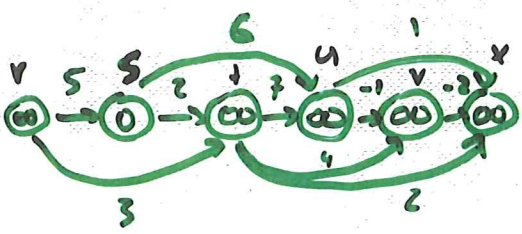
GRAPHS

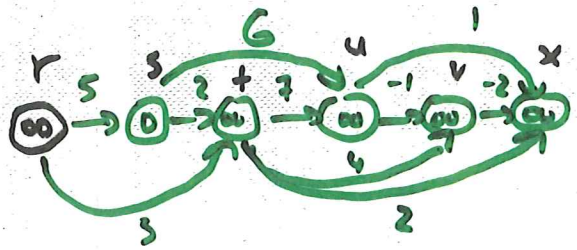
* DAG-SHORTEST-PATHS (PSEUDOCODE)

```
DAG_SHORTEST_PATHS(G,w,s)
    Topologically sort the vertices of G
    INITIALIZE_SINGLE_SOURCE(G,s)
    for each vertex u taken in topologically
        sorted order do
        for each vertex v ∈ Adj[u] do
            RELAX(u,v,w)
```

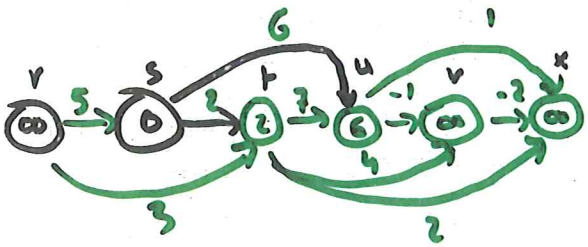
* EXAMPLE



(a)

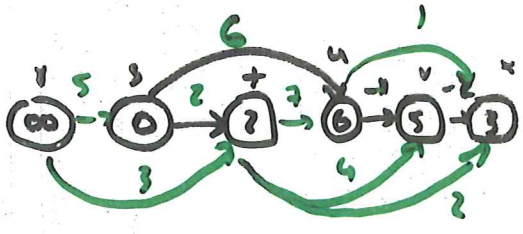


(b)



(c)

.....



(final)

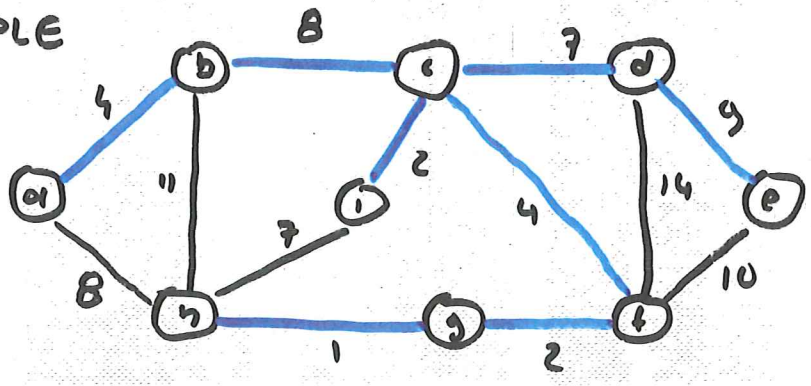
THE VERTICES ARE TOPOLOGICALLY SORTED FROM LEFT TO RIGHT.

GRAPHS

* A FIXED CONNECTED SUBGRAPH CONTAINING ALL THE VERTICES OF AN UNDIRECTED GRAPH (WITH POSITIVE WEIGHTS ON THE EDGES) SUCH THAT THE SUM OF WEIGHTS IS MINIMUM, IS CALLED MINIMUM-COST SPANNING TREE (MCST).

* THIS SUBGRAPH MUST BE A TREE.

* EXAMPLE



* THE PROBLEM

GIVEN AN UNDIRECTED CONNECTED WEIGHTED GRAPH $G=(V,E)$, FIND A SPANNING TREE T OF G OF MINIMUM COST.

GRAPHS

* KRUSKAL'S ALGORITHM (PSEUDOCODE)

```
KRUSKAL(G,w)
  A ← ∅
  for each vertex v ∈ V[G] do
      MAKE-SET(v)
  sort the edges of E by nondecreasing weight w
  for each edge (u,v) ∈ E do {in order by non-}
                             {decreasing weight}
      if FIND_SET(u) ≠ FIND_SET(v) then
          A ← A ∪ {(u,v)}
          UNION(u,v)
  return A
```

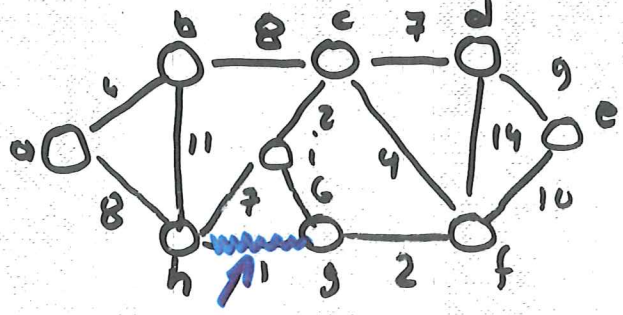
MAKE-SET(v) MAKES A TREE CONTAINING THE VERTEX v.

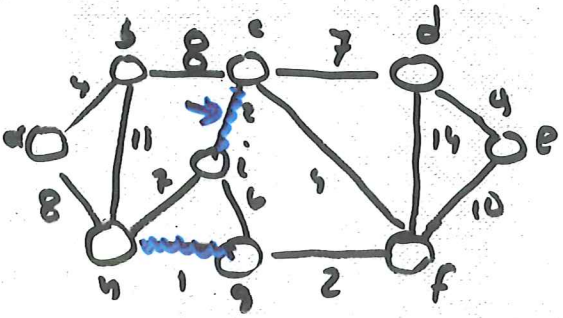
FIND_SET(u) RETURNS A POINTER TO THE REPRESENTATIVE OF THE SET CONTAINING u.

UNION(u,v) UNITES THE SETS THAT CONTAIN u AND v.

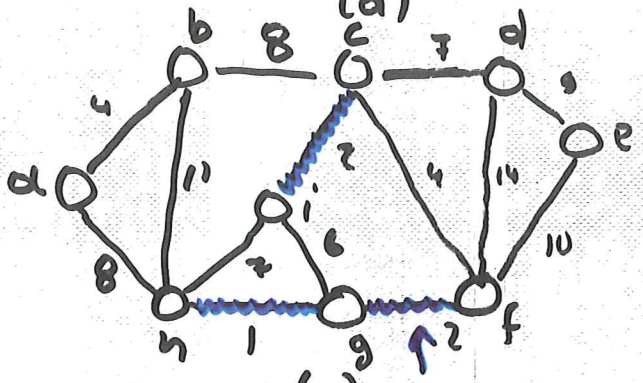
GRAPHS

* EXAMPLE OF THE KRUSKAL'S ALGORITHM

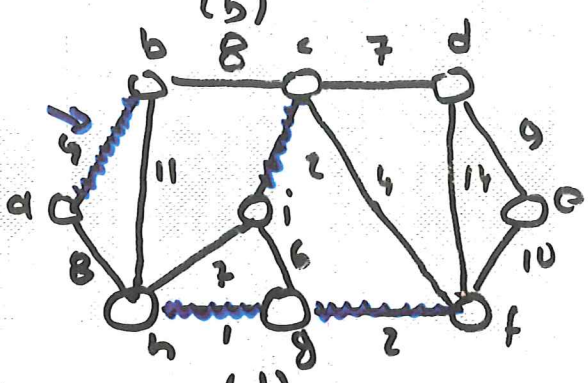




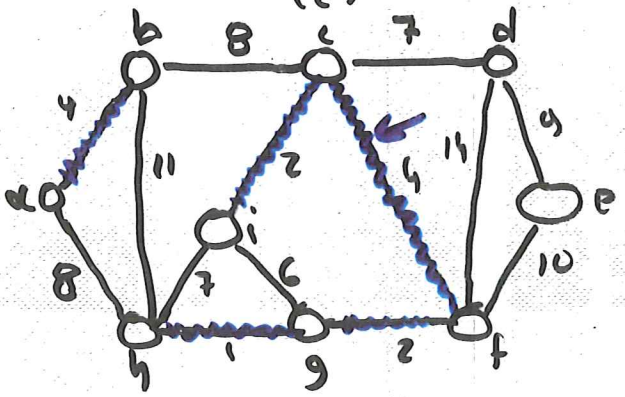
(a)



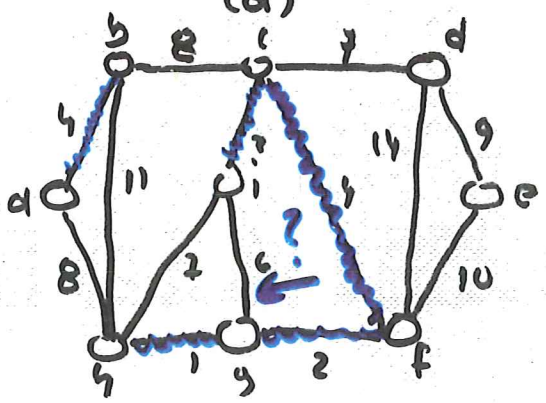
(b)



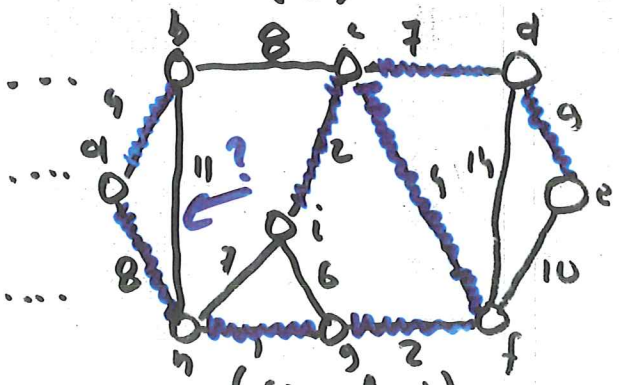
(c)



(d)

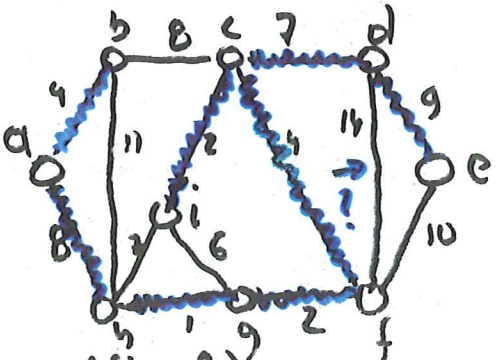


(e)



(final - 1)

(f)



(final)

GRAPHS

* PRIM's ALGORITHM (PSEUDOCODE)

```
PRIM(G, w, s)
    Q ← V[G]
    for each v ∈ Q do
        key[v] ← ∞
    key[s] ← 0
    π[s] ← NIL
    while Q ≠ 0 do
        u ← EXTRACT_MIN(Q)
        for each v ∈ Adj[u] do
            if v ∈ Q and w(u,v) < key[v] then
                π[v] ← u
                key[v] ← w(u,v)
```

Q IS A PRIORITY QUEUE

* THE TOTAL TIME IS $O(V \log V + E \log V) = O(E \log V)$.

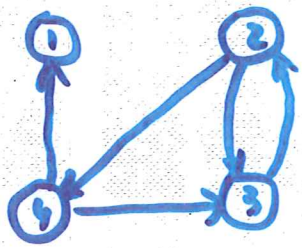
GRAPHS

* GIVEN A DIRECTED GRAPH $G=(V,E)$, THE TRANSITIVE CLOSURE $C=(V,F)$ OF G IS A DIRECTED GRAPH SUCH THAT THERE IS AN EDGE (v,w) IN C IF AND ONLY IF THERE IS A DIRECTED PATH FROM v TO w IN G.

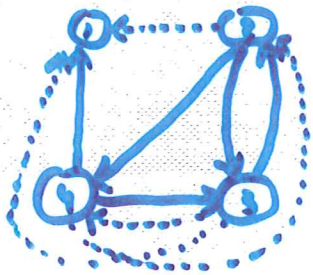
* PROBLEM

  GIVEN A DIRECTED GRAPH $G=(V,E)$, FIND ITS TRANSITIVE CLOSURE.

* EXAMPLE



GRAPH G



TRANSITIVE CLOSURE OF G

* INPUT: AN ADJACENCY-MATRIX $n \times n$ REPRESENTING A DIRECTED GRAPH.

  $A[i,j]$ = TRUE IF THE EDGE (i,j) BELONGS TO THE GRAPH, AND FALSE OTHERWISE.

  $A[i,i]$ = TRUE FOR ALL i.

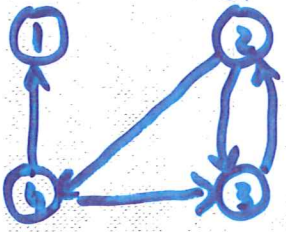
GRAPHS

* TRANSITIVE CLOSURE (PSEUDOCODE)

```
TRANSITIVE_CLOSURE(A)
    for m ← 1 to n do
        for i ← 1 to n do
            for j ← 1 to n do
                if A[i,m] and A[m,j] then
                    A[i,j] ← TRUE
```

* EXAMPLE (TRUE ↦ 1 FALSE ↦ 0)

GRAPH G



INITIAL

$$A=\begin{bmatrix}1&0&0&0\\0&1&1&1\\0&1&1&0\\1&0&1&1\end{bmatrix}$$

m = 1

$$A=\begin{bmatrix}1&0&0&0\\0&1&1&1\\0&1&1&0\\1&0&1&1\end{bmatrix}$$

m = 2

$$A=\begin{bmatrix}1&0&0&0\\0&1&1&1\\0&1&1&1\\1&0&1&1\end{bmatrix}$$

m = 3

$$A=\begin{bmatrix}1&0&0&0\\0&1&1&1\\0&1&1&1\\1&1&1&1\end{bmatrix}$$

m = 4

$$A=\begin{bmatrix}1&0&0&0\\1&1&1&1\\1&1&1&1\\1&1&1&1\end{bmatrix}$$

* IMROVED ALGORITHM

```
IMPR_TRANSITIVE_CLOSURE(A)
    for m ← 1 to n do
        for i ← 1 to n do
            if A[i,m] then
                for j ← 1 to n do
                    if A[m,j] then
                        A[i,j] ← TRUE
```

GRAPHS.

* A SIMILAR PROBLEM TO THE PROBLEM OF TRANSITIVE CLOSURE IS THE ALL-PAIRS SHORTEST PATHS PROBLEM.

* THE PROBLEM

GIVEN A WEIGHTED GRAPH $G=(V,E)$ (DIRECTED OR UNDIRECTED) WITH NONNEGATIVE WEIGHTS, FIND THE MINIMUM-LENGTH PATHS BETWEEN ALL PAIRS OF VERTICES.

* AN INTERESTING ALGORITHM HAS BEEN PROPOSED BY FLOYD AND WARSHALL.

* INPUT: WEIGHT (AN ADJACENCY-MATRIX $n \times n$ REPRESENTING A WEIGHTED GRAPH).
WEIGHT[i,j] IS THE WEIGHT OF THE EDGE (i,j) IF IT EXISTS, OR ∞ OTHERWISE.
WEIGHT[i,i] IS $\emptyset$ FOR ALL i.

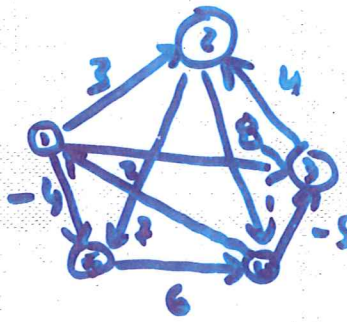
* THE ALGORITHM RUNS IN TIME $O(V^3)$.

GRAPHS

* FLOYD-WARSHALL ALGORITHM

```
FLOYD_WARSHALL(WEIGHT)
    for m ← 1 to n do
        for i ← 1 to n do
            for j ← 1 to n do
                if WEIGHT[i,m] + WEIGHT[m,j]
                    < WEIGHT[i,j] then
                WEIGHT[i,j] ← WEIGHT[i,m]
                    + WEIGHT[m,j]
```

* EXAMPLE



GRAPH G

INITIAL

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

m = 1

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

m = 2

$$WEIGHT = \begin{bmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

. . . .

. . . .

. . . .

m = 5

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

GRAPHS

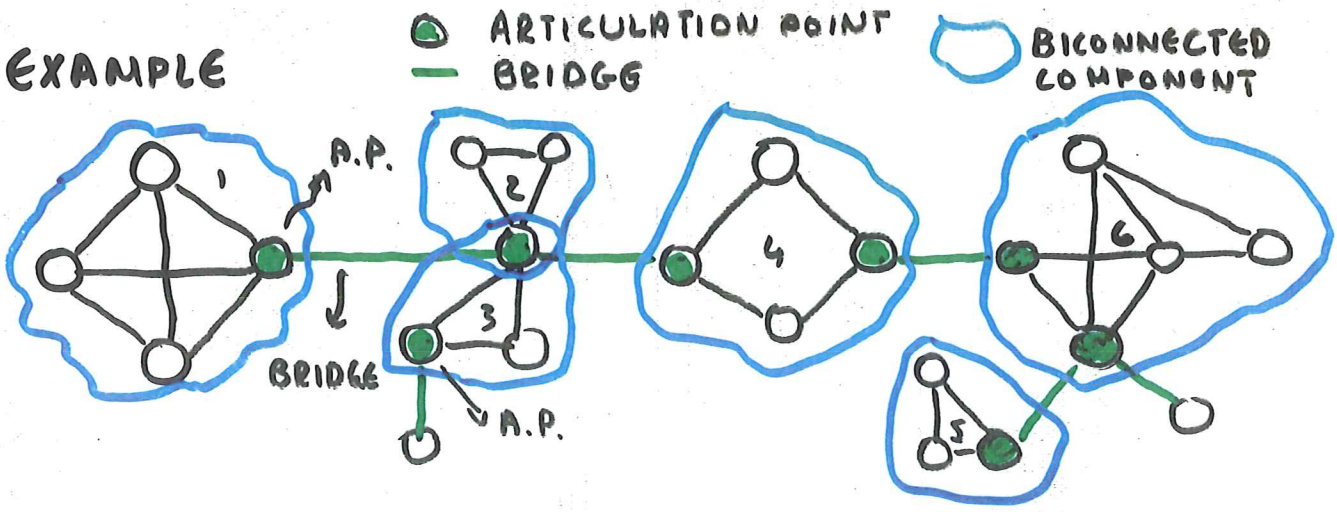
* AN UNDIRECTED GRAPH IS CALLED BICONNECTED IF THERE ARE AT LEAST TWO VERTEX DISJOINT PATHS FROM EVERY VERTEX TO EVERY OTHER VERTEX.

* IN GENERAL, AN UNDIRECTED GRAPH IS CALLED K-CONNECTED IF THERE ARE AT LEAST K VERTEX DISJOINT PATHS BETWEEN EVERY TWO VERTICES.

* AN ARTICULATION POINT OF A GRAPH IS A VERTEX WHOSE REMOVAL DISCONNECTS THE GRAPH.

* A BRIDGE IS AN EDGE WHOSE REMOVAL DISCONNECTS THE GRAPH.

* EXAMPLE



GRAPHS

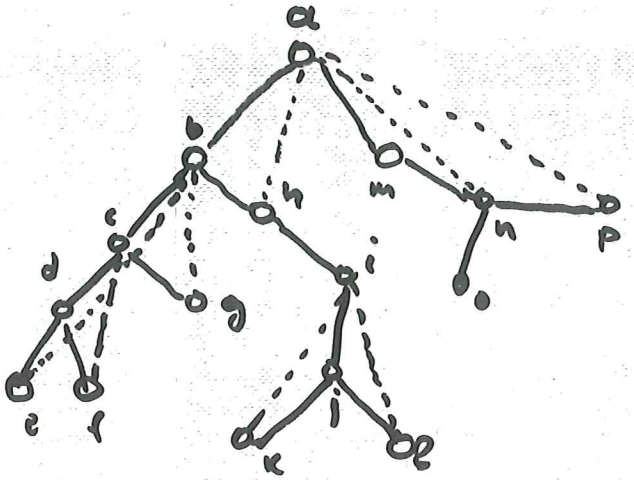
* BICONNECTED COMPONENTS (PSEUDOCODE)

```
BICON_COMPONENTS(G, s, n)   {s IS THE ROOT}
    for every vertex v ∈ V[G] do   {n IS THE NUMBER}
        DFS_Number[v] ← 0          {OF VERTICES}
    DFS_N ← n
    BC(s)

BC(v)
    DFS_Number[v] ← DFS_N
    DFS_N ← DFS_N - 1
    Insert v into the Stack
    High[v] ← DFS_Number[v]
    for every vertex w ∈ Adj[v] do
        Insert edge (v,w) into the Stack
        if w is not the parent of v then
            if DFS_Number[w] = 0 then
                BC(w)
                if High[w] ≤ DFS_Number[v] then
                    Remove all edges and vertices from
                    Stack until v is reached and
                    mark the subgraph they form
                    as a biconnected component
                    Insert v back into the Stack
                High[v] ← max(High[v], High[w])
            else
                High[v] ← max(High[v], DFS_Number[w])
```

GRAPHS

* EXAMPLE



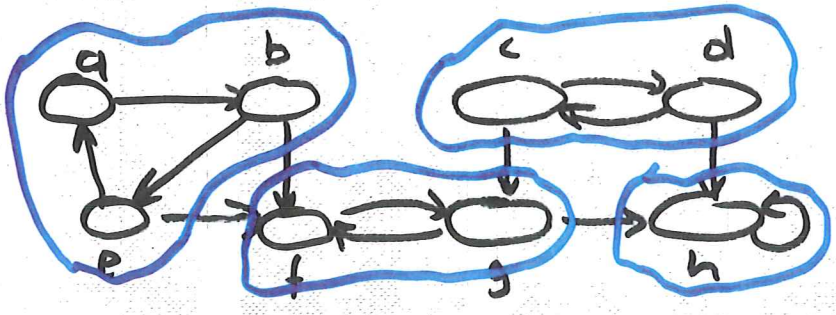
	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
a	16															
b	16	15														
c	16	15	14													
d	16	15	14	13												
e	16	15	14	13	15											
f	16	15	14	15	15											
g	16	15	14	15	15	14										
h	16	15	14	15	15	14										
i	16	15	15	15	15	14										
j	16	15	15	15	15	14	15									
k	16	15	15	15	14	13	15									
l	16	15	15	15	15	14	15	16								
m	16	15	15	15	15	14	15	16	8							
n	16	15	15	15	15	14	15	16	8	7						
o	16	15	15	15	15	14	15	16	8	7	8					
p	16	15	15	15	15	14	15	16	8	8	8					
q	16	15	15	15	15	14	15	16	8	8	8	8				
r	16	15	15	15	15	14	15	16	8	8	8	8				
s	16	15	15	15	15	14	15	16	8	8	8	8				
t	16	15	15	15	15	14	15	16	8	8	8	8				
u	16	15	15	15	15	14	15	16	8	8	8	8	4			
v	16	15	15	15	15	14	15	16	8	8	8	8	4	16		
w	16	15	15	15	15	14	15	16	8	8	8	8	4	16	2	
x	16	15	15	15	15	14	15	16	8	8	8	8	4	16	2	
y	16	15	15	15	15	14	15	16	8	8	8	8	4	16	2	16
z	16	15	15	15	15	14	15	16	8	8	8	8	4	16	2	16
	16	15	15	15	15	14	15	16	8	8	8	8	16	16	2	16
	16	15	15	15	15	14	15	16	8	8	8	8	16	16	2	16

GRAPHS

* A DIRECTED GRAPH IS STRONGLY CONNECTED IF, FOR EVERY PAIR OF VERTICES w AND v, THERE IS A PATH FROM v TO w AND A PATH FROM w to v.

* A STRONGLY CONNECTED COMPONENT IS A MAXIMAL SUBSET OF THE VERTICES SUCH THAT ITS INDUCED SUBGRAPH IS STRONGLY CONNECTED.

* EXAMPLE



* THE TRANSPOSE G^T OF A GRAPH G CONSISTS OF THE SAME VERTICES AND THE SAME EDGES (WITH THEIR DIRECTIONS REVERSED).

* EACH VERTEX BELONGS TO EXACTLY ONE STRONGLY CONNECTED COMPONENT.

GRAPHS

* STRONGLY CONNECTED COMPONENTS ALGORITHM

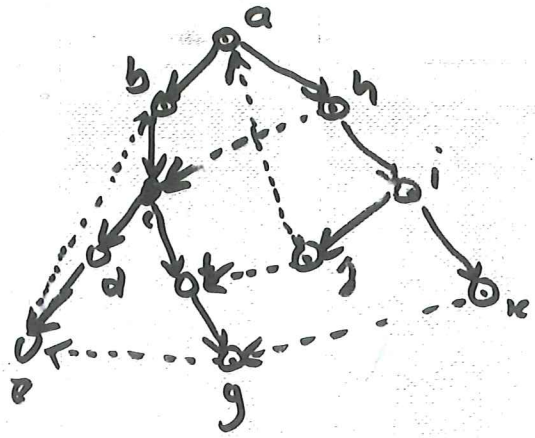
```
STRONGLY_CONN_COMP(G,s,n)

    for every vertex v ∈ V[G] do
        DFS_Number[v] ← 0
        Component[v] ← 0
    Current_Component ← 0
    DFS_N ← n
    while there exists a vertex v with DFS_Number[v]=0 do
        SCC(v)

SCC(v)
    DFS_Number[v] ← DFS_N
    DFS_N ← DFS_N - 1
    Insert v into the Stack
    High[v] ← DFS_Number[v]
    for every vertex w ∈ Adj[v] do
        if DFS_Number[w] = 0 then
            SCC(w)
            High[v] ← max(High[v], High[w])
        else
            if DFS_Number[w] > DFS_Number[v] and
               Component[w]=0 then
                High[v] ← max(High[v], DFS_Number[w])
    if High[v]=DFS_Number[v] then
        Current_Component ← Current_Component +1
        repeat
            remove x from the top of the Stack
            Component[x] ← Current_Component
        until x=v
```

GRAPHS

* EXAMPLE



	a	b	c	d	e	f	g	h	i	j	k
	11	10	9	8	7	6	5	4	3	2	1
a	11										
b	11	10									
c	11	10	9								
d	10	10	9	8							
e	11	10	9	8	10						
d	11	10	9	10	10						
c	11	10	10	10	10						
f	11	10	10	10	10	6					
g	11	10	10	10	10	6	7				
f	11	10	10	10	10	7	7				
c	11	10	10	10	10	7	7				
(b)	11	10	10	10	10	7	7				
a	11	10	10	10	10	7	7				
h	11	10	10	10	10	7	7	4			
i	11	10	10	10	10	7	7	4	3		
j	11	10	10	10	10	7	7	4	3	11	
i	11	10	10	10	10	7	7	4	11	11	
(K)	11	10	10	10	10	7	7	4	11	11	1
i	11	10	10	10	10	7	7	4	11	11	1
h	11	10	10	10	10	7	7	11	11	11	1
(a)	11	10	10	10	10	7	7	11	11	11	1

GRAPHS

* GIVEN A GRAPH, A MATCHING IS A SUBSET OF THE EDGES IN WHICH NO VERTEX APPEARS MORE THAN ONCE.

* A VERTEX THAT IS NOT INCIDENT TO ANY EDGE IN THE MATCHING IS CALLED UNMATCHED.

* A PERFECT MATCHING IS ONE IN WHICH ALL VERTICES ARE MATCHED.

* A MAXIMUM MATCHING IS ONE WITH THE MAXIMUM NUMBER OF EDGES.

* A MAXIMAL MATCHING IS A MATCHING THAT CANNOT BE EXTENDED BY THE ADDITION OF AN EDGE.

* MATCHING IN GENERAL GRAPHS IS A DIFFICULT PROBLEM.

* SEVERAL APPLICATIONS: MEDICAL-STUDENT MATCHING PROBLEM,
STABLE MARRIAGE PROBLEM,
COURSE-HOUR SCHEDULING.

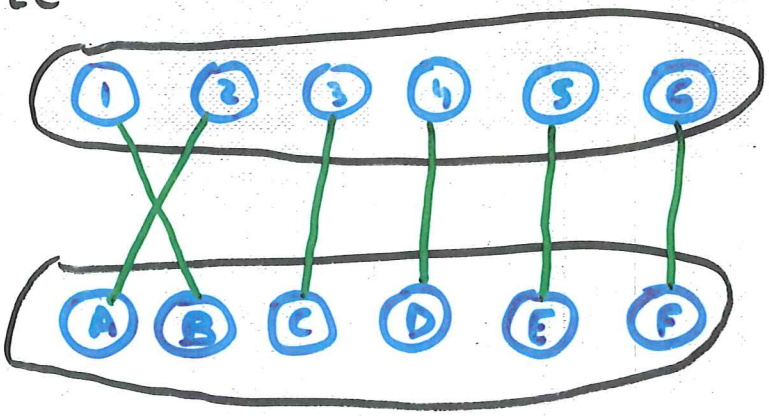
GRAPHS

* BIPARTITE GRAPHS AND MATCHING.

* WE MIGHT BE MATCHING MEN AND WOMEN FOR A DATING SERVICE, JOB APPLICANTS TO AVAILABLE POSITIONS, COURSES TO AVAILABLE HOURS, OR MEMBERS OF CONGRESS TO COMMITTEE ASSIGNMENTS.

* THE GRAPHS ARISING IN SUCH CASES ARE CALLED BIPARTITE GRAPHS.

* EXAMPLE



* WE CAN REDUCE THE MATCHING PROBLEM TO THE NETWORK FLOW PROBLEM. WE HAVE TO ADD A SINK VERTEX AND A SOURCE VERTEX. FURTHERMORE, THE CORRESPONDING EDGES SHOULD BE ADDED.

GRAPHS

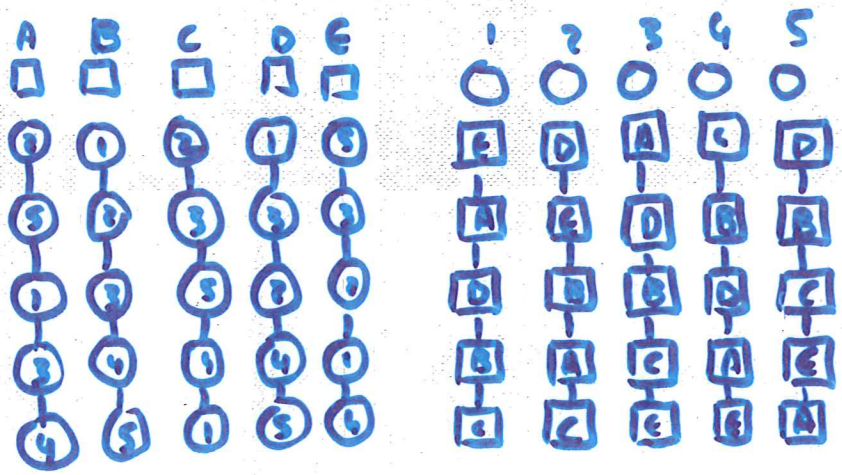
* STABLE MARRIAGE PROBLEM

LET US SUPPOSE THAT WE HAVE N MEN AND N WOMEN WHO HAVE EXPRESSED MUTUAL PREFERENCES. THE PROBLEM IS TO FIND A SET OF N MARRIAGES THAT RESPECTS EVERYONE'S PREFERENCES.

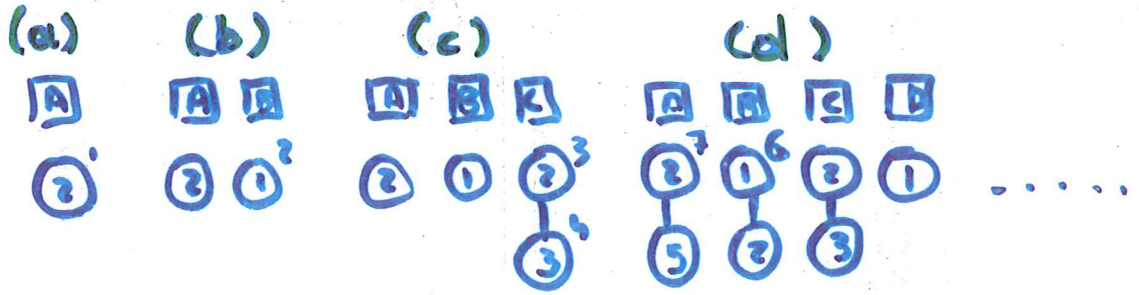
* A SET OF MARRIAGES IS CALLED UNSTABLE IF TWO PERSONS WHO ARE NOT MARRIED (TO EACH OTHER) BOTH PREFER EACH OTHER TO THEIR SPOUSES.

* EXAMPLE

PREFERENCE LISTS



STEPS



GRAPHS

* THE FLOW OF $\underset{\sim}{x}$ ($\underset{\sim}{x}$ CAN BE MATERIAL, LIQUID, ETC.) IS RELATED TO THE RATE AT WHICH THE $\underset{\sim}{x}$ MOVES.

* FLOW NETWORKS CAN BE USED TO MODEL LIQUIDS FLOWING THROUGH PIPES, PARTS THROUGH ASSEMBLY LINES, CURRENT THROUGH ELECTRICAL NETWORKS, INFORMATION THROUGH COMMUNICATION NETWORKS, AND SO FORTH.

* EACH DIRECTED EDGE IN A FLOW NETWORK CAN BE THOUGHT OF AS A CONDUIT FOR THE MATERIAL.

* EACH CONDUIT HAS A STATED CAPACITY.

* MAXIMUM-FLOW PROBLEM

  FIND THE GREATEST RATE AT WHICH $\underset{\sim}{x}$ CAN BE SHIPPED FROM THE SOURCE TO THE SINK WITHOUT VIOLATING ANY CAPACITY CONSTRAINTS.

GRAPHS

* A FLOW NETWORK $G = (V, E)$ IS A DIRECTED GRAPH IN WHICH EACH EDGE $(u, v) \in E$ HAS A NONNEGATIVE CAPACITY $c(u, v) \geq 0$.

* WE HAVE A SOURCE s AND A SINK t.

* FOR CONVENIENCE, IT IS ASSUMED THAT EVERY VERTEX LIES ON SOME PATH FROM THE SOURCE TO THE SINK.

* A FLOW IN G IS A REAL-VALUED FUNCTION f THAT SATISFIES THE FOLLOWING THREE PROPERTIES:

CAPACITY CONSTRAINT: FOR ALL $u, v \in V$, WE REQUIRE

$$f(u, v) \leq c(u, v)$$

SKEW SYMMETRY: FOR ALL $u, v \in V$, WE REQUIRE

$$f(u, v) = -f(v, u)$$

FLOW CONSERVATION: FOR ALL $u \in V - \{s, t\}$, WE REQUIRE $\sum_{v \in V} f(u, v) = 0$

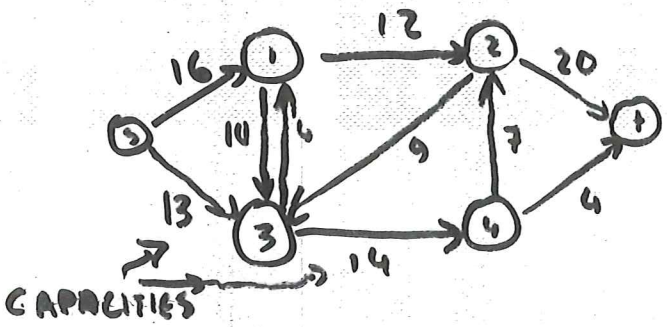
* THE VALUE OF FLOW IS DEFINED AS $|f| = \sum_{v \in V} f(s, v)$.

* THE POSITIVE NET FLOW ENTERING A VERTEX v IS DEFINED BY:

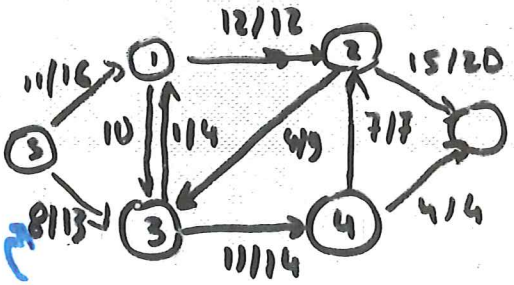
$$\sum_{\substack{u \in V \\ f(u,v) > 0}} f(u,v)$$

* ONE INTERPRETATION OF THE FLOW-CONSERVATION PROPERTY IS THAT THE POSITIVE NET FLOW ENTERING A VERTEX OTHER THAN THE SOURCE OR SINK MUST EQUAL THE POSITIVE NET FLOW LEAVING THE VERTEX.

* EXAMPLE



(a)

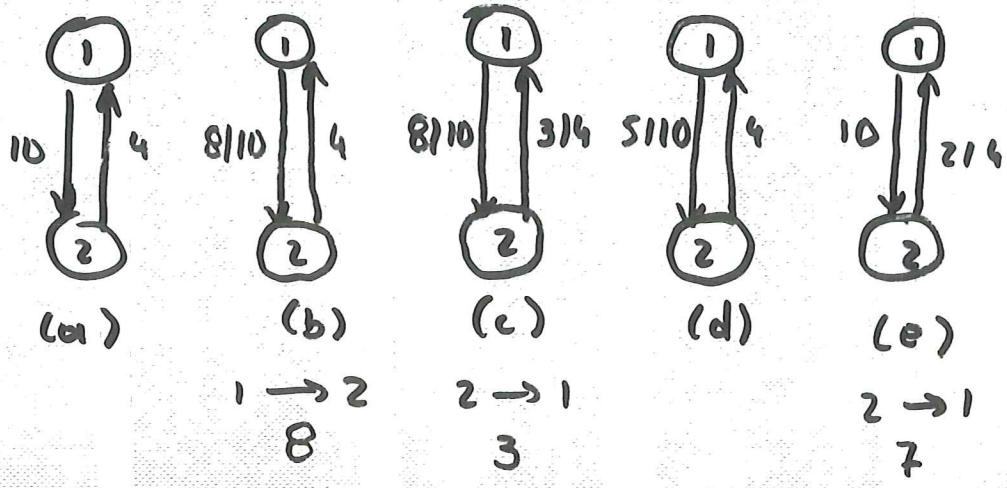


(b)

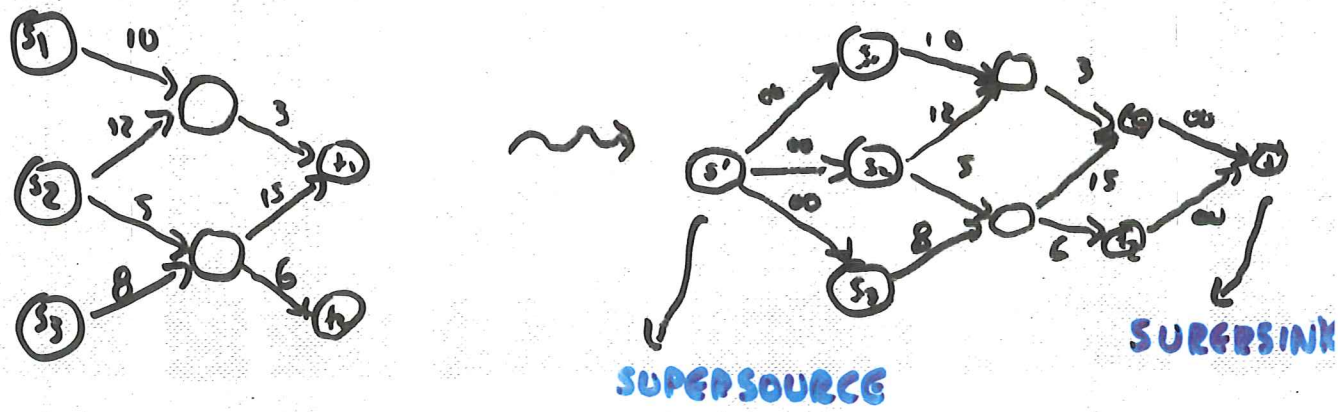
IN FIGURE (b), IF $f(u,v) > 0$ EDGE (u,v) IS LABELED BY $f(u,v) / c(u,v)$. IF $f(u,v) \leq 0$, EDGE (u,v) IS LABELED ONLY BY ITS CAPACITY.

GRAPHS

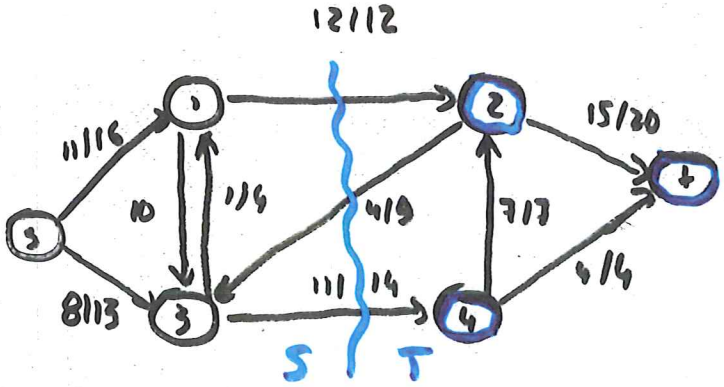
* CANCELLATION



* MULTIPLE SOURCES AND SINKS



* CUT



$f(S,T) = 19$ AND $c(S,T) = 26$

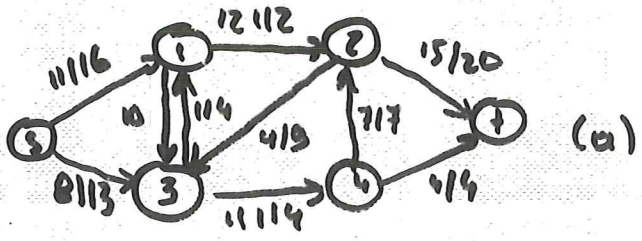
GRAPHS

* GIVEN A FLOW NETWORK $G=(V,E)$ AND A FLOW f, THE RESIDUAL NETWORK OF G INDUCED BY f IS $G_f=(V,E_f)$, WHERE

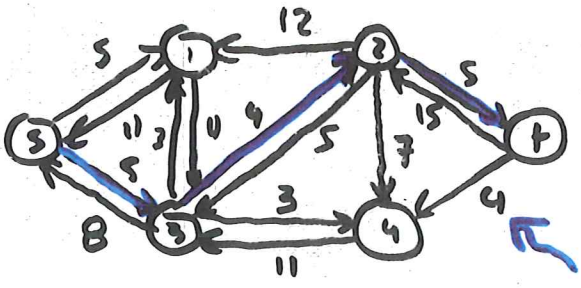
$$E_f = \{(u,v) \in V \times V : c(u,v) - f(u,v) > 0\}.$$

* GIVEN A FLOW NETWORK $G=(V,E)$ AND A FLOW f, AN AUGMENTING PATH p IS A SIMPLE PATH FROM s TO t IN THE RESIDUAL NETWORK.

* EXAMPLE



(a)



(b)

RESIDUAL NETWORK G_f

AUGMENTED PATH

GRAPHS

* FORD-FULKERSON ALGORITHM

```
FORD_FULKERSON(G,s,t)
    for each edge (u,v) ∈ E[G] do
        f[u,v] ← 0
        f[v,u] ← 0
    while there exists a path p from s to t in the residual
    network G_f do
        c_f(p) ← min{c_f[u,v]: (u,v) is in p}
        for each edge (u,v) in p do
            f[u,v] ← f[u,v] + c_f(p)
            f[v,u] ← -f[u,v]
```

$c_F[u,v]$ IS DEFINED AS $c_F[u,v] = c[u,v] - f[u,v]$

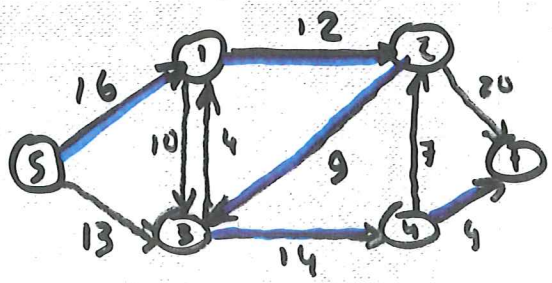
* THE RUNNING TIME OF THE FORD-FULKERSON ALGORITHM DEPENDS ON HOW THE AUGMENTING PATH p IS DETERMINED

* THE ALGORITHM CAN RUN IN POLYNOMIAL TIME.

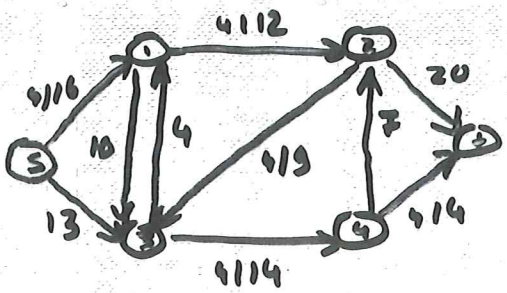
GRAPHS

* EXAMPLE

(a)

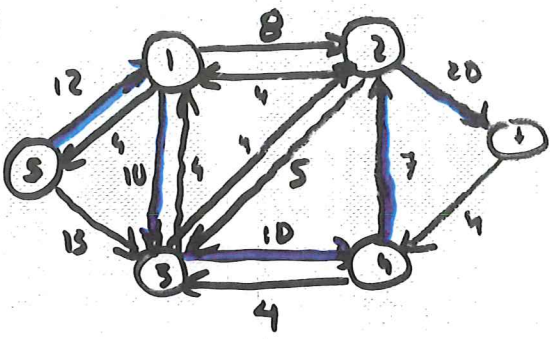


(1)

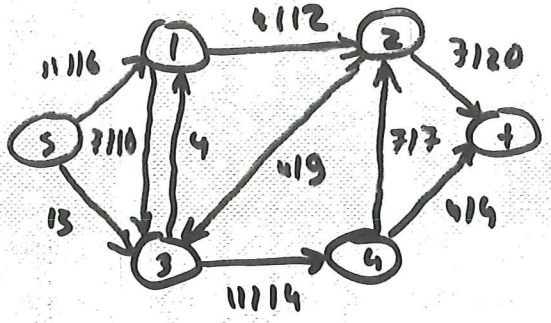


(2)

(b)

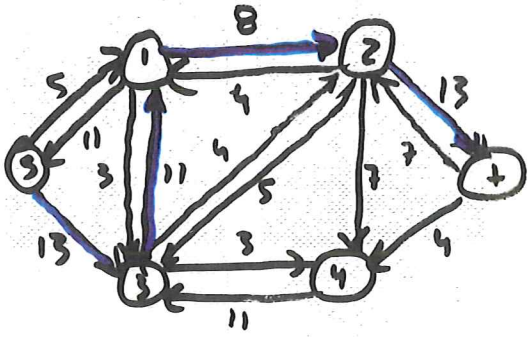


(1)

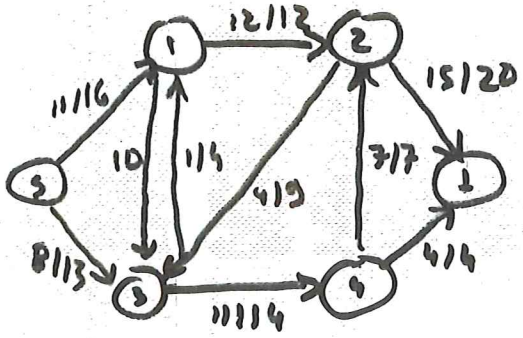


(2)

(c)

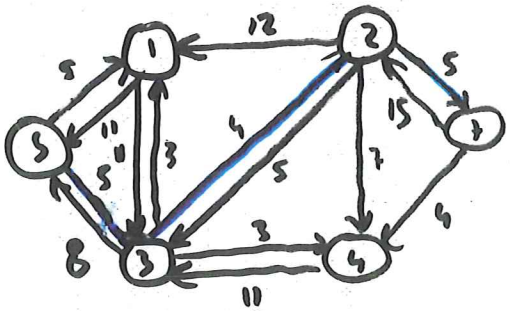


(1)

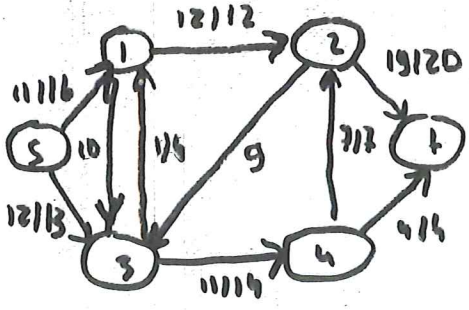


(2)

(d)

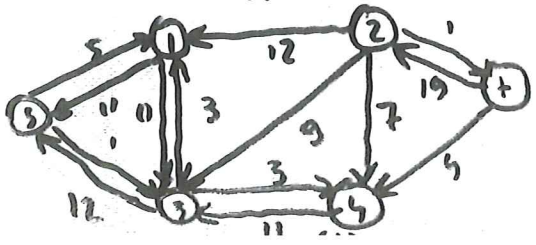


(1)

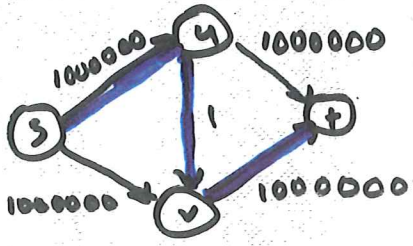


(2)

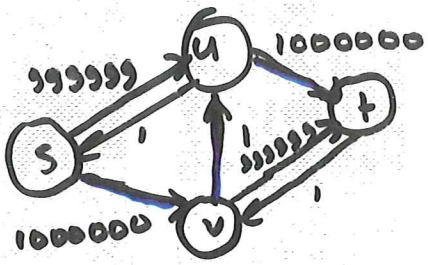
(e)



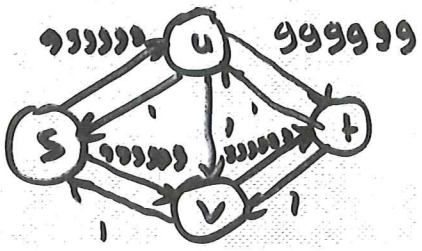
* ANOTHER EXAMPLE



(a)



(b)



(c)

* A MODIFICATION OF THE FORD-FULKERSON ALGORITHM RESULTS INTO THE EDMONDS-KARP ALGORITHM.

GEOMETRIC ALGORITHMS

GEOMETRIC ALGORITHMS

* GEOMETRIC ALGORITHMS ARISE IN THE AREA OF COMPUTATIONAL GEOMETRY.

* GEOMETRIC ALGORITHMS DEAL WITH CONTINUOUS AND DISCRETE ASPECTS OF GEOMETRIC OBJECTS.

* A POINT p IS REPRESENTED AS A PAIR OF COORDINATES (x,y). A FIXED COORDINATE SYSTEM IS ASSUMED.

* A LINE IS REPRESENTED BY A PAIR OF POINTS p AND q AND IT IS DENOTED BY -p-q-.

* A LINE SEGMENT IS REPRESENTED BY A PAIR OF POINTS p AND q AND WE DENOTE THE LINE SEGMENT BY p-q.

* A PATH P IS A SEQUENCE OF POINTS $p_1, p_2, \ldots, p_n$ AND THE LINE SEGMENTS CONNECTING THEM.

GEOMETRIC ALGORITHMS

* A CLOSED PATH IS A PATH WHOSE LAST POINT IS THE SAME AS ITS FIRST POINT.

* A CLOSED PATH IS ALSO CALLED A POLYGON.

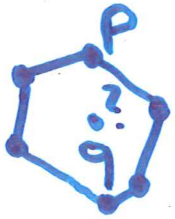
* A SIMPLE POLYGON IS ONE WHOSE CORRESPONDING PATH DOES NOT INTERSECT ITSELF.

* A CONVEX POLYGON IS A POLYGON SUCH THAT ANY LINE SEGMENT CONNECTING TWO POINTS INSIDE THE POLYGON IS ITSELF ENTIRELY INSIDE THE POLYGON.

* PROBLEM

  GIVEN A SIMPLE POLYGON P AND A POINT q, DETERMINE WHETHER THE POINT IS INSIDE OR OUTSIDE THE POLYGON.

* EXAMPLE



GEOMETRIC ALGORITHMS

* ALGORITHM (PSEUDOCODE)

```
POINT_IN_POLYGON_1(P,q)   {P IS A SIMPLE}
 Pick an arbitrary points  {POLYGON AND q}
 outside the polygon       {IS A POINT.}
 Let L be the line segment q-s
 count ← 0
 for all edges ei of the polygon do
     if ei intersects L then {SEVERAL}
          increment count {ASSUMPTIONS ARE
                           {USED.}
 if count is odd then Inside ← TRUE
 else Inside ← FALSE
 return Inside
```

* THE ALGORITHM CAN BE SIMPLIFIED IF THE LINE SEGMENT q-s IS PARALLEL TO ONE OF THE AXES.

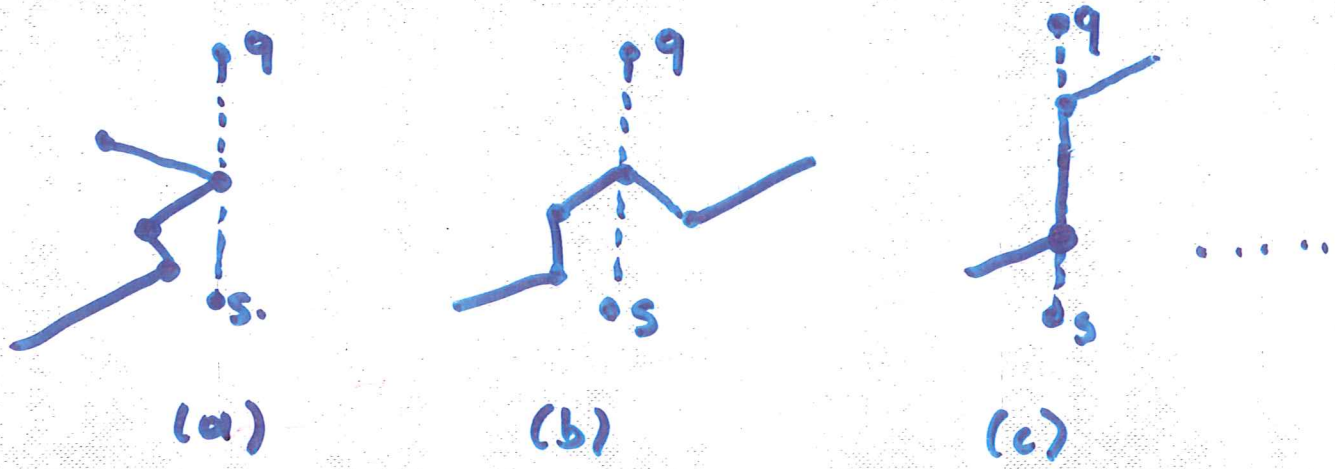
GEOMETRIC ALGORITHMS

* ALGORITHM(PSEUDOCODE)

```
POINT_IN_POLYGON_2(P, q)                    {q=(x0,y0).}
    count←0
    for all edges ei of the polygon do
        if the line x=x0 intersects ei then
                                    {IT IS ASSUMED THAT THE}
                                    {INTERSECTION IS NOT AT A}
                                    {VERTEX NOR IS THE LINE x=x0}
                                    {OVERLAPPING WITH ei.}
            Let yi be the y coordinates of the
            intersection between the line x=x0 and ei
            if  yi < y0 then
                increment count
    if count is odd then Inside← TRUE
    else Inside← FALSE
return Inside
```

GEOMETRIC ALGORITHMS

* SPECIAL CASES



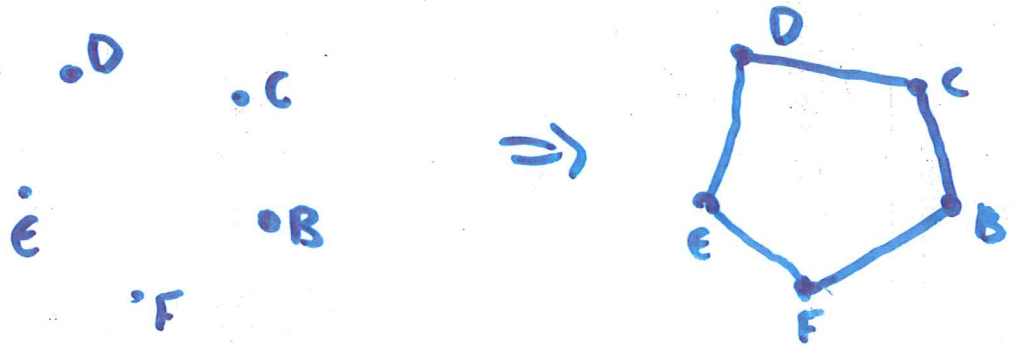
(a) (b) (c) .....

* THE TOTAL TIME IS $O(n)$ (n IS THE SIZE OF THE POLYGON).

* PROBLEM

GIVEN A SET OF n POINTS IN THE PLANE, CONNECT THEM IN A SIMPLE CLOSED PATH.

* EXAMPLE



GEOMETRIC ALGORITHMS

* ALGORITHM (PSEUDOCODE)

SIMPLE POLYGON ($p_1, \ldots, p_n$)

for $i \leftarrow 2$ to n do

compute the angle α_i between

the line $-p_1 - p_i -$ and the x axis

{IT IS MORE DESIRABLE TO SELECT}

{p_1 TO BE THE POINT WITH THE LARGEST}

{x COORDINATE (AND SMALLEST y COORDINAT

{IF THERE ARE SEVERAL POINTS WITH THE

{SAME LARGEST x COORDINATE).}

sort the points according to the angles

$\alpha_2, \ldots, \alpha_n$

P is the polygon defined by the list

of points in sorted order

* THE RUNNING TIME OF THE ALGORITHM IS $O(n \log n)$.

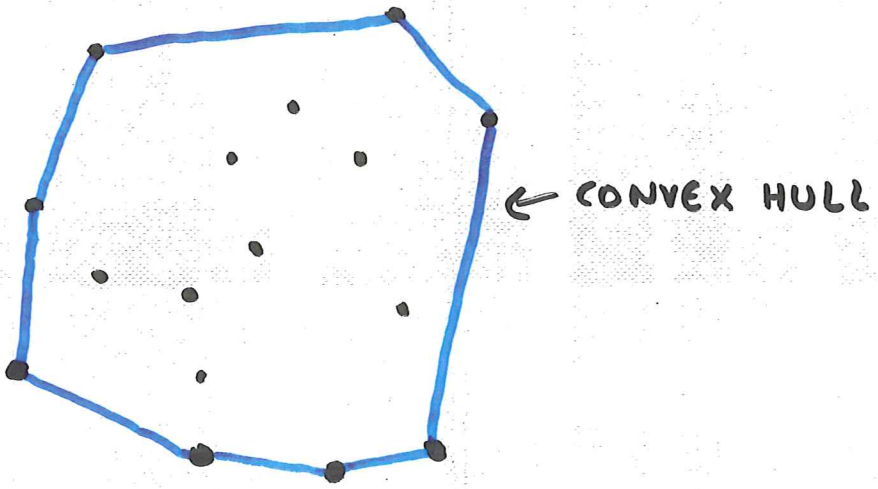
GEOMETRIC ALGORITHMS

* THE CONVEX HULL OF A SET OF POINTS IS DEFINED AS THE SMALLEST CONVEX POLYGON ENCLOSING ALL THE POINTS.

* PROBLEM

  COMPUTE THE CONVEX HULL OF n GIVEN POINTS IN THE PLANE.

* EXAMPLE



* VARIOUS ALGORITHMS. SOME ALGORITHMS WORK WELL WHEN THERE ARE MANY POINTS. OTHERS WORK BETTER IF THERE ARE ONLY A FEW.

GEOMETRIC ALGORITHMS

* GIFT-WRAPPING (PSEUDOCODE)

```
GIFT-WRAPPING(p1, p2, ..., pN)
    Set P to be the empty set
    Let p be the point in the set with the
    largest x coordinate (and the smallest
    y coordinate, if there are several points
    with the same largest x coordinate)
    Add p to P
    Let L be the line containing p which is
    parallel to the x axis
    while P is not complete do
        Let q be the point such that the angle
        between the line -p-q- and L (in
        counterclockwise fashion) is minimal
        among all points
        add q to P
        L ← line -p-q-
        p ← q
```

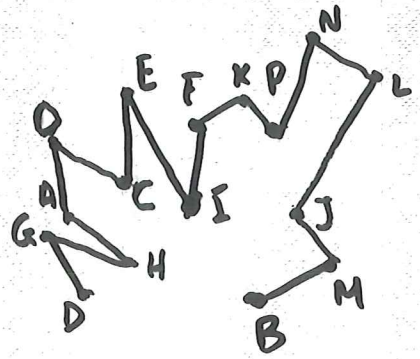
GEOMETRIC ALGORITHMS

* GRAHAM'S SCAN(PSEUDOCODE)

```
GRAHAM'S SCAN(p1, p2, ..., pN)
    Let p1 be the point in the set with the
    largest x coordinate (and smallest y
    coordinate if there are several points with
    the same largest x coordinate)
    Use algorithm SIMPLE.POLYGON to arrange
    the points around p1 in sorted order; let
    the order be p1, p2, ..., pN
    q1 ← p1
    q2 ← p2
    q3 ← p3
    m ← 3
    for k ← 4 to N do
        while the angle between -qm-1-qm- and
        -qm-pk- is ≥ 180° do
            m ← m-1
        m ← m+1
        qm ← pk
```

GEOMETRIC ALGORITHMS

* EXAMPLE OF GRAHAM'S SCAN



(1)

(2)

(3) .....

* THE TOTAL RUNNING TIME IS $O(n \log n)$. n IS THE SIZE OF THE POLYGON.

* THE COMPLEXITY OF THE ALGORITHM IS DOMINATED BY THE SORTING.

* THE RUNNING TIME FOR THE GIFT-WRAPPING ALGORITHM IS $O(n^2)$.

* THE INTERIOR ELIMINATION TECHNIQUE CAN IMPROVE THE CONVEX HULL ALGORITHMS.

* AFTER THE SORT, THE GRAHAM'S SCAN IS A LINEAR TIME PROCESS.

GEOMETRIC ALGORITHMS

* PROBLEM

  GIVEN A SET OF n POINTS IN THE PLANE, FIND A PAIR OF CLOSEST POINTS.

* A STRAIGHTFORWARD APPROACH IS TO CHECK THE DISTANCES BETWEEN ALL PAIRS AND TO TAKE THE MINIMAL ONE.

* THIS SOLUTION REQUIRES $n(n-1)/2$ DISTANCE COMPUTATIONS AND $n(n-1)/2 - 1$ COMPARISONS.

* THE TOTAL NUMBER OF DISTANCE COMPUTATIONS $T(n)$ SATISFIES THE RECURRENCE RELATION $T(n) = T(n-1) + n-1$ $(T(2)=1)$. THIS GIVES US $T(n) = O(n^2)$.

* DIVIDE-AND-CONQUER TECHNIQUES ARE BETTER FOR THIS PROBLEM.

GEOMETRIC ALGORITHMS

* ALGORITHM (PSEUDOCODE)

CLOSEST-PAIR-1($P_1, P_2, .., P_n$) {IT RETURNS THE
{DISTANCE BETWEEN
{THE TWO CLOSEST
{POINTS IN THE SET

Sort the points according to their x coordinates

Divide the set into two equal-sized parts

Recursively, compute the minimal distance in each part

Let d be the minimal of the two minimal distances

Eliminate points that lie farther than d apart from the separation line

Sort the remaining points according to their y coordinates

Scan the remaining points in the y order and compute the distances of each point to its five neighbors

if any of these distances is less than d then
    update d

return d

GEOMETRIC ALGORITHMS

* ALGORITHM (PSEUDOCODE)

CLOSEST_PAIR_2 ($P_1, P_2, \ldots, P_n$)

Sort the points according to their x coordinates

Divide the set into two equal-sized parts

Recursively do the following:

Compute the minimal distance in each part

Sort the points in each part according to their y coordinates

Merge the two sorted lists into one sorted list

Let d be the minimal of the minimal distances

Eliminate points that lie further than d apart from the separation line

Scan the point in the y order and compute the distanc of each point to its five neighbors

if any of these distances is less than d then

update d

return d

GEOMETRIC ALGORITHMS

* PROBLEM

GIVEN A SET OF n HORIZONTAL AND m VERTICAL SEGMENTS IN THE PLANE, FIND ALL THE INTERSECTIONS AMONG THEM.

* ALGORITHM (PSEUDOCODE)

INTERSECTION($(v_1, v_2, \ldots, v_m), (h_1, h_2, \ldots, h_n)$)

Sort all x coordinates in increasing order and place them in Q
$V \leftarrow \emptyset$
while Q is not empty do
  remove the first endpoint p from Q
  if p is the right endpoint of h_k then
    remove h_k from V
  else if p is the left endpoint of h_k then
    insert h_k into V
  else if p is the x coordinate of a vertical line v_i then
    perform a one-dimensional range query for the range $y_B(v_i)$ to $y_T(v_i)$ in V

$y_B(v_i)$ AND $y_T(v_i)$ DENOTE THE BOTTOM AND THE TOP y COORDINATES OF LINE v_i.

ALGEBRAIC AND NUMERICAL ALGORITHMS

ALGEBRAIC AND NUMERICAL ALGORITHMS

* EXPONENTIATION (PROBLEM DESCRIPTION)

GIVEN TWO POSITIVE INTEGERS n AND k, COMPUTE n^k.

* FIRST ATTEMPT FOR A SOLUTION (PSEUDOCODE)

```
POWER(n, k)                          {IT RETURNS THE VALUE OF
    P ← n                             {n^k.}
    for i ← 1 to k-1 do
        P ← n * P
    return P
```

* THIS APPROACH IS INAPPROPRIATE FOR LARGE VALUES OF k.

* IF $k/2$ IS INTEGER, THEN $n^k = (n^{k/2})^2$.

* IF $k/2$ IS NOT INTEGER, THEN $(k-1)/2$ IS AN INTEGER THEN, n^k IS COMPUTED AS $n^k = n(n^{(k-1)/2})^2$.

ALGEBRAIC AND NUMERICAL ALGORITHMS

* SECOND ATTEMPT (PSEUDOCODE)

```
POWER_BY_REPEATED_SQUARING(n, k)
    if k = 1 then
        P ← n
    else
        z ← POWER_BY_REPEATED_SQUARING(n, k div 2)
        if k mod 2 = 0 then
            P ← z * z
        else
            P ← n * z * z
    return P
```

* THE NUMBER OF MULTIPLICATIONS IS O(log k).

* EUCLID'S ALGORITHM (PROBLEM DESCRIPTION)

FIND THE GREATEST COMMON DIVISOR OF TWO GIVEN POSITIVE INTEGERS.

ALGEBRAIC AND NUMERICAL ALGORITHMS

* GCD ALGORITHM (PSEUDOCODE)

```
GCD(m,n)                          {IF RETURNS THE gcd OF}
                                  {m and n}

    a ← max(n,m)
    b ← min(n,m)
    r ← 1
    while  r > 0   do
        r ← a mod b
        a ← b
        b ← r
    gcd ← a
    return gcd
```

* THE EUCLID'S ALGORITHM HAS RUNNING TIME $O(\log(n+m))$.

ALGEBRAIC AND NUMERICAL ALGORITHMS

* MATRIX MULTIPLICATION (PROBLEM DESCRIPTION)

COMPUTE THE PRODUCT $C = A \cdot B$ OF TWO $n \times n$ MATRICES OF REAL NUMBERS.

* THE STRAIGHTFORWARD APPROACH REQUIRES n^3 MULTIPLICATIONS AND $(n-1)n^2$ ADDITIONS. IT COMPUTES THE PRODUCT BY USING THE FOLLOWING RELATION

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}.$$

* WINOGRAD'S ALGORITHM

$$A_i = \sum_{k=1}^{n/2} a_{i,2k-1} \cdot a_{i,2k} \text{ and } B_j = \sum_{k=1}^{n/2} b_{2k-1,j}\, b_{2k,j}.$$

$$c_{ij} = \sum_{k=1}^{n/2} (a_{i,2k-1} + b_{2k,i})(a_{i,2k} + b_{2k-1,j}) - A_i - B_j$$

n : EVEN

MULTIPLICATIONS: $\frac{1}{2}n^3 + n^2$.

ADDITIONS HAVE INCREASED BY ABOUT $1/2\, n^3$.

ALGEBRAIC AND NUMERICAL ALGORITHMS

* STRASSEN'S ALGORITHM IS BASED ON THE DIVIDE-AND CONQUER METHOD (ONE OF ITS MOST FAMOUS APPLICATIONS).

* STEPS

(a) DIVIDE THE INPUT MATRICES A AND B INTO $\frac{n}{2} \times \frac{n}{2}$ SUBMATRICES.

(b) USING $\Theta(n^2)$ SCALAR ADDITIONS AND SUBTRACTION COMPUTE 14 $n/2 \times n/2$ matrices $A_1, B_1, A_2, B_2, \ldots, A_7, B_7$.

(c) RECURSIVELY COMPUTE THE SEVEN MATRIX PRODUCTS $P_i = A_i B_i$ FOR $i = 1, 2, \ldots, 7$.

(d) COMPUTE THE DESIRED SUBMATRICES r, s, t, u OF THE RESULT MATRIX C $\left(C = \begin{pmatrix} r & s \\ t & u \end{pmatrix}\right)$ BY ADDING ANDIOR SUBTRACTING VARIOUS COMBINATIONS OF THE P_i MATRICES, USING ONLY $\Theta(n^2)$ SCALAR ADDITIONS AND SUBTRACTION

ALGEBRAIC AND NUMERICAL ALGORITHMS

* STRASSEN'S ALGORITHM IS DESCRIBED BY THE RECURRENCE EQUATION

$$T(n) = 7\,T(n/2) + \Theta(n^2)$$

$$\Downarrow$$

$$T(n) = \Theta(n^{\log 7}).$$

* WITHOUT THE TRICKS, THE ALGORITHM IS DESCRIBED BY THE FOLLOWING RECURRENCE EQUATION

$$T(n) = 8\,T(n/2) + \Theta(n^2)$$

$$\Downarrow$$

$$T(n) = \Theta(n^3).$$

* IT IS ASSUMED THAT

$$C = \begin{pmatrix} r & s \\ t & u \end{pmatrix} = A \cdot B = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} e & g \\ f & h \end{pmatrix}$$

WHERE

$r = ae + bf, \quad s = ag + bh$

$t = ce + df, \quad u = cg + dh.$

ALGEBRAIC AND NUMERICAL ALGORITHMS

* P_i CAN BE WRITTEN AS:

$$P_i = A_i B_i = (a_{i1}a + a_{i2}b + a_{i3}c + a_{i4}d) \cdot (B_{i1}e + B_{i2}f + B_{i3}g + B_{i4}h),$$

WHERE a_{ij} AND B_{ij} ARE ALL DRAWN FROM THE SET $\{-1, 0, 1\}$.

* CALCULATION OF THE P_i's.

$$r = ae + bf = (a\ b\ c\ d) \begin{pmatrix} +1 & 0 & 0 & 0 \\ 0 & +1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} e \\ f \\ g \\ h \end{pmatrix}$$

$$s = ag + bh = (a\ b\ c\ d) \begin{pmatrix} 0 & 0 & +1 & 0 \\ 0 & 0 & 0 & +1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} e \\ f \\ g \\ h \end{pmatrix}$$

$$t = ce + df = (a\ b\ c\ d) \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ +1 & 0 & 0 & 0 \\ 0 & +1 & 0 & 0 \end{pmatrix} \begin{pmatrix} e \\ f \\ g \\ h \end{pmatrix}$$

$$u = cg + dh = (a\ b\ c\ d) \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & +1 & 0 \\ 0 & 0 & 0 & +1 \end{pmatrix} \begin{pmatrix} e \\ f \\ g \\ h \end{pmatrix}$$

$$s = \underbrace{ag - ah}_{P_1} + \underbrace{ah + bh}_{P_2} = P_1 + P_2 = A_1B_1 + A_2B_2$$

$$t = \underbrace{ce + de}_{P_3} + \underbrace{df - de}_{P_4} = P_3 + P_4 = A_3B_3 + A_4B_4$$

ALGEBRAIC AND NUMERICAL ALGORITHMS

* STRASSEN'S ALGORITHM (CONTINUED)

$P_5 = A_5 B_5$
$= (a+d)(e+h)$

$P_6 = A_6 B_6$
$= (b-d)(f+h)$

$r = P_5 + P_4 - P_2 + P_6$
$= ae + bf$

$P_7 = A_7 B_7$
$= (a-c)(e+g)$

$u = P_5 + P_1 - P_3 - P_7$
$= cg + dh.$

* DRAWBACKS OF STRASSEN'S ALGORITHM

- IF $n < 100$, THEN STRASSEN'S ALGORITHM IS NOT FASTER THAN THE STRAIGHTFORWARD APPROACH.
- STRASSEN'S ALGORITHM IS LESS STABLE THAN THE STRAIGHTFORWARD APPROACH.
- STRASSEN'S ALGORITHM IS COMPLEX AND NOT EASILY PARALLELIZABLE.

ALGEBRAIC AND NUMERICAL ALGORITHMS

* POLYNOMIAL MULTIPLICATION (PROBLEM DESCRIPTION)

COMPUTE THE PRODUCT OF TWO POLYNOMIALS OF DEGREE $n-1$.

* LET $P = \sum_{i=0}^{n-1} p_i x^i$ AND $Q = \sum_{i=0}^{n-1} q_i x^i$ BE TWO POLYNOMIALS OF DEGREE $n-1$. THEN, PQ IS

$$PQ = \{p_{n-1}x^{n-1} + \dots + p_0\}\{q_{n-1}x^{n-1} + \dots + q_0\}$$

$$= p_{n-1}q_{n-1}x^{2n-2} + \dots + \{p_{n-1}q_{i+1} + p_{n-2}q_{i+2} + \dots + p_{i+1}q_{n-1}\}x^{n+i} + \dots + p_0q_0.$$

* THE NUMBER OF ADDITIONS AND MULTIPLICATIONS IS $O(n^2)$.

* IMPROVEMENT? $P = P_1 + P_2 x^{n/2}$, $Q = Q_1 + Q_2 x^{n/2}$.

$$PQ = P_1Q_1 + (P_1Q_2 + P_2Q_1)x^{n/2} + P_2Q_2x^n.$$

* NUMBER OF OPERATIONS $T(n) = 4T(n/2) + O(n)$, $T(1) = 1$

ALGEBRAIC AND NUMERICAL ALGORITHMS

* NEW IMROVEMENT ?

-	x	P_1	P_2
	Q_1	A	B
	Q_2	C	D

- $PQ = A + (B+C)x^{n/2} + Dx^n$

- WE NEED TO COMPUTE A, D, AND B+C.

- $B+C = (P_1+P_2)(Q_1+Q_2) - A - D.$

- THE NEW RECURRENCE RELATION IS

$$T(n) = 3T(n/2) + O(n) \Rightarrow$$

$$T(n) = O(n^{\log 3}) = O(n^{1.59}).$$

- EXAMPLE $P = 1 - x + 2x^2 - x^3$ AND $Q = 2 + x - x^2 + 2x^3$.

$A = (1-x)(2+x) = 2 - x - x^2$

$D = (2-x)(-1+2x) = -2 + 5x - 2x^2$

$(P_1+P_2)(Q_1+Q_2) = (3-2x)(1+3x) = 3 + 7x - 6x^2$

$B+C = 3 + 3x - 3x^2$ $\quad PQ = 2 - x + 2x^2 + 3x^3 - 5x^4 + 5x^5 - 2x^6.$

ALGEBRAIC AND NUMERICAL ALGORITHMS

* THE FAST FOURIER TRANSFORM (FFT) IS ONE OF THE MOST POPULAR AND IMPORTANT ALGORITHMS OF COMPUTER SCIENCE.

* WE WILL CONCENTRATE ON ONLY ONE APPLICATION OF THE FFT -- POLYNOMIAL MULTIPLICATION.

* POLYNOMIAL MULTIPLICATION (PROBLEM DESCRIPTION)

  GIVEN TWO POLYNOMIALS $P(x)$ AND $Q(x)$, COMPUTE THEIR PRODUCT $P(x)Q(x)$.

* A POLYNOMIAL OF DEGREE n IS UNIQUELY DEFINED BY $n+1$ POINTS.

* EXAMPLE

  THE SECOND DEGREE POLYNOMIAL $P(x): x^2+3x+1$ IS DEFINED BY THE POINTS $(1,5)$, $(2,11)$, AND $(3,19)$.

ALGEBRAIC AND NUMERICAL ALGORITHMS

* FFT PROVIDES AN EFFICIENT MECHANISM FOR CHANGING REPRESENTATIONS IN ORDER TO ACHIEVE A VERY GOOD POLYNOMIAL MULTIPLICATION ALGORITHM.

* CONVERTING FROM COEFFICIENTS TO POINTS CAN BE DONE BY POLYNOMIAL EVALUATION. WE CAN COMPUTE THE VALUE OF A POLYNOMIAL $P(x)$, GIVEN BY ITS LIST OF COEFFICIENTS, AT ANY GIVEN POINT BY HORNER'S RULE USING n MULTIPLICATIONS. n IS THE DEGREE OF THE POLYNOMIAL $P(x)$.

* THE EVALUATION OF $P(x)$ AT n ARBITRARY POINTS REQUIRES n^2 MULTIPLICATIONS.

* CONVERTING FROM POINTS TO COEFFICIENTS IS CALLED INTERPOLATION AND REQUIRES $O(n^3)$ OPERATIONS.

ALGEBRAIC AND NUMERICAL ALGORITHMS

* FORWARD FOURIER TRANSFORM

- WE NEED TO EVALUATE TWO $n-1$ DEGREE POLYNOMIALS, EACH AT $2n-1$ POINTS, SO THAT THEIR PRODUCT, WHICH IS $2n-2$ DEGREE POLYNOMIAL, CAN BE INTERPOLATED.

- AN $n-1$ DEGREE POLYNOMIAL CAN BE REPRESENTED AS A $2n-2$ DEGREE POLYNOMIAL.

- THE PROBLEM CAN BE REDUCED TO A PROBLEM OF EVALUATING A POLYNOMIAL $P = \sum_{i=0}^{n-1} a_i x^i$ AT n DISTINCT POINTS $(x_0, \ldots, x_{n-1})$.

- A MATRIX-VECTOR REPRESENTATION

$$\begin{pmatrix} 1 & x_0 & (x_0)^2 & \cdots & (x_0)^{n-1} \\ 1 & x_1 & (x_1)^2 & \cdots & (x_1)^{n-1} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & x_{n-1} & (x_{n-1})^2 & \cdots & (x_{n-1})^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} P(x_0) \\ P(x_1) \\ \vdots \\ P(x_{n-1}) \end{pmatrix}$$

ALGEBRAIC AND NUMERICAL ALGORITHMS

* FORWARD FOURIER TRANSFORM

- AN IMPROVED (COMPUTATIONALLY) MATRIX-VECTOR REPRESENTATION.

$$\begin{pmatrix} 1 & x_0 & (x_0)^2 & \dots & (x_0)^{n-1} \\ 1 & x_1 & (x_1)^2 & \dots & (x_1)^{n-1} \\ \vdots & & & & \\ 1 & x_{n/2-1} & (x_{n/2-1})^2 & \dots & (x_{n/2-1})^{n-1} \\ 1 & -x_0 & (-x_0)^2 & \dots & (-x_0)^{n-1} \\ 1 & -x_1 & (-x_1)^2 & \dots & (-x_1)^{n-1} \\ \vdots & & & & \\ 1 & -x_{n/2-1} & (-x_{n/2-1})^2 & \dots & (-x_{n/2-1})^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} P(x_0) \\ P(x_1) \\ \\ \\ P(-x_{n/2-1}) \end{pmatrix}$$

$$P(x) = E + O = \sum_{i=0}^{n/2-1} a_{2i} x^{2i} + \sum_{i=0}^{n/2-1} a_{2i+1} x^{2i+1},$$

$$E = \sum_{i=0}^{n/2-1} a_{2i} (x^2)^i = P_E(x^2), \qquad O = x \sum_{i=0}^{n/2-1} a_{2i+1} (x^2)^i = P_O(x^2)$$

$$P(x) = P_E(x^2) + x P_O(x^2),$$

$$P(-x) = P_E(x^2) + (-x) P_O(x^2).$$

- THIS IS AN IMPORTANT STEP IN THE DECOMPOSITION OF THE PROBLEM.

- SINCE WE WANT TO BUILD A RECURSIVE SCHEME, WE HAVE TO USE COMPLEX NUMBERS.

ALGEBRAIC AND NUMERICAL ALGORITHMS

* FORWARD FOURIER TRANSFORM

- THE PRIMITIVE nTH ROOT OF UNITY IS USED.

- IT IS DENOTED BY ω.

- $\omega^n = 1$ AND $\omega^j \neq 1$ FOR $0 < j < n$.

- WE WANT TO COMPUTE THE FOLLOWING PRODUCT

$$\underbrace{\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ \cdot & \omega^2 & \omega^{2\cdot 2} & \cdots & \omega^{2(n-1)} \\ \cdot & \cdot & \vdots & \ddots & \vdots \\ \cdot & \cdot & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{(n-1)2} & \cdots & \omega^{(n-1)(n-1)} \end{pmatrix}}_{V(\omega)} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} P(1) \\ P(\omega) \\ P(\omega^2) \\ \vdots \\ P(\omega^{n-1}) \end{pmatrix}$$

- THIS PRODUCT IS CALLED THE FOURIER TRANSFORM OF $(a_0, a_1, \ldots, a_{n-1})$.

- WE CAN OBSERVE THAT FOR ANY j $(0 \leq j < n/2)$, WE HAVE $x_{j+n/2} = \omega^{n/2} x_j = -x_j$.

ALGEBRAIC AND NUMERICAL ALGORITHMS

* FAST FOURIER TRANSFORM (PSEUDOCODE)

```
FAST_FOURIER_TRANSFORM(n, a_0, a_1, ..., a_{n-1}, w, V)
                    { V IS THE OUTPUT ARRAY. }

if n = 1 then
    V[0] ← a_0
else
    FAST_FOURIER_TRANSFORM(n/2, a_0, a_2 ..., a_{n-2}, w^2, U)
    FAST_FOURIER_TRANSFORM(n/2, a_1, a_3, ..., a_{n-1}, w^2, W)
    for j ← 0 to n/2 - 1 do
        V[j] ← U[j] + w^j W[j]
        V[j + n/2] ← U[j] - w^j W[j]
```

* THE INVERSE FOURIER TRANSFORM

$$[V(w)]^{-1} = \frac{1}{n} V\left(\frac{1}{w}\right).$$

NP-COMPLETENESS

NP-COMPLETENESS

* THE RUNNING TIMES OF MOST OF THE ALGORITHMS THAT WE HAVE SEEN WERE BOUNDED BY SOME POLYNOMIAL IN THE SIZE OF THE INPUT.

* SUCH ALGORITHMS ARE CALLED EFFICIENT.

* THE CORRESPONDING PROBLEMS ARE CALLED TRACTABLE.

* AN ALGORITHM IS CALLED EFFICIENT IF ITS RUNNING TIME IS $O(P(n))$, WHERE $P(n)$ IS A POLYNOMIAL IN THE SIZE OF THE INPUT n.

* THE CLASS OF PROBLEMS THAT CAN BE SOLVED BY EFFICIENT ALGORITHMS IS DENOTED BY P.

* THERE ARE MANY PROBLEMS FOR WHICH NO POLYNOMIAL TIME ALGORITHM IS KNOWN.

* THERE IS A CLASS OF PROBLEMS, CALLED THE NP-COMPLETE PROBLEMS.

NP-COMPLETENESS

* NO POLYNOMIAL ALGORITHM HAS YET BEEN DISCOVERED FOR AN NP-COMPLETE PROBLEM.

* THE SO-CALLED $P \neq NP$ QUESTION HAS BEEN ONE OF THE MOST CHALLENGING PROBLEMS IN THEORETICAL COMPUTER SCIENCE.

* MANY RESEARCHERS BELIEVE THAT THE NP-COMPLETE PROBLEMS ARE INTRACTABLE.

* IF ANY SINGLE NP-COMPLETE PROBLEM CAN BE SOLVED IN POLYNOMIAL TIME, THEN EVERY NP-COMPLETE PROBLEM HAS A POLYNOMIAL-TIME ALGORITHM.

* APPROXIMATION ALGORITHMS ARE WIDELY USED.

* THOUSANDS OF NP-COMPLETE PROBLEMS

- HAMILTONIAL CYCLE
- INDEPENDENT SET
- PARTITION
- TRAVELING SALESMAN
- KNAPSACK
- BIN PACKING

.....