Date: 25-6-18
MON ☑ TUE ☐ WED ☐ THU ☐ FRI ☐ SAT

Chapter-1

Sparse matrix is a matrix that has many elements with a value zero in order to efficiently utilize the memory specialized algorithm and data structure take advantage of the sparse matrix should be used.

Sparse matrix can easily compressed which in turn can significantly reduce memory used.

Two types of sparse matrix
1. Lower triangular
2. Upper triangular

1. Lower Triangular Matrix : In this first type of sparse matrix all the elements above the main diagonal have a value zero.

Ex:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 5 & 2 & 0 & 0 \\ -9 & 7 & 3 & 0 \\ 4 & 6 & 8 & 4 \end{bmatrix}$$

In a lower triangular matrix.

$A_{i,j} = 0 \qquad i < j$

And nxn lower triangular matrix A has one non-zero element in the first row, two non-zero element in the second row, likewise 'n' non-zero element in the n^{th} row.

Scanned by CamScanner

the mapping between 2d matrix and 1d matrix can be done in any one of the following ways:

1. Rowwise mapping
   A[] = {1, 5, 2, -9, 7, 3, 4, 6, 8, 4}

2. Columnwise mapping
   A[] = {1, 5, -9, 4, 2, 7, 6, 3, 8, 4}

2. Upper Triangular Matrix: All the elements below the main diagonal have the value ZERO.
   $A_{ij} = 0 \quad ; \quad i > j$

3. Tri-diagonal Matrix: In Tri-diagonal matrix, elements with the non-zero values can appear only on the diagonal of immediately above or below diagonal hence in a tri-diagonal matrix.
   $A_{ij} = 0 \quad ;$ where $|i-j| > 1$

Ex:

$$\begin{bmatrix} 4 & 1 & & & \\ 5 & 1 & 2 & & \\ & 9 & 3 & 1 & \\ & & 9 & 2 & 2 \\ & & & 5 & 9 & 9 \\ & & & & 6 & 7 \end{bmatrix}$$

a) In main diagonal, it contains non-zero elements for $i=j$. In all there will be 'n' elements.

Scanned by CamScanner

b) Below the main diagonal, it contains non-zero elements for $i=j+1$, In all there will be $n-1$ element.

c) Above the main diagonal, it consist non-zero elements for $i=j-1$, In all there will be $n-1$ elements.

→ Row wise.
{ 4, 1, 5, 1, 2, 1, 3, 1, 1, 2, 7, 5, 1, 1, 6, 7 }

→ Column wise.
{ 4, 5, 1, 1, 1, 2, 3, 1, 1, 2, 5, 2, 7, 6, 1, 7 }

→ Diagonal wise
{ 5, 1, 1, 5, 6, 1, 1, 3, 2, 1, 7, 1, 2, 1, 2, 7 }

Stack : One of the most important linear data structure of variable size is stack.

5	6	10	12	
0	1	2	3	4

There are 3 operations on stacks.
→ PUSH
→ POP
→ PEEP

A pointer TOP keeps track of topmost element of stack. Initially when the stack is empty the TOP has value ZERO, if stacks contains a single element

Top has a value 1 and so on.

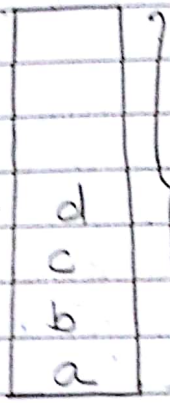
- PUSH (S, TOP, x) (vectors)

1. [check for stack overflow]
   if top ≥ N.
   Printf("Stack overflow")
   Return

2. [Increment Top]
   Top ← Top+1

3. [Insert element]
   S[Top] ← x . S[Top] →

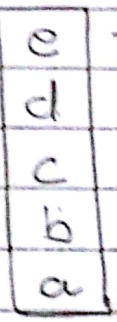
4. [Finished]
   Return



1. a, b, c, d, e, N = 5, Top

i) Top ≥ N
   0 ≥ 5
   False
   Top = 1
   S[1] = a.

ii) Top ≥ N
   1 ≥ 5 False
   Top = 2
   S[2] = b



Scanned by CamScanner

iii) $2 \geq 5$ False
Top = 3
S[3] = c

iv) $3 \geq 5$ False
Top = 4
S[4] = d

v) $4 \geq 5$ False
Top = 5.
S[5] = e

vi) $5 \geq 5$ True.
Stack Overflow

vii) ~~If N = 6, then~~

~~6~~

- Reverse a string using stack.

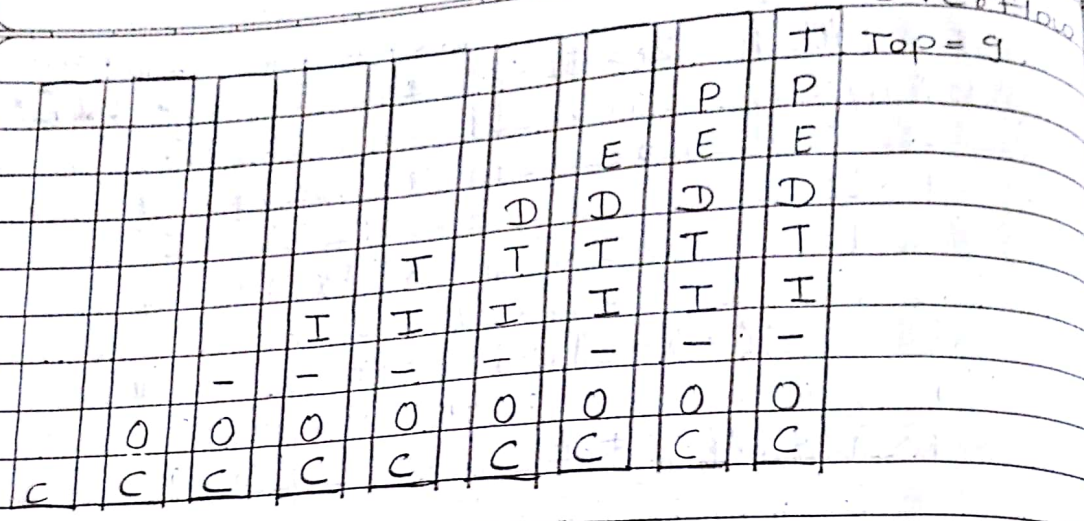
CO-ITDEPT

i) Top = 1
S[1] = C

Scanned by CamScanner

Date.:
MON TUE WED THU FRI SAT

overflow

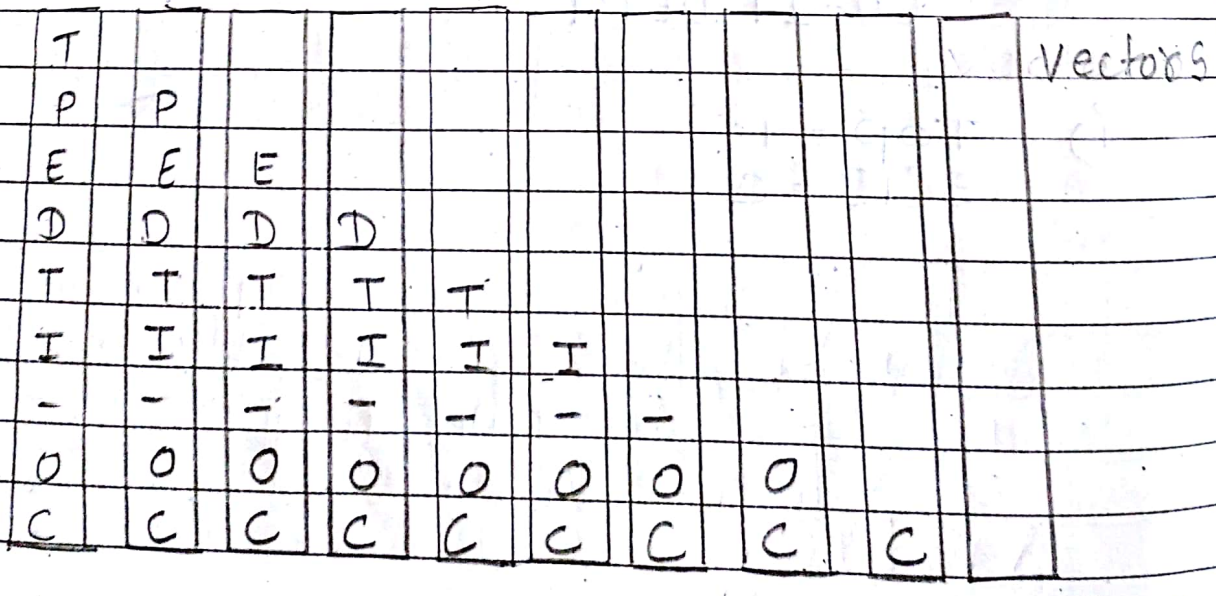


- POP (S, Top)

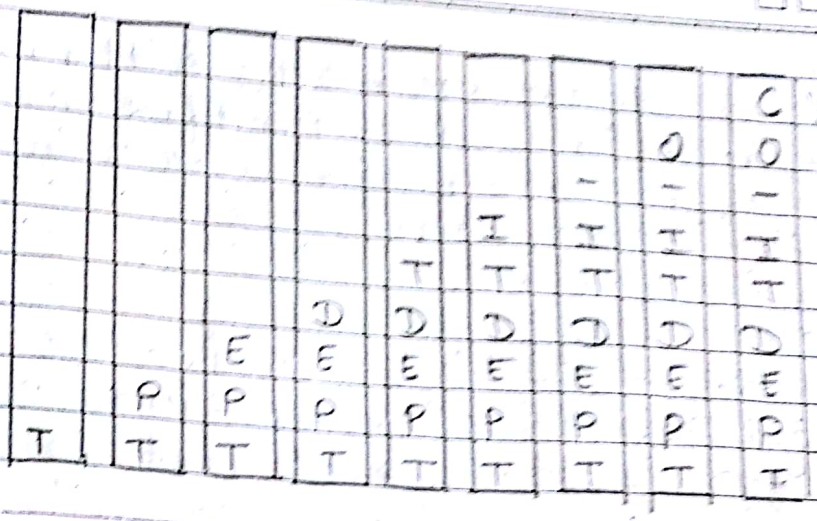
1. [check for underflow]
   if Top == -1
   then write ('STACK UNDEFLOW')

2. [Decrement Pointer]
   Top ← Top - 1

3. [Return element]
   Return [S[TOP] + 1]



Scanned by CamScanner



- PEEP

1. [check for stack underflow]
   if Top-i+1 ≤ 0.
   print ('Stack Underflow on Peep').

2. [Return i^{th} element from top of stack)
   Return S[top]-i+1]

- Infix Notation
  The operator written in betⁿ the operand are called infix notation. Eg: a+b

- Prefix Notation
  The operator written before the operands is called prefix notation or polish notation.
  Eg: +ab

Scanned by CamScanner

- Postfix Notation
  The operator written after operands is called postfix notation or suffix notation or reverse polish notation.
  Eg ab+

- Advantage of Postfix Notation
  - Although infix expressions are easy for us to write but computer finds it difficult to parse. Because they need a lot of information (operator precedence, associativity, brackets) to evaluate the expressions so computers work more efficiently with postfix notations.

Date.:
MON TUE WED THU FRI SAT

★ (A-B) * (C+D)
(AB-) * (CD+)
(AB-)(CD+)*

= A(BC+DE+F*+)G1/+ // Fully Parenthize expression

★ A↑B+C*D%.E↑F
AB↑+C*D%.EF↑
AB↑+DC*%.EF↑
AB↑+DC*EF↑%.
AB↑DC*EF↑%.+

★ A+(B*C-CD/E^F)*G1)*H
A+(B*C-(D/EF^)*G1)*H
A+(B*C-DEF^/*G1)*H
A+(CB*-DEF^/*G1)*H
A+(CB*-DEF^/G1*)*H
A+(BC*DEF^/G1*-)*H
A+(BC*DEF^/G1*-)H*
A(BC*DEF^/G1*-)H*+
ABC*DEF^/G1*-H*+

A+(B*C-(D/^EF)*G1)*H
A+(B*C-/D^EF*G1)*H
A+(*BC-*/D^EFG1)*H
A+-*BC*/D^EFG1*H
A+*-*BC*/D^EFG1H
+A*-*BC*/D^EFG1HA

Scanned by CamScanner

Date.:
MON TUE WED THU FRI SAT

- How to Evaluate Suffix Notation

The method of evaluating suffix expression can be summerized by the four steps which are repeatedly applied until all operator haven been processed.

1) Find the left most operator in the expression
2) Select the two operands immediately to the left of the operator found.
3) Perform the indicated operation
4) Replace the operator & operands with the result.

abc/d*+ a=5, b=4, c=2, d=2

					/					*				
					c			d		d			+	
		b		b		T_1		T_1		T_1			T_2	
a		a		a		a		a		a			a	

$T_1 = b/c = 2$

$T_2 = T_1 * d = 2*2 = 4$

$T_3 = a + T_2 = 5 + 4$

$= 9$

A-B/(C*D^E)

= A-B/(C*DE^)

= A-B/(CDE^*)

= A-BCDE^*/

= ABCDE^*/-

Scanned by CamScanner

Date :

↑
E
D
C
B
A

$D^1E = 2^1 = 2 = T_1$

*
T_1
C
B
A

$T_2 = C * T_1 = 2 * 2$
$= 4$

/
T_2
B
A

$= B / T_2 = 4/4$
$T_3 = 1$

−
T_3
A

$T_4 = 5 - 1 = 4$

Scanned by CamScanner

Date:
MON TUE WED THU FRI SAT

- To Determine Whether an expression is Valid we next associate a rank with each expression.

1. The Rank of a Symbol $S_i = 1$
2. The Rank of an operator $O_j = -1$
3. The Rank of an arbitary sequence of Symbols & operators is the sum of the ranks of the individual symbols & operators.

- Theorem

- A Polish suffix formula is well formed if & only if the rank of the formula is '1'.

1)	a+*b	0	Invalid
2)	a-b*c	1	Valid
3)	a+b/d-	0	Invalid

- Convert unparenthized infix into Suffix using stack.

Step 1. Initialize Stack Contains to the Symbol #.

Step 2. Scan the left most Symbol in the given infix expression & denote it as current Symbol.

Step 3. Repeat through Step-6 while the current input Symbol is #.

step 4. Ren
pr
e
c

step 5. Pu
t

step 6. S

Scanned by CamScanner

Date:
MON TUE WED THU FRI SAT

S.S ≥ C.S

step 4. Remove an output all stacks symbols whose precedence value are greater than or equal to the precedence of the current input symbol.

step 5. Push the current input symbols on to the stack.

step 6. Scan the left most symbol in the infix expression & let it be the current input symbol.

SYMBOL	Precedence	rank
+ −	1	−1
* /	2	−1
a,b,c	3	1
#	0	−

a+b*c-d/e*h

(character scanned)	Contents of stack (right most symbol is top of stack)	RP expression	Rank
	# ← TOP		
a	#a	−	−
+	#+	a	1
b	#+b	a	1
*	#+*	ab	2
c	#+*c	ab	2
−	#−	abc*+	2-1
d	#-d	abc*+	2-1

Scanned by CamScanner

Date:
MON TUE WED THU FRI SAT

/	#-/	abc*+d	3
e	#-/e	abc*+d	3
*	#-*	abc*+d	2
		e/	
h	#-*h	abc*+de	2
		/-	
#	#	abc*+de	1
		/h*-	

a/e*h-b+c

	#		
a	#a	-	-
/	#/	a	1
e	#/e	a	1
*	#*	a e /	1
h	#*h	a e /	1
-	#-	a e / h *	1
b	#-b	a e / h *	1
+	#+	a e / h * b -	1
c	#+c	a e / h * b -	1
#	#	a e / h * b - c +	1

1. [i

2. [i

3. [

4. [

5. [

6.

Scanned by CamScanner

1. [Initialize Stack]
   Top ← 1
   S[TOP] ← '#'

2. [Initialize o/p string & rank count]
   POLISH ←
   RANK ← 0

3. [Get first i/p symbol].
   NEXT ← NEXTCHAR (INFIX)

4. [Translate the infix expression]
   Repeat through step 6 while NEXT ≠ #

5. [Remove symbols with greater or equal precedence from stack].

   Repeat while f(NEXT) ≤ f(S[TOP])
   TEMP ← POP(S, TOP)
   POLISH ← POLISH ○ TEMP
   RANK ← RANK + r(TEMP)
   if RANK < 1
   then write('Invalid') Exit

6. [Push current symbol or to stack & explain next Symbol]
   Call PUSH (S, TOP, NEXT)
   NEXT ← NEXT CHAR (INFIX)

Scanned by CamScanner

Date.:
MON TUE WED THU FRI SAT

7. [Remove remaining element from STACK]
Repeat while S[TOP] ≠ '#'
TEMP ← POP[S, TOP]
POLISH ← POLISH ⊙ TEMP

8. [is the expression valid!]
if RANK = 1
then write ('VALID')
else write ('Invalid') &exit

a↑b↑c*d/e

	#		
a	#a	-	-
↑	#↑	a	1
b	#↑b	a	1
↑	#↑	ab↑	1
c	#↑c	ab↑	1
*	#*	ab↑c↑	1
d	#*d	ab↑c↑	1
/	#/	ab↑c↑d*	1
e	#/e	ab↑c↑d*	1
#	#	ab↑c↑d*e/	1

Scanned by CamScanner

Date:
MON TUE WED THU FRI SAT

Symbol	Input Precedence fun^n f	Stack Precedence fun^n g	Rank fun r
+, −	1	2	−1
*, /	3	4	−1
↑	6	5	−1
Variable	7	8	1
(	9	0	−
)	0	−	−

(a+b↑c↑d)*(e+f/d))

Current Symbol	Stack Contents	R.P Exp	Rank
	(		
(	((	−	−
a	((a	−	−
+	((+	a	1
b	((+ b	a	1
↑	((+ ↑	ab	2
c	((+ ↑ c	ab	2
↑	((+ ↑↑	abc	3
d	((+ ↑↑ d	abc	3
)	(	abcd↑↑+(	1
*	(*	abcd↑↑+	1
(	(*(	abcd↑↑+	1
e	(*(e	abcd↑↑+	1
+	(*(+	abcd↑↑+e	2
f	(*(+f	abcd↑↑+e	2
/	(*(+/	abcd↑↑+ef	3
d	(*(+/d	abcd↑↑+ef	3
)	(*	abcd↑↑+efd/	1
)		+*	

Scanned by CamScanner

(a+b)*(c↑d))

	C		
(	(C	-	-
a	(Ca	-	-
+	(C+	a	1
b	(C+b	a	1
)	(	ab+	1
*	C*	ab+	1
(	C*(	ab+	1
c	C*(c	ab+	1
↑	C*(↑	ab+c	2
d	C*(↑d	ab+c	2
)	(	ab+cd↑*	1

1. [Initialize Stack]
   top = 1
   S[top] = 'C';

2. [Initialize Output String & Rank Count]
   polish = ''
   Rank = 0

3. [Get First Input Symbol]
   Next ← Next char (Infix)

4. [Translate the infix expression
   Repeat through Step-7 while
   Next ≠ '\0'

5. [Remove Symbols with greater precedence]
    If top < 1 then
        write ('Invalid') Exit
    Repeat while F(Next) < G('(Top))
        TEMP ← POP(S, TOP)
        POLISH ← POLISH ⊕ TEMP
        RANK ← RANK + r(TEMP)
        If RANK < 1
        then write ('Invalid') Exit

6. [Are there matching parenthisis?]
    If F(Next) ≠ G'(S(TOP))
    then Call PUSH(S, TOP, Next)
    else
        pop(S, TOP).

7. [Get Next Input Symbol]
    Next ← Nextchar (Infix)

8. [Is the expression valid]
    If Top ≠ 0 or Rank ≠ 1
    then write ('Invalid')
    else
        write ('valid')

[Count]

Scanned by CamScanner

Date.:
MON TUE WED THU FRI SAT

Convert infix into postfix using stack.

a-b/(c*d↑e))

(	(	-	-
a	(a	a	1
-	(-	a	1
b	(-b	ab	2
/	(-/	ab	2
(	(-/(	ab	2
c	(-/(c	ab	2
*	(-/(*	abc	3
d	(-/(*d	abc	3
↑	(-/(*↑	abcd	4
e	(-/(*↑e	abcd	4
)	(	abcde↑*/-	1

Step-1: Reverse the infix exp
Ex: (a+b)+(c-d)
(d-c)*(b+a))

Step-2: Convert ~~in~~ above ⇑ string into RP Notation using stack.

Step-3: Reverse converted string.

Scanned by CamScanner

(e↑d*c)/b-a)

	S		
(	(	-	-
e	(e	-	-
↑	(e↑	e	1
d	(e↑d	e	1
*	(e↑*	ed↑	1
c	(e↑*c	ed↑	1
)	(	ed↑c*	1
/	(/	ed↑c*	1
b	(/b	ed↑c*	1
-	(-	ed↑c*b/	1
a	(-a	ed↑c*b/	1
)	(	ed↑c*b/a-	1

-a/b*c↑de

a+(b*c-(d/e↑f)*g)*h)

h*(g*(f↑e/d)-c*b)+a)

	S		
	(	-	-
h	(h	-	-
*	(h*	h	1
(	(*(	h	1
g	(*(g	h	1
*	(*(*	hg	2
(	(*(*(	hg	2
c	(*(*(c	hg	2
f	(*(*(cf	hg	2
↑	(*(*(↑	hgf	3
e	(*(*(↑e	hgf	3
/	(*(*(↑/	hgfe	4

Scanned by CamScanner

Date.:
MON TUE WED THU FRI SAT

d	C*C*C↑/d	hgfe	4
)	C*C*C	hgfed/↑	3
-	C*C*C-	hgfed/↑	3
C	C*C*C-c	hgfed/↑c	4
*	C*C*C-*	hgfed/↑c	4
b	C*C*C-*b	hgfed/↑cb*-	3
)	C*C*C	hgfed/↑cb*-	3
+	C*C*C+	hgfed/↑cd*-	3
a	C*C*C+a	hgfed/↑cd*-a+	3
)	C*C*(	hgfed/↑cd*-a+	1
)	C*		

**

**+a-*dc↑/defgh

Scanned by CamScanner