Stack

Stack using linked list

```
#include<iostream>
using namespace std;
struct ll{
    int data;
    struct ll* next;
    };
typedef struct ll node;
  node* top = NULL;
create()
{
  top = new node;
  cout<<"enter the element";
  cin >> top->data;
  top->next = NULL;
}
push()
{
  if (top == NULL)
    create();
  else
  {
    node *p;
    p = new node;
    cout<<"enter the element:";
    cin >> p->data;
    p->next = top;
      top = p;
  }
}
```

```
pop()
{
  if (top==NULL)
   cout<<"Stack is empty";
  else
  {
    node* q;
    q = top;
  cout<<"the deleted element is"<<q->data;
    top = q->next;
    delete(q);
  }
}
display()
{
  if (top == NULL)
   cout<<"Stack is empty";
  else
  {
    node* p = top;
   while (p != NULL)
   {
     cout<<p->data;
      p = p->next;
   }
```

Reverse a string using stack.

```
#include<iostream>
#include<stack>
using namespace std;

void reversesentence(string s)
{
    stack<string> st;
   for(int i=0; i<s.length(); i++)
    {
        string word = " ";
      while (s[i] != ' ' && i<s.length())
       {
            word += s[i]
             i++;
       }
        st.push(word);
    }
    while(!st.empty())
     {
        cout << st.top() << " ";
         st.pop()
     } cout << endl;
}
int main()
{
    string s;
    cout<<"enter the sentence";
    getline(cin, s);
     reversesentence(s);
}
```

^	R→L
*, /	L→R
+, -	L→R

PAGE:
DATE: / /

Infix expression (Human)

<operand> <operation> <operand>

Prefix expression (Comp)

<operation> <operand> <operand>

e.g ((4 * 2) + 3) → + * 4 2 3

(* (4 2) + 3)

+ (* (4 2) 3)

e.g (5 - (6 / 3)) → - 5 / 6 3

(5 - / (6 3))

- (5 / (6 3))

Postfix expression (Comp)

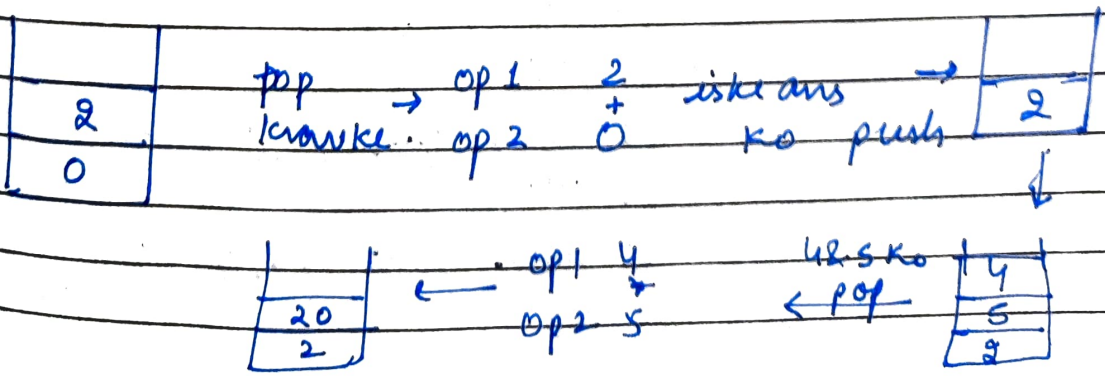
<operand> <operand> <operation>

((4 * 2) + 3) → 4 2 * 3 +

(5 - (6 / 3)) → 5 6 3 / -

✱ Prefix Expression Evaluation

- + 7 * 4 5 + 20



DATE: / /

Prefix evaluation

Code

```
int prefixeva(string s)
{
stack<int> st;
for(int i=s.length()-1; i>=0; i--)
{
if(s[i]>='0' && s[i]<='9')
{
st.push(s[i]-'0');
}
else
{
int op1=st.top();
st.pop();
int op2=st.top();
st.pop();
switch(s[i])
{
case '+':
st.push(op1+op2);
break;
case '-':
st.push(op1-op2);
break;
case '*':
st.push(op1*op2);
break;
case '/':
st.push(op1/op2); break;
case '^':
st.push(pow(op1,op2));
break;
}
}
return st.top();
}
```

for operand →

for operator →

```
int main()
{
cout<<prefixeva
("-+7*45+
<<endl;
return 0
}
```

// 25

6
4

op1 = 6
op2 = 4

PAGE:
DATE: / /

Postfix evaluation code:

Same as prefix

change condition for loop

(int i=0; i < s.length(); i++)

and

op2 = st.top()
st.pop()
op1 = st.top()
st.pop()

Rest is same.

① -+7*45+20

←

0	4	7	27	
+	*	+	-	= 25 Ans.
2	5	20	2	

② 46+2/5*7+

→

4	16	5	25	
+	/	*	+	= 32 Ans
6	2	5	7	

Infix to Postfix

(a-b/c) * (a/k-l)

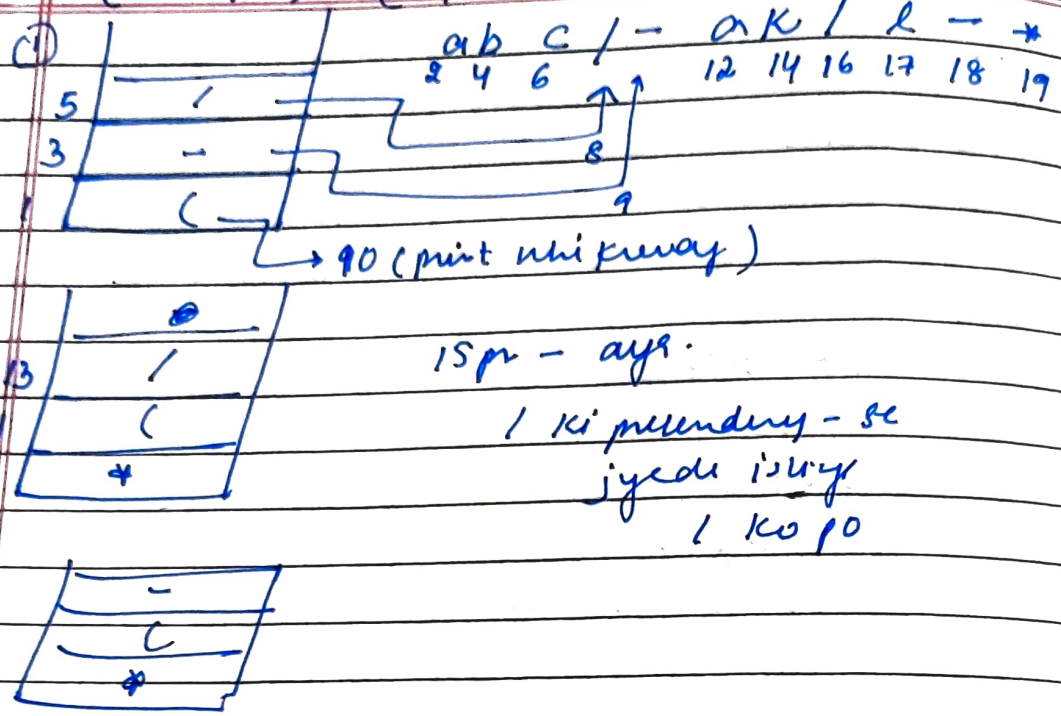
→

if operand → print
if '(' → push
if ')' → pop and print until '('
if operator → pop and print until an operator with less precedence is found.

→ closing bracket agi ab pop until (

7

(a-b/c) * (a/k-l)



isp - ayga.

/ ki pending - se jyada isliye / ko pop

Infix to postfix code

```
int pri(char c)
{
    if (c == '^')
    return 3;
    if (c == '*' || c == '/')
    return 2;
    if (c == '+' || c == '-')
    return 1;
    else
    return -1;
}
string infix_to_postfix (string s)
{
    stack <char> st;
    string res;
```

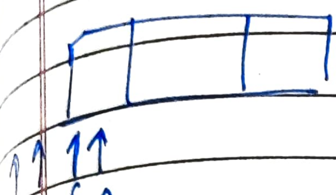
```
for (int i=0; i<s.length(); i++)
{
  if (s[i] >= 'a' && s[i] <= 'z' || s[i] >= 'A' && s[i] <= 'Z')
  {
    res += s[i];
  }
else if (s[i] == '(')
  {
    st.push(s[i]);
else if (s[i] == ')')
{
  while (!st.empty() && st.top() != '(')
    {
      res += st.top();
      st.pop();
    }
  if (!st.empty())
      st.pop();
else
{
  while (!st.empty() && prec(st.top()) > prec(s[i]))
    {
      res += st.top();
      st.pop();
    } st.push(s[i]);
} ans
} while (st.empty());
    {
      res += st.top();
      st.pop();
    }
  return res;
```

→ First In First Out.

FIFO

QUEUE
- enqueue() → front, Back. → F = -1, B = -1, F vhi rahega, B++
- dequeue() → F++
- peek() → F jisko point karsha
- empty() → F > B then empty.

using array



F=-1 B=-1 F B

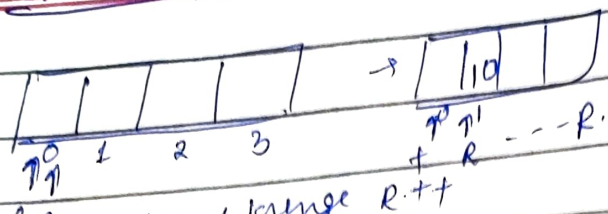
a[b]

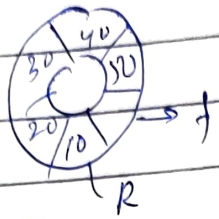
```
int f=-1;
int b=-1;
int n=5;
int a[n];
enqueue()
{
  b++
  if(b>=n)
  {
    cout<<"overflow"<<endl;
  }
  else.
  { if(f==-1)
      f=0;
    cin>>a[b];
  }
}
```

```
dequeue()
{
  if(f==-1 || f>b)
  {
    cout<<"underflow";
  }
  else
  {
    cout<<a[f];
      f++;
  }
}
```

```
print ()
{
  for(int i=f; i<=b; i++)
  {
    cout<<a[i]<<endl;
  }.
```

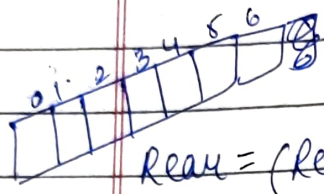
Circular Queue





Java → point Rear with empty numeric

f = R [this is not possible because such condition empty queue but queue is actually full.



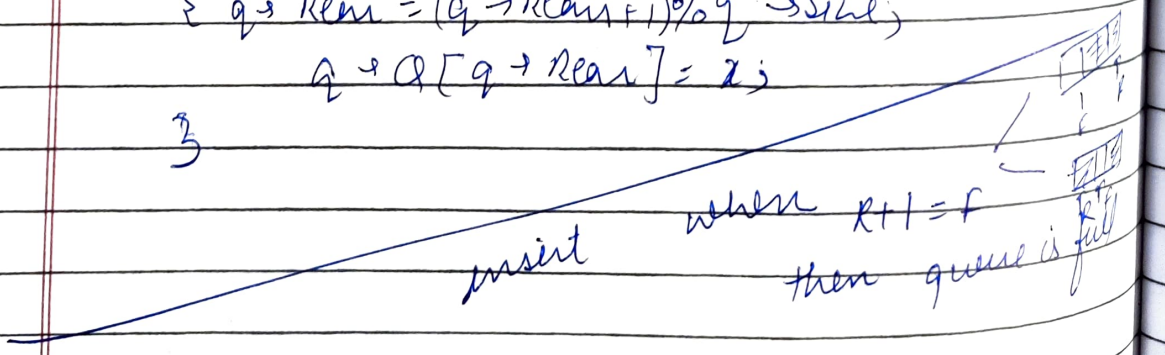
Rear = (Rear+1)%Size

0	(0+1) % 7	1
1	(1+1) % 7	2
2	(2+1) % 7	3
3	(3+1) % 7	4
4	(4+1) % 7	5
5	(5+1) % 7	6
6	(6+1) % 7	0
0		

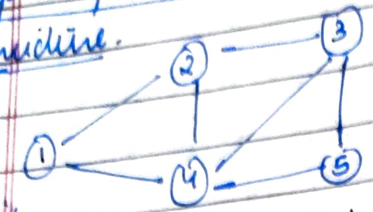
Same for front

```
void enqueue(struct Queue *q, int x)
  if ((q->Rear+1) % q->Size == q->front)
     printf("Queue is full");
  else
  { q->Rear = (q->Rear+1)%q->Size;
     q->Q[q->Rear] = x;
  }
```

insert when R+1 = f then queue is full



★ Graph Representation in Data structure. → Adjacency Matrix. / Adjacency list



when graph is dense use this → Adjacency Matrix:

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

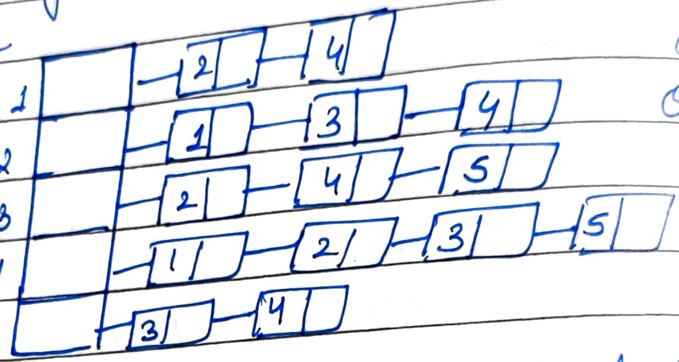
5×5

space complexity $O(n^2)$

$a[i][j] = 1$, if i, j are adjacent
$= 0$, otherwise

[for each vertex one link list] adjacency list

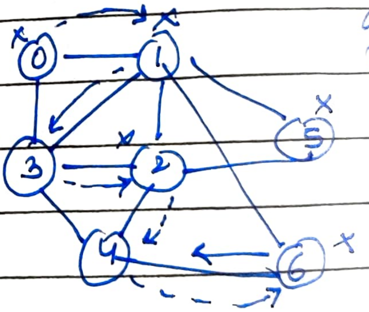
when sparse graph use →



space complexity $O(n+2e)$ → edge is written two times

★ Graph Traversals → BFS Breadth first search (level order traversal) / DFS Depth first search → any node as root

let us take 0 as root node.



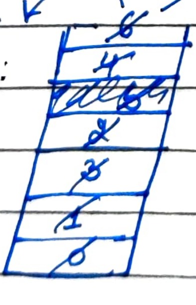
Root choose kre uske saath ke sare adjacent vertices likhe ← For BFS.

Queue :- 0 | 1 | 3 | 2 | 5 | 6 | 4

Result :- 0 1 3 2 5 6 4

For DFS. ← isme sirf ek choose kre until dead end then back track

Stack :



① like we cannot go any further so we will pop up the top

Result :- 0, 1, 3, 2, 4, 6, 5

② Now top is 4
check if it has any unvisited adjacent Node
↗ No – pop it out
↘ yes → print and push in the stack

Backtrack krte jao
and if there is any unvisited
pop krte jao
when stack is empty stop

* Types of edges in DFS Traversal
(4) → Tree edge
→ Forward edge
→ Back edge
→ Cross edge

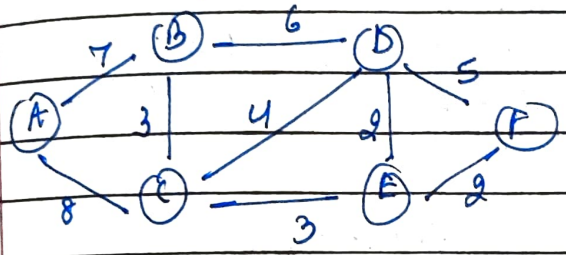
PAGE :
DATE : / /

* MST
Graph :- G(V, E)
Spanning Tree :- G'(V', E')
V' = V (same no. of vertices)
E' ⊂ E
E' = |V| - 1

* For a complete Graph (n^{n-2}) spanning trees are possible
* max (e - n + 1) edges can be removed

no parallel edge
no self loops.

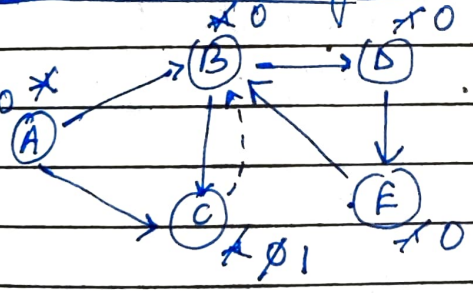
* Prim's Algorithm



worst case
O(V + E)

* Kruskal's.

* How to detect cycle in Directed Graph using DFS



E
D
C
B
A

Visited set :-
A B C D E

forms a cycle :-
B → D → E → B

Parent Map

Vertex	Parent
A	-1
B	A
C	B
D	B
E	D

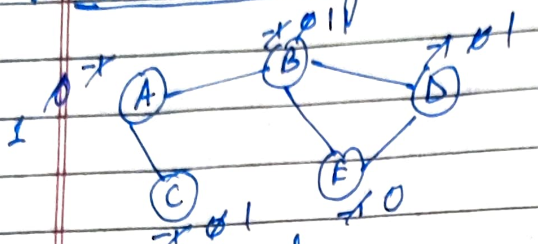
Flag :-
-1 unvisited
0 visited & in stack
1 visited & popped out from stack

* If any vertex finds its adjacent vertex flag 0 it means that graph contain a cycle

* If -1 only then it can be pushed in stack
* If no edge then pop it out

→ DFS
BFS
Topological Sort
Disjoint Sets

★ How to Detect cycle in undirected Graph



-1 = unvisited
0 = Inqueue
1 = visited.

if
f[v] == 0
return
if
f[v] = -1
insert

BFS traversal.

Queue:- | A | B | C | D | E | |

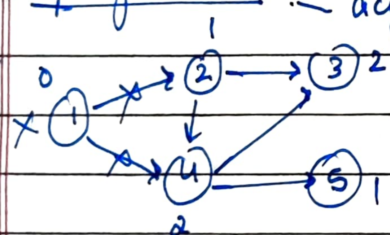
visited set : A B C D

↑ Now at this pt E should be having -1 in order to come ← But it is 0 so there is a cycle.

this means it is the adjacent vertex of some other two

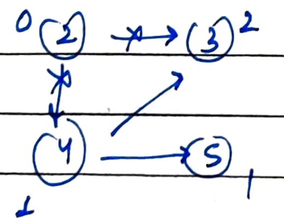
★ Topological Sort: directed acyclic] graph. ← DAG

1.



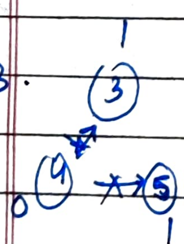
① find Indegree
② 1 → least indegree
③ delete one and delete edges Indegree changes

2.



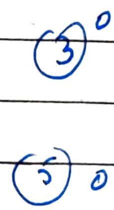
2

3.



4

4.



3

1 2 4 3 5
1 2 4 5 3] Topological ordering

A = 65 a = 97 lower case −
B = 66 b = 98 upper case = 32
⋮
Z = 90 z = 122

PAGE:
DATE: / /

Strings

S [w | e | l | c | o | m | e | \0]
0 1 2 3 4 5 6 7

*** length of string**

```
int main ()
{
  char *S = "welcome";
  int i;
  for(i= 0; S[i]!='\0'; i++)
  { }
  cout << i;
       ↳ length of string
```

*** counting words**

```
int vc=0;
char A[] ="How are you";
  for( int i = 0; A[i]!= '\0';i++)
  { if ( a,e,i,o,u
      vc++;
  }
```

counting words word=1

```
for (
  if (A[i] = =' ' && A[i-1]!=' ')
     word++;
```

*** Reverse**

A [p | y | t | h | o | n | \0]
0 1 2 3 4 5 6

B ~~[p y t h o n]~~

*** finding duplicates**

[| | | | .]
i↑ j↑

found toh usko 0 se replace kardo

*** changing case**

for UC to LC

```
for (int i=0; A[i]!='\0'; i++)
  {
     A[i] = A[i] +32;
  }
```

*** toggle**

```
for (int i=0; A[i]!='\0', i++)
{
  if( A[i] >=65 && A[i] <= 90)
  {  A[i] += 32;
  }
  else if ( A[i] >='a' &&
  {  A[i] -=32;
  }
```

*** validate string**

name [A | n | i | ? | 3 | 2 | 1 | \0]

```
int main()
{
  char *name = "Ani?321";
```

```
valid (char *name)
{ int i;
  for(int i=0...   )
```

65 − 90
97 − 122
48 − 57

f	i	n	a	l	i	n	g	\0
0	1	2	3	4	5	6	7	

→ i

97 - 112
A - Z
a - z

using hash table / array.

make an array of size 26

f = 102 - 97 = 5 ——, increment 5th array at this point

Initialize array elements to zero

```
for(int i = 0; A[i] != '\0'; i++)
{
    H[A[i] - 97] += 1;
}
for(int i = 0; i < 26; i++)
{
    if(H[i] > 1)
    cout << i + 97 ;
    cout << H[i];
}
```

* Check for Anagram
  - ↳ string formed using same alphabets