 19 = 36

MONEY = 13 + 15 + 14 + 5 + 25 = 72

ARE = 1 + 18 + 5 = 24

SOON = 19 + 15 + 15 + 14 = 63

PARTED = 16 + 1 + 18 + 20 + 5 + 4 = 64

Let h(key) = key mod 13

$\therefore$ h(A) = 1 mod 13 = 1

h(FOOL) = 48 mod 13 = 9

h(AND) = 19 mod 13 = 6

$h(\text{HIS}) = 36 \bmod 13 = 10$

$h(\text{SOON})$

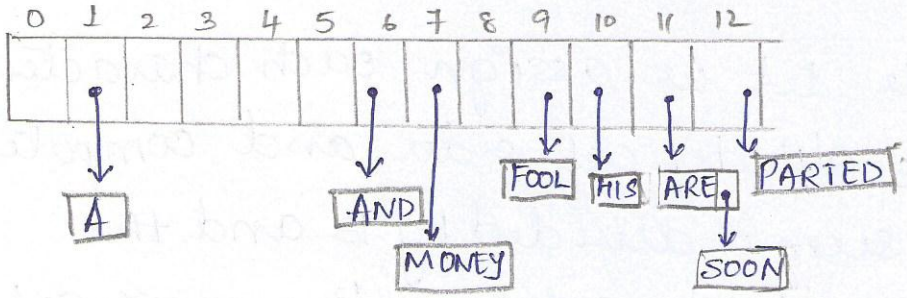
$h(\text{MONEY}) = 72 \bmod 13 = 7$

$h(\text{ARE}) = 24 \bmod 13 = 11$

$h(\text{SOON}) = 63 \bmod 13 = 11$

$h(\text{PARTED}) = 64 \bmod 13 = 12$

The hash table thus constructed is



In the above example the words ARE and SOON have the same hash address. This condition is called collision. Collision can be overcome by using singly linked list attached to the array cell. Collision must be avoided in order to obtain efficient search table. Therefore, a suitable hash function has to be obtained.

TIME EFFICIENCY

The time efficiency of hashing depends on the linked list length which in turn depends on the size of table and hash function.

If the hash function distributes n keys among m cells of the hash table almost evenly then, each linked list will be n/m keys long. The ratio $\alpha = n/m$ is called the load factor of the hash table. Then, if S is the number of successful search and U is the number of unsuccesful search we have,

$$S \approx 1 + \frac{\alpha}{2}$$

$$U = \alpha .$$

In the best case possibility if the hash function is suitably choosen then,

$$T(n) = \Omega(1)$$

CLOSED HASHING (OPEN ADDRESSING)

In closed hashing all the keys are stored in a hash table without the use of linked lists. In order to employ collision resolution the simplest method is linear probing.

In this method using the hash function a hash address to a key is obtained. If the cell is vacant then the key is placed in that cell. Else it checks for the vaccancy of the cell's immediate successor. If that is vacant the key is filled in. Else the vaccancy is checked. Note that

if we encounter the end of hash table then the search is proceeded from the begining of the table, that is, it is treated as a circular array.

To search for a given key k, we start by computing h(k) where h is the hash function used in the table's construction. If the cell h(k) is empty, the search is unsuccessful. If the cell is not empty, we must compare k with cell's occupant: if they are equal then key is found else we compare k with the adjacent cells and continue in this manner til we encounter a the key or an empty cell.

ILLUSTRATION OF CLOSED HASHING

Let h(key) = key mod 13

KEYS	A	FOOL	AND	HIS	MONEY	ARE	SOON	PARTED
HASH ADDRESS	1	9	6	10	7	11	11	12

0	1	2	3	4	5	6	7	8	9	10	11	12
	A											
	A								FOOL			
	A					AND			FOOL			
	A					AND			FOOL	HIS		
	A					AND	MONEY		FOOL	HIS		
	A					AND	MONEY		FOOL	HIS	ARE	
	A					AND	MONEY		FOOL	HIS	ARE	SOON
PARTED	A					AND	MONEY		FOOL	HIS	ARE	SOON

In the above example a collision occurs between "ARE" and "SOON". Since "ARE" is placed in cell 11 and cell 12 is vacant SOON occupies cell 12. Further, "PARTED" occupies cell 0 as it is the next cell vaccant after 12.

COLLISION

DEFINITION: If the size of the hash table is smaller than the total number of keys generated then 2 or more keys will have same hash address. This phenomenon is called collision.

This can be overcome using linked list and is called as collision resolution.