ADVANCED ALGORITHMS

UNIT: 1

→ In multiplicaⁿ of $p \times q$ & $q \times r$ matrices,
no. of multiplicatⁿs = $p \times q \times r$

Matrix Chain Multiplicatⁿ:

→ minimise no. of multiplications
→ Determine the way matrices are parenthesised.

Q) $A_{2\times10}$, $B_{10\times50}$, $C_{50\times20}$

$B \times C$

$A(BC) \Rightarrow 2\times(10\times50\times20) = 2(120{,}000)$

$A(BC) = 2\times10\times20 = 400$ (~~2 × 20~~ ~~matrix~~)

Total 10400

$(AB)C \Rightarrow (2\times10\times50)\times20 = \cancel{(1000)20}$

$= 2\times50\times20 = 2000$

⇒ Total $2000 + 1000 = 3000$

$(M1\times M2)\times(M3\times M4)$ $pqr + rst + prt$

$20000 + 8000 + 16000 = \underline{44000}$

$((M1\times M2)\times M3)\times M4$ $pqr + prs + pst$

$20000 + 1000 + 4000$

1000

4000

$\underline{25000}$

$(M1\ M2)(M3\ M4)$

✓ $((M1\ M2)\ M3)\ M4$

$M1\ ((M2\ M3)\ M4)$

$(M1.\ (M2\ M3))\ M4$

$M1\ (M2\ (M3\ M4$

8) $A_1 \times A_2 \times A_3$

$A_1 \underset{}{(2\times3)}$ (P_0 P_1)

$A_2 (3\times4)$ (P_1 P_2)

$A_3 (4\times2)$ (P_2 P_3)

$i \le k < j$

$1 \le k < j$

$A_1 \times (A_2 \times A_3)$

$c[1,1] + c[2,3]$

$0 + 24$ $(P_0 P_1 P_2)$

$M(2\times3)M(3\times2) \Rightarrow 2\times3\times2 = 12$

Total $= 12 + 24 = 36$

$(A_1 \times A_2) \times A_3$

$c[1,2] + c[3,3]$

i) $2\times3\times4 + 0 = 24$

ii) $2\times4\times2 = 16$

Total $\Rightarrow 24 + 16$

$= 40$

$c[1,1] + c[2,3] + P_0P_1P_3$

$c[1,2] + c[3,3] + P_0P_2P_3$

minimisation of value obtained

$$m[i,j] = \begin{cases} 0 & \text{if } i=j \\ \min\limits_{i \le k < j} \{m[i,k] + m[k+1,j] + P_{i-1}P_kP_j\} & \text{if } i<j \end{cases}$$

4/11/22

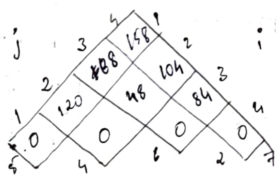
A_1 A_2 A_3 A_4

$P_0 = 5$ $P_1 = 4$ $P_2 = 6$ $P_3 = 2$ $P_4 = 7$

Find $m[1,4]$

$(A_1A_2A_3)A_4$

$((A_1)(A_2A_3))A_4$



A1	A2	A3	A4

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

$i \le k \le j$

$m[1,2] = m[1,1] + m[2,2] + P_0P_1P_2$

$1 \le k < 2 = 5\times4\times6 = 120$

$m[1,3] = \min\{m[1,1] + m[2,3] + P_0P_1P_3,$

$1 \le k < 3$ $\quad m[1,2] + m[3,3] + P_0P_2P_3\}$

$= \min\{0 + m[2,3] + 5\times4\times2,$ $\quad (48 + \cancel{120}\ 40,\ 120+60)$

$120 + 0 + 5\times6\times2\}$ $\quad 88,\ 180$

$m[2,3] = \min\{m[2,2] + m[3,3] + P_1P_2P_3\}$

$2 \le k < 3 = 0 + 0 + 4\times6\times2 = 48$

$m[3,4] = m[3,3] + m[4,4] + P_2P_3P_4 = 6\times2\times7$

$= 84$

$m[2,4] = \min\{m[2,2] + m[3,4] + P_1P_2P_4,$

$m[2,3] + m[4,4] + P_1P_3P_4\}$

$= \min(84 + 4\times6\times7,\ 48 + 4\times2\times7)$

$= \min(84 + 168,\ 48 + 56) = 104$

$m[1,4] = \min\{m[1,1] + m[2,4] + P_0P_1P_4,$

$m[1,2] + m[3,4] + P_0P_2P_4,$

$m[1,3] + m[4,4] + P_0P_3P_4\}$

$= \min(104 + 5\times4\times7,\ 120 + 84 + 5\times6\times7,\ 88 + 5\times2\times7)$

$= \min(88 + 70) = \underline{158}$

eg: $A_1\ A_2 \dots A_6$ for i = 2 to n-l+1

$p_0 = 30 \quad p_1 = 35 \quad p_2 = 15 \quad p_3 = 5,\ p_4 = 10,\ p_5 = 20 \quad p_6 = 25$

→ ((A1)(A2A3))((A4 A5) A6)

5/11/22

LCS (Longest Common Subsequence)

LCS (x, y, i, j)
if x[i] = y[j]

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ 1 + c[i-1, j-1], & x_i = y_j \\ \max(c[i-1,j],\ c[i,j-1]) & \text{if } x_i \neq y_j \end{cases}$$

$$b[i,j] = \begin{cases} \text{"}\nwarrow\text{"} & \text{if } x_i = y_j \\ \text{"}\uparrow\text{"} & x_i \neq y_j \ \& \ c[i-1,j] \geq c[i,j-1] \\ \text{"}\leftarrow\text{"} & x_i \neq y_j \ \& \ c[i-1,j] < c[i,j-1] \end{cases}$$

Determine the LCS for
X = ABCBDAB & Y = BDCABA

		A	B	C	B	D	A	B
~~B~~	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
~~A~~C	0	0	1	2	2	2	2	2
BA	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

Sequence :
B C A B
B D A B
B C B A
~~B C A B~~

X = BACDB   Y = BDCB

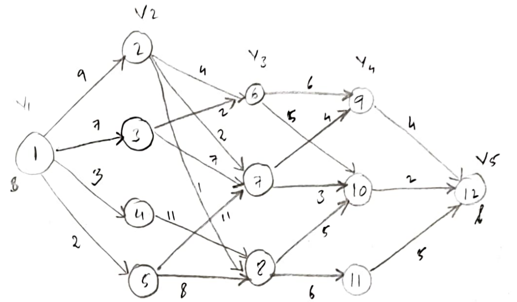
LCS = BCB

		B	A	C	D	B
	0	0	0	0	0	0
B	0	1	1	1	1	1
D	0	1	1	1	2	2
C	0	1	1	2	2	2
B	0	1	1	2	2	3

sequence : BCB
BDB

9/11/22

Finding the Shortest Path in multistage Graph using DP :



$$\text{cost}(i,j) = \min\{c(j,t) + \text{cost}(i+1, t)\}$$

temp; stage; length of edge <j,t>; (prev. one)

Path:
1 → 2 → 7 → 10 → 12
1 → 3 → 6 → 10 → 12

Forward approach:

cost(4, 9) = c(9,12) = 4
cost(4,10) = c(10,12) = 2
cost(4,11) = c(11,12) = 5

cost(3,6) = min {c(6,9) + cost(4,9), c(6,10) + cost(4,10)} = 5+2 = 7
cost(3,7) = min {c(7,9) + cost(4,~~10~~9), c(7,10) + cost(~~10~~ 4,10)} = 3+2 = 5
cost(3,8) = min {c(8,10) + cost(4,10), c(8,11) + cost(4,11)} = 5+2 = 7

cost(2,2) = min {c(2,6) + cost(3,6), c(2,7) + cost(3,7), c(2,8) + cost(3,8)} = { 4+7 = 11, 2+5 = 7, 1+7 = 8 }
cost(2,3) = min {c(3,6) + cost(3,6), c(3,7) + cost(3,7)} = { 2+7 = 9, 7+5 }
cost(2,4) = min {c(4,8) + cost(3,8)} = 11+7 = 18
cost(2,5) = min {c(5,7) + cost(3,7), c(5,8) + cost(3,8)} = { 11+5, 8+7 = 15 }

~~cost~~ min ⇒ ~~8+9~~ 7+9, 7+9, 18+3, 2+15 = 16

21/11/22

Backward approach:

$$bcost(i,j) = \min\{bcos(i-1,k) + c(k,j)\} \quad k \in V_{i-1}$$

$bcost(2,2) = 9$

$bcost(2,3) = 7$

$bcost(2,4) = 3$

$bcost(2,5) = 2$

$$bcost(3,6) = \min\{bcost(2,2) + c(2,6),\ bcost(2,3) + c(3,6)\}$$
$$= \min\{9+4,\ 7+2\} = 9$$

$$bcost(3,7) = \min\{bcost(2,2) + c(2,7),\ bcost(2,3) + c(3,7),\ bcost(2,5) + c(5,7)\}$$
$$= \min\{9+2,\ 7+7,\ 2+11\} = 11$$

Longest Increasing Subsequence (LIS)

I/p arr[] = {3, 10, 2, 11}

LIS [] = {1, 1, 1, 1} (initial)

→ arr[2] > arr[1], LIS[2] = max(LIS[2], LIS[1]+1) = 2

→ arr[3] < arr[1], No change

→ arr[3] < arr[2], "

→ arr[4] > arr[1], LIS[4] = max(LIS[4], LIS[1]+1) = max(1, 1+1) = 2

→ arr[4] > arr[2], LIS[4] = max(LIS[4], LIS[2]+1) = max(1, 2+1) = 3

arr[]	3	10	2	11
LIS[]	1	2	1	3

③ arr = [10, 22, 9, 33, 21, 50, 41, 60]

LIS = [1, 1, 1, 1, 1, 1, 1, 1] For i=1

LIS = [1, 2, 1, 2, 2, 2, 2, 2] For i=2

LIS = [1, 2, 1, 3, 2, 3, 3, 3] For i=4

LIS = [1, 2, 1, 3, 2, 4, 4, 4] For i=6

LIS = [1, 2, 1, 3, 2, 4, 4, 5] For i=8

sequence: 10, 22, 33, 50, 60
41

Rod Cutting Problem

$C(i) = \max \{ v_k + C(i-k) \}$

or $\text{cutRod}(n) = \max(\text{price}[i] + \text{cutRod}(n-i-1))$

Length :	1	2	3	4	5	6	7	8
Price :	1	5	8	9	10	17	17	20
len(i) :	1	2	3	4	5	6	7	8
r[i] Optimal :	1	5	8	10	13	17	18	22
split from	1	2	3	2	2,3	6	1,2,7	2,6

$C(1) = 1$

$C(2) = \max \begin{cases} v_1 + c(1) = 1+1 = 2 \\ v_2 = 5 \end{cases}$

$c(3) = \max \begin{cases} v_1 + c(2) = 1+5 = 6 \\ v_2 + c(1) = 5+1 = 6 \\ v_3 = 8 \end{cases}$

$c(4) = \max \begin{cases} v_1 + c(3) = 1+8 = 9 \\ v_2 + c(2) = 5+5 = 10 \\ v_3 + c(1) = 8+1 = 9 \\ v_4 = 9 \end{cases}$

$c(5) = \max \begin{cases} v_1 + c(4) = 1+10 = 11 \\ v_2 + c(3) = 5+8 = 13 \\ v_3 + c(2) = 8+5 = 13 \\ v_4 + c(1) = 9+1 = 10 \\ v_5 = 10 \end{cases}$

$c(6) = \max \begin{cases} v_1 + c(5) = 1+13 = 14 \\ v_2 + c(4) = 5+10 = 15 \\ v_3 + c(3) = 8+8 = 16 \\ v_4 + c(2) = 9+5 = 14 \\ v_5 + c(1) = 10+1 = 11 \\ v_6 = 17 \end{cases}$

$c(7) = \max \begin{cases} v_1 + c(6) = 1+17 = 18 \\ v_2 + c(5) = 5+13 = 18 \\ v_3 + c(4) = 8+9 = 17 \\ v_4 + c(3) = 9+8 = 17 \\ v_5 + c(2) = 10+5 = 15 \\ v_6 + c(1) = 17+1 = 18 \\ v_7 = 17 \end{cases}$

$c(8) = \max \begin{cases} v_1 + c(7) = 1+17 = 18 \\ v_2 + c(6) = 5+17 = 22 \\ v_3 + c(5) = 8+13 = \cancel{18}\ 21 \\ v_4 + c(4) = 9+10 = \cancel{18}\ 19 \\ v_5 + c(3) = 10+8 = 18 \\ v_6 + c(2) = 17+5 = 22 \\ v_7 + c(1) = 17+1 = 18 \\ v_8 = 20 \end{cases}$

22/11/22

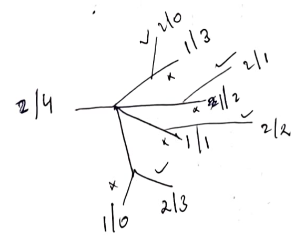
Bottom-up approach: $q = \max(q, p[i] + r[j-i])$

Egg Dropping Puzzle

n eggs, k floors

$$\text{eggdrop}(n,k) = 1 + \min \{ \max(\overbrace{\text{eggdrop}(n-1, x-1)}^{\text{egg breaks}}, \overbrace{\text{eggdrop}(n, k-x)}^{\text{egg doesn't break}}), \; x \text{ in } 1:k \}$$

In worst case, egg needs to be dropped x times.

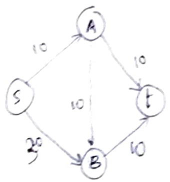


Egg/Floors	0	1	2	3	4	5	6
1	0	1	2	3	4	5	6
2	0	1	2	2	3	3	3
3	0	1	2	2	3	3	3

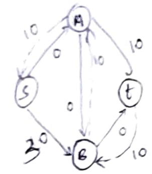
30/11/22

Maximum Flow :

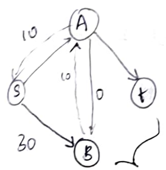
To calculate max-flow path.



→



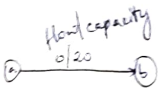
→ But 30 path gives max-flow, so



move -10 up cancels out

this has to have → 10

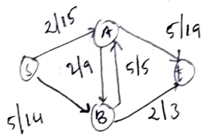
eg:

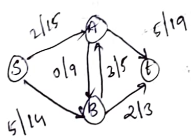


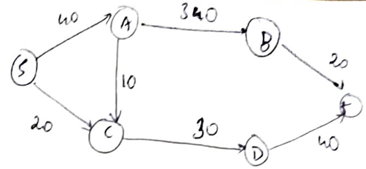
O(fE) E → number of edges

f → no. of times augmented path is found.

- Residual graph
- FF considers reverse paths using residual graph until no augmented ~~graph~~ path can be found.







Path

1) S → A → C → D → t

~~Max flow~~ : 50

2) Trace max flow of each path & add them

Min s-t cut :

s-t cut :→ cut the graph s.t. s & t are on opp. sides.

→ Find edges weights of all edges that have been cut.

2/12/22 → Min-cut gives max-flow.

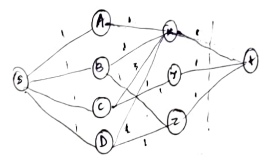
Multiple Sources or Sinks :

→ Modify graph to create single supersource & supersink (combine sources & sinks)

Applicatn - Bipartite Matching :

eg: Maximise the no. of marriages

let w=1

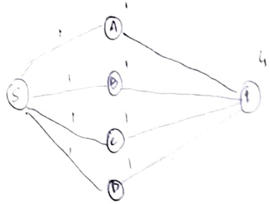


S → A → X → t

S → B → Z → t

S → C → Y → t

or S → D → Z → t



1

2

3

4

Multi-threaded Algorithms

1/12/22

eg: Fibonacci

```
P fib (n)          (parallel fib)
    if n ≤ 1
        return n
    else   x = spawn P-fib(n-1)
           y = P-fib(n-2)
           sync
           return x+y
```

spawn → (fork)

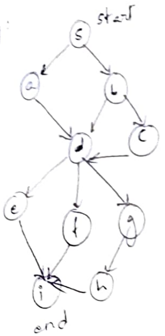
sync → merge 2 child processes

parallel ⇒

its upto to the scheduler to decide if to assign processor to child processes or not

DAG (Directed acyclic graph)

eg:



0 → operations

→ - dependence edges

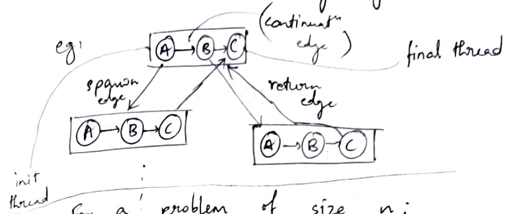
if 2 vertices have no edge in between → can run in parallelism

Continuation edge: (u,v) connects thread u to its successor v within same procedure instance.

- When thread u creates new child thread v → (u,v) is spawn edge

- Return edge → on using sync

eg:



for a problem of size n;

Span (S) T_∞: No. of vertices on the longest directed path from start to finish in the computation DAG.

(the critical path)

Work (W) or $T_1(n)$: Total time to execute the entire computat on one processor. Defined as no. of vertices in the computation DAG.

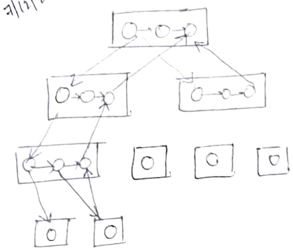
$T_p(n)$: Total time to execute entire comput with p processors.

Speed up: T_1/T_p : How much faster it is using p processors

Parallelism: T_1/T_∞ : Max possible speed up.

The run-time if each vertex of DAG has its own processor.

7/12/22



span = 8 time units

work = 17 units

Work law

$PT_p \geq T_1$

work law: $T_p \geq T_1/p$

An ideal parallel computer with P processors can do at most P units of work & thus

Span Law:

$T_p \geq T_\infty$

A finite no. of processors cannot outperform infinite processors.

Speed Up & Parallelism:

$\frac{T_1}{T_p}$ $\frac{T_1}{T_\infty}$ provides a limit on the possibility of attaining perfect linear speedup.

Parallelism for eg prob $= \frac{T_1}{T_\infty} = \frac{17}{8} = 2.125$

Speedup $= \frac{T_1}{T_4}$

Scheduling:

Greedy scheduler: assigns as many strands to processors as possible in each time step

processors → P

threads > P → complete step

threads < P → incomplete step

Greedy Scheduler Theorem:

$T_p <= T_1/p + T_\infty$

Slackness:

Slackness $= (T_1/T_\infty)/P$

if its less than 1, we cannot expect a linear speedup.

13/12/22 **Parallel loops:**

Matrix multiplication

```
parallel for i in m[i]
    for j in m[i][j]
        y[i] = ...
```

$$\begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

result: $[y_1 \ y_2 \ y_3]$

```
Mat-vec(A, x) {
    n = A.rows
    let y be new vector of length n
    parallel for i=1 to n
        y_i = 0
    parallel for i=1 to n
        for j=1 to n
            y_i = y_i + a_ij x_j
    return y
}
```

```
Mat-vec-main-loop(A, x, y, n, i, i')
{   if i == i'
        for j=1 to n
            y_i = y_i + a_ij x_j
    else mid = ⌊(i+i')/2⌋
        spawn mat-vec-main-loop(A, x, y, n, i, mid)
        mat-vec-main-loop(A, x, y, n, mid+1, i')
        sync.
```

Race Conditions :

→ determinant , non-determinant

Matrix Multiplication :

$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$, $B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$ $C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11}B_{11} & A_{11}B_{12} \\ A_{21}B_{11} & A_{21}B_{12} \end{pmatrix} + \begin{pmatrix} A_{12}B_{21} & A_{12}B_{22} \\ A_{22}B_{21} & A_{22}B_{22} \end{pmatrix}$$

(8 multiplications ⇒ 8 processors parallel
+ 4 additions.

8 multiplicat^n

$T_1(n) = 8T_1(n/2) + \theta(\cancel{n^2}) = \theta(n^2)$

$A_\infty(n) = A_\infty(n/2) + \theta(1) = \theta(\log n)$

8 mul + 4 addit^n

$A_1(n) = 4A_1(n/2) + \theta(1) = \theta(n^2)$

$M_1(n) = 8M_1(n/2) + A_1(n) = 8M_1(n/2) + \theta(n^2) = \theta(n^3)$

$M_\infty(n) = M_\infty(n/2) + \theta(\log n) = \theta(\log^2 n)$

Parallelism of matrix multiplicat^n

$T_1(n) / T_\infty(n) = \theta(n^3/(\log n)^2)$

14/12/22 Multithreaded mergesort

```
Merge sort (A, p, r)
  if p < r
     q = ⌊(p+r)/2⌋
     spawn mergesort (A, p, q)
     merge sort (A, q+1, r)
     sync
     merge (A, p, q, r)
```

```
p-merge (T, p1, r1, p2, r2, A, p3)
  n1 = r1 - p + 1
  n2 = r2 - p2 + 1
  if n1 < n2
     exch p1 with p2
     xchg r1 with r2
     xchg n1 with n2
  if n1 == 0 return
  else q1 = ⌊(p1 + r1)/2⌋
    q2 = binsearch
         (T[q1], T, p2, r2)
    q3 = p3 + (q1 - p1) + (q2 - p2)
    A[q3] = T[q1]
    spawn pmerge
      (T, p1, q1 - 1, p2, q2 - 1, A, p3)
    p-merge (T, q1 + 1, r1, q2,
             r2, A, q3 + 1)
    sync
```

15/12/22

String Matching Algorithm :

↳ Overlapping suffix lemma

Suppose that x, y, z are strings. λ' $x \sqsupset z$ and $y \sqsupset z$ ($\sqsupset$ → ~~prefix~~ suffix). If $|x| \leq |y|$, then $x \sqsupset y$, if $|x| \geq |y|$ then $y \sqsupset x$. If $|x| = |y|$ then $x = y$.

↳ Naive Algorithm :

```
n ← length [T]
m ← length [P]
~~if~~ for  si < n-m
   for j < m
   if  s[j] == T[i+j] ~~and~~
       ~~naive ()~~ continue

   break

if j == m
   return true
```

$T(n) = \sum_{i=0}^{n} \sum_{j=0}^{m}$

~~$m(n-m)$~~

$m(n-m)$

worst : $(n-m+1)m$

⇒ $O(mn)$

↳ Rabin - Karp algorithm.

eg text : a b c d e f g h i j k

pattern : cdef

h(cdef) = h (3, 4, 5, 6) = 18   abgh also gives 18

h (abcd) = h (1, 2, 3, 4) = 10   fdec also gives 18

instead,

make use of radix.

h(abcd) ⇒ h (1, 2, 3, 4) = 1000 + 200 + 30 + 4 = 1234

h(bcde) h (

but when n>> or m>> multiplicat^ns ↑.

Using a prime number,

$h(abcd) = (1 * 10^3 + 2 * 10^2 + 3 * 10^1 + 4 * 10^0) \bmod 113$

$= (1234) \bmod 113 = 104$

$h(bcde) = (((h(abcd) - (1 \cdot 10^3) \bmod 113) \cdot 10) \bmod 113 + 5) \bmod 113$

$= (((104 - 96) \cdot 10) \bmod 113 + 5) \bmod 113$

$= (80 \bmod 113 + 5) \bmod 113 = 85$

Sanity check $(2000 + 300 + 40 + 5) \bmod 113 = 85$

Boyer Moore

- compare pattern characters to text characters from right to left
- bad symbol table : indicates how much to shift based on the text's char that causes a mismatch
- good suffix table : based on matched part of the pattern
  ↓ no. of ch. that matched

$d_1 = \max\{t(c) - k, 1\}$ } bad symbol shift

$t(c)$ = from longpaul shift

$d_2(k)$ → good suffix shift

$d = \max\{d_1, d_2\}$ (algorithm shifts the pattern by after matching successfully $0 < k < m$ characters)

(Prepare 2 tables
- good suffix table
- shift table)

eg BESS_KNEW_ABOUT_BAOBAB

Shift table :

A	B	O	*
1	2	3	6

d_1

pattern BAOBAB

Good suffix table

k	pattern	d_2
1	BAOBAB	2
2	BAOBAB	5
3	BAOBAB	5
4	BAOBAB	5
5	BAOBAB	5

1)

B E S S _ K N E W _ A B O U T _ B A O B A B S
B A O B A B

$d_1 = t_1(K) - 0 = 6$ BAOBAB

BAOBAB

$d_1 = t_1(\_) - 2 = 4$

$d_2 = 5$

$d = \max(4, 5) = 5$

BAOBAB

$d_1 = t_1(\_) - 1 = 5$

$d_2 = 2$

$d = 5$

2)

CAT_NAMED_MEOW_SAID_MEOWMEOW
MEOWMEOW
MEOWMEOW
MEOWMEOW
MEOWMEOW
MEOWMEOW

A C

$d_1 = t(E) - 0$

$d_2 =$

(~~A C D E I M N O S T W~~)

MEOWMEOW

	E	M	O	W	*
d_1	2	3	1	4	8

k	MEOWMEOW	d_2
1	MEOWMEOW	4
2	MEOWMEOW	4
3	EOW	4
4	MEOW	4
5	WMEOW	8
6	OWMEOW	8
7	EOWMEOW	8
8	MEOWMEOW	8

6/1/23 # UNIT-4 - LINEAR PROGRAMMING

Q ₹ 20 → book

₹ 18 → calc

$5x + 4y \leq 27000$

$5x + 15y \leq 30 \times 24 \times 60$
43200

$20x + 18y = z = 1,10,945.29$

$y = 1472.72$

$x = 4221.81$

Standard & Slack form

Standard: in this form, the linear program is the maximisation of the linear function subject to linear inequalities

Slack: a linear program is the maximizatⁿ of a linear functⁿ subject to linear inequalities

Converting to standard form

1) If objective fn is minimize ⇒ maximise

minimize $4x + y + z$ ⇒ max $-4x - y - z$

2) Variable does not have non-negative constraint

⇒ 2 vars with non-~~zero~~ negative constraints

3) There may be equality constraints having an equal sign instead of a less-than-or-equal-to sign

10/1/23 Converting a LP in slack form: Maximise $2x_1 - 3x_2 + 3x_3$

$x_1 + x_2 - x_3 \leq 7$

$-x_1 - x_2 + x_3 \leq -7$

$x_1 - 2x_2 + 2x_3 \leq 4$

$x_1, x_2, x_3 \geq 0$

⇒

$x_4 = 7 - x_1 - x_2 + x_3$

$x_5 = -7 + x_1 + x_2 - x_3$

$x_6 = 4 - x_1 + 2x_2 - 2x_3$

$x_1, x_2, x_3, x_4, x_5, x_6 \geq 0$

⇒ $z = 2x_1 - 3x_2 + 3x_3$

$x_4 = 7 - x_1 - x_2 + x_3$

$x_5 = -7 + x_1 + x_2 - x_3$

$x_6 = 4 - x_1 + 2x_2 - 2x_3$

Simplex Method:

For n variables, each constraint defines a half-space in n-dimensional space. The feasible region formed by intersection of these half-spaces is called simplex.

Steps

→ Move from vertex to vertex, looking for optimal solution

→ find a vertex on polytype

Here Non basic var = $\{x_1, x_2, x_3\}$

basic var = $\{x_4, x_5, x_6\}$

eg: Step 0 Maximise

$3x_1 + x_2 + 2x_3$

$x_1 + x_2 + 3x_3 \leq 30$

$2x_1 + 2x_2 + 5x_3 \leq 24$

$4x_1 + x_2 + 2x_3 \leq 36$

$x_1, x_2, x_3 \geq 0$

⇒

Step 1 to Slack form

$z = 3x_1 + x_2 + 2x_3$

$x_4 = 30 - x_1 - x_2 - 3x_3$

$x_5 = 24 - 2x_1 - 2x_2 - 5x_3$

$x_6 = 36 - 4x_1 - x_2 - 2x_3$

$x_1, x_2, x_3, x_4, x_5, x_6 \geq 0$

Step 2: Initial Basic Solution

$(x_1\ x_2\ x_3, x_4, x_5, x_6) = (0, 0, 0, 30, 24, 36)$

⑧ maximise ~~x_1~~ $x_1 + x_2$

$4x_1 - x_2 \le 8$

$2x_1 + x_2 \le 10$

$5x_1 - 2x_2 \ge -2$

$x_1, x_2 \ge 0$

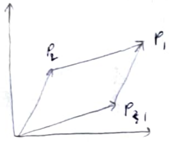
17/1/23

UNIT-5 COMPUTATIONAL GEOMETRY

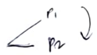
clockwise - counter clockwise:

$p_1 \times p_2 = \det\begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = x_1 y_2 - x_2 y_1$

$= -p_2 \times p_1$



p1 (20, 10)
p2 (10, 20) $\rightarrow$ p1 × p2 = 300



p1 (10, 20)
p2 (20, 10) $\Rightarrow$ p1 × p2 = −300

Shifting of Origin:
(to $P_0(x_0\ y_0)$)

$\Rightarrow$ ~~$p_1 \times p_2$~~ $\rightarrow \begin{pmatrix} x_1 - x_0 & x_2 - x_0 \\ y_1 - y_0 & y_2 - y_0 \end{pmatrix}$

$= (p_1 - p_0) \times (p_2 - p_0)$

→ Turn left & right (11th)

Turn left at P_1
$(P_2 - P_0) \times (P_1 - P_0) < 0$

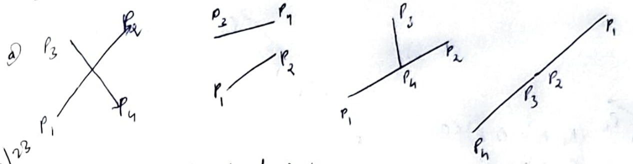
Turn right at P_1
$(P_2 - P_0) \times (P_1 - P_0) < 0$

if $(P_2 - P_0) \times (P_1 - P_0) = 0 \Rightarrow$ collinear

Two segments intersect

5 cases:

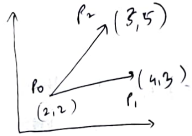
a)



25/1/23

Clockwise or Counterclockwise

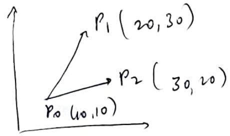
$(P_2 - P_0) \times (P_1 - P_0)$



$\Rightarrow (3-2)(4-2) - (3-2)(5-2)$

$\Rightarrow 1(2) - (1)(3) = -1 < 0$

$\Rightarrow P_2$ is counterclockwise to P_1



$\Rightarrow (P_1 - P_0) \times (P_2 - P_0)$

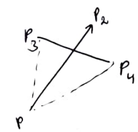
$= (x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0)$

$= 300 > 0$

$\Rightarrow P_2$ is clockwise to P_1

Two segments intersect:

i)



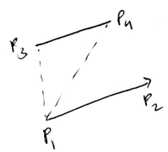
P_1P_2, P_3P_4 intersect:

$(P_3 - P_1) \times (P_2 - P_1) < 0$

$(P_4 - P_1) \times (P_2 - P_1) > 0$

(not complete will have to check wrt P_2 also.)

ii)

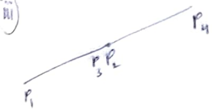


do not intersect:

$(P_3 - P_1) \times (P_2 - P_1) < 0$ or > 0

$(P_4 - P_1) \times (P_2 - P_1) \lessgtr 0$ $\quad > 0$

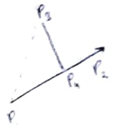
iii)



$(P_3 - P_1) \times (P_2 - P_1) = 0$

$(P_4 - P_1)(P_2 - P_1) = 0$

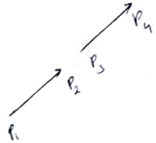
iv)



$(P_3 - P_1) \times (P_2 - P_1) < 0$

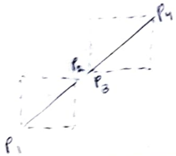
$(P_4 - P_1) \times (P_2 - P_1) < 0$

v)



do not intersect

Bounding Boxes



(max of P_1P_2) & (min of P_3P_4) if they cross-over or intersect $\Rightarrow$ lines intersect.

Code

Direction (P_i, P_j, P_k)

return $(P_k - P_i) \times (P_j - P_i)$

On-segment (P_i, P_j, P_k) {

}

$d_1 = \text{direction}(P_3, P_4, P_1)$ $\quad d_2 = \text{direction}(P_3, P_4, P_2)$

$d_3 = \text{direction}(P_1, P_2, P_3)$ $\quad d_4 = \text{direction}(P_1, P_2, P_4)$

if $((d_1 > 0$ & $d_2 < 0)$ or $(d_1 < 0$ & $d_2 > 0)$ and

$(d_3 > 0$ & $d_4 < 0)$ or $(d_3 < 0$ & $d_4 > 0))$

$\Rightarrow$ intersect

return True

}

on-segment)

$d_1 == 0$ or $d_2 == 0$ or $d_3 == 0$ or $d_4 == 0$

use

else if $d_1 == 0$ & on-segment (P_3, P_4, P_1)

return true

else-if $d_2 == 0$ & on-segment (P_3, P_4, P_2)

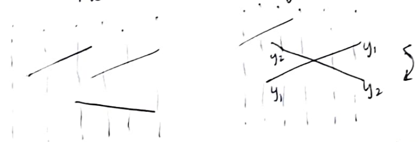
⋮

else return false.

Plane Sweep Algorithm

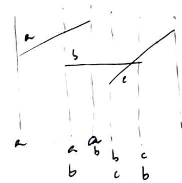
Assumptions:

- No input segment is vertical
- No three input segments intersect at single point.

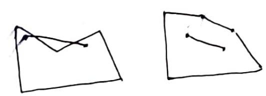


→ vertical lines → place them at starting & ending points

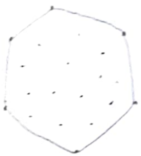
ie



Convex Hulls



Convex hull of a set Q of points is the smallest convex polygon P for which each point in Q is either on the boundary of P or in its interior.



start with

i) Minimum y value

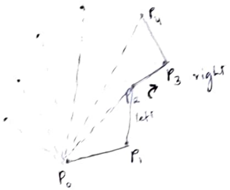
ii) Choose

Graham Scan method

1) Let p_0 be the point in Q with min y-coordinate or leftmost such point in case of tie

2) Let $(p_1, p_2 \dots p_m)$ be the points sorted on basis of angles wrt p_0.

3) Start plotting to next points



⇒ P_2 popped out

3) Algo:

push (P_0, S)
push (p_1, S)
push (p_2, S)

for i = 3 to m
while angle formed by next-to-top(S), top(S) and p makes non-left turn.
pop(P_{i-1}, S)
push(p_i, S)

Jarvis's march (Package Wrapping)

(minimum angle at each point)

Closest Pair Problem

↳ Inclusion problems :

↳ Intersection problems :

↳ Proximity problems :

↳ Construction problems :

---

convex combination: of two distinct points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ is any point $p_3 = (x_3, y_3)$ λ' for some α in range $0 \le \alpha \le 1$,

$x_3 = \alpha x_1 + (1-\alpha) x_2$ &
$y_3 = \alpha y_1 + (1-\alpha) y_2$ } ⇒ $p_3 = \alpha p_1 + (1-\alpha) p_2$

⇒ line segment $\overline{p_1 p_2}$ is a set of convex combinations of p_1 & p_2

UNIT 2

SEE

Spawn: (parent) may continue to execute in parallel with spawned subroutine (child) instead of waiting for the child to complete.

Sync: indicates procedure must wait for all its spawned children to complete

Parallel: loop body can be executed in parallel

Fibonacci

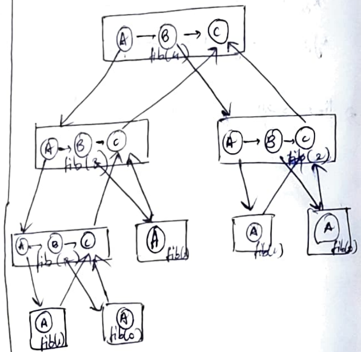
naive approach: $T(n) = T(n-1) + T(n-2) + \Theta(1)$

$T(n) = \Theta(F_n) = \Theta\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right)$

parallel:

```
PFIB(n):
    if n ≤ 1
        return n
    else  x = spawn PFIB(n-1)
          y = PFIB(n-2)
          sync
          return x+y
```

FIB (4) DAG



Span S or $T_\infty(n)$: No. of vertices on longest path

Work W or $T_1(n)$: Total time to execute on one processor

T_p : Total time using p processors

Speed up : $\frac{T_1}{T_p}$

Parallelism : $\frac{T_1}{T_\infty}$ (max possible speed up)

Work law: $PT_p \geq T_1 \Rightarrow T_p \geq \frac{T_1}{P}$ $P \to$ processors

Span law: $T_p \geq T_\infty$

"computer with unlimited no. of processors can emulate a P-processor machine by using just P of its processors.

→ **Greedy scheduler** assigns as many strands to processors as possible in each time step

Theorem : $T_p \leq \frac{T_1}{P} + T_\infty \leq 2\max\left(T_1/P, T_\infty\right) \leq 2T_p^*$

Slackness $= \frac{\left(\frac{T_1}{T_\infty}\right)}{P}$ ie $\frac{\text{Parallelism}}{P}$

if $< 1 \Rightarrow$ we cannot hope to achieve linear speedup

if slackness is big, $P \ll T_1/T_\infty$ then

T_p is approx $\frac{T_1}{P}$

Fibonacci (Parallel): $T_\infty(n) = \max\left(T_\infty(n-1), T_\infty(n-2)\right) + \Theta(1)$
$= T_\infty(n-1) + \Theta(1)$
$= \Theta(n)$

↳ An mt-algo is deterministic iff it does same thing on the same inputs

A multithreaded algo that is intended to be deterministic fails to be.

Determinacy Race: occurs when 2 logically parallel instructions access the same memory & local & atleast one of them performs a write

Matrix-Multiply (C, A, B, n):

if n == 1:

$C[1,1] = A[1,1] \cdot B[1,1]$

else

allocate temp matrix $T[1 \ldots n, 1 \ldots n]$

partition A, B, C, & T into $n/2 \times n/2$ submatrices

spawn Matrix-Multiply ($C_{11}, A_{11}, B_{11}, n/2$)

" " " ($C_{12}, A_{11}, B_{12}, n/2$)

" " " ($C_{21}, A_{21}, B_{11}, n/2$)

C_{22}, A_{21}, B_{12} "

" T_{11}, A_{12}, B_{21} "

" T_{12}, A_{12}, B_{22} "

" T_{21}, A_{22}, B_{21} "

(~~spawn~~) " T_{22}, A_{22}, B_{22} "

sync

Matrix-Add (C, T, n)

11	11	11
12	11	12
21	21	11
22	21	12

11 → 0
12 → 1
21 → 2
22 → 3

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

Matrix-Add (C, T, n):

~~Game~~ if n == 1 : $C[1,1] = C[1,1] + T[1,1]$

else

partition --

Spawn Matrix-Add ($C_{11}, T_{11}, n/2$)

sync

00
11
22
33

$T_\infty(n) = T_\infty(n/2) + \Theta(\log n)$

$T_\infty(n) = \Theta((\log n)^2)$

Parallelism $\dfrac{T_1(n)}{T_\infty(n)} = \Theta\left(\dfrac{n^3}{(\log n)^2}\right)$

Multithreaded Mergesort

mersort (A, p, r)

if p < r

$q = \lfloor (p+r)/2 \rfloor$

spawn Mersort' (A, p, q)

mersort' (A, q+1, r)

sync

merge (A, p, q, r)

$MS_1(n) = 2MS_1(n/2) + \Theta(n)$
$= \Theta(n \log n)$

$MS_\infty(n) = MS_\infty(n/2) + \Theta(n)$
$= \Theta(n)$

parallelism = $\Theta(\log n)$

$PMS_1(n) = 2PMS_1(n/2) + \Theta(n)$

$PMS_\infty(n) = PMS_\infty(n/2) + \Theta(\log^2 n)$

Parallelism = $\dfrac{\Theta(n \log n)}{\Theta(\log^3 n)} = \Theta\left(\dfrac{n}{\log^2 n}\right)$

---

eggDrop (n, k)

n → no. of floor
k → no. of egg

eggDrop (n, k) = 1 + max (eggDrop (n-1, k-1), eggDrop (0, k))

eggDrop ( ) = 1 + max (eggDrop(n-2, k-1), eggDrop (1, k))

⋮

1 + max (eggDrop (0, k-1), eggDrop (n-1, k))

mins of