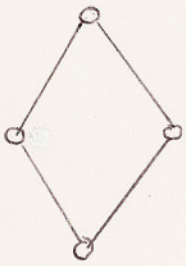
Graph G (V, E)

V – vertices(nodes) $|V| = n$

E – edges $|E| = m$

Pseudo graph

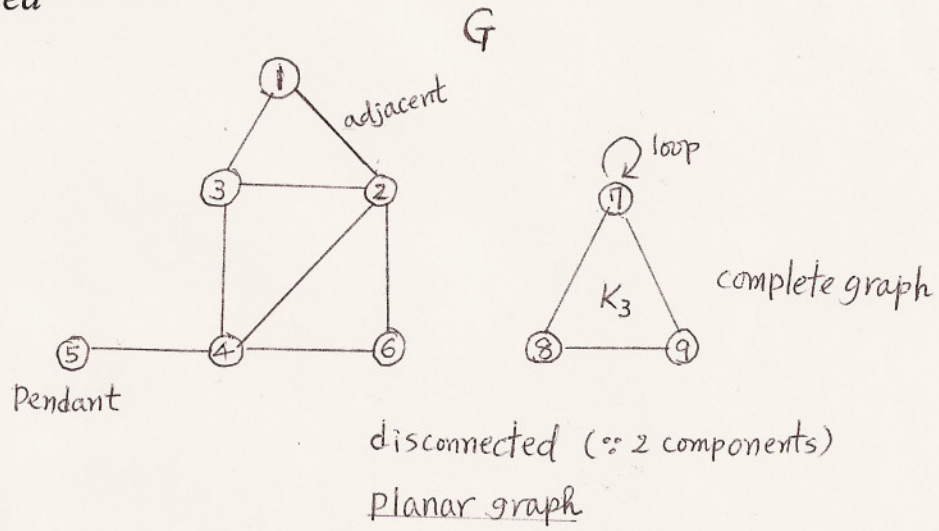
Multigraph



Simple graph

Directed

Undirected



disconnected (∵ 2 components)

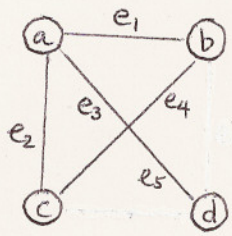
planar graph

Distance: dist (2,5) = 2 length of shortest path

Diameter(G) = maximum distance between any two vertices

Degree: d(1) = 2, d(2) = 4, d(3) = 3

Graph Representation



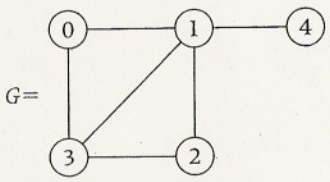
(1) Adjacency Matrix

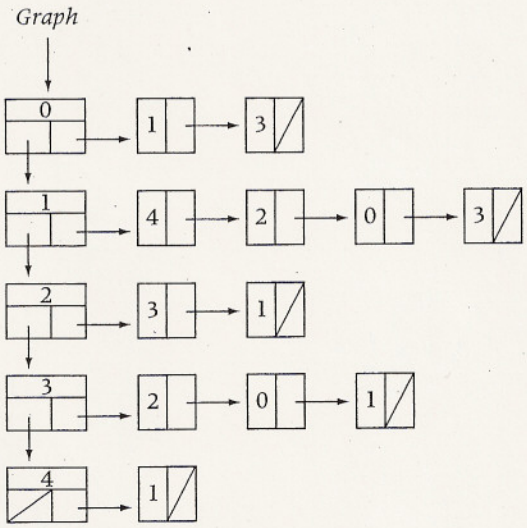
	a	b	c	d
a	0	1	1	1
b	1	0	1	0
c	1	1	0	0
d	1	0	0	0

(2) Incidence Matrix

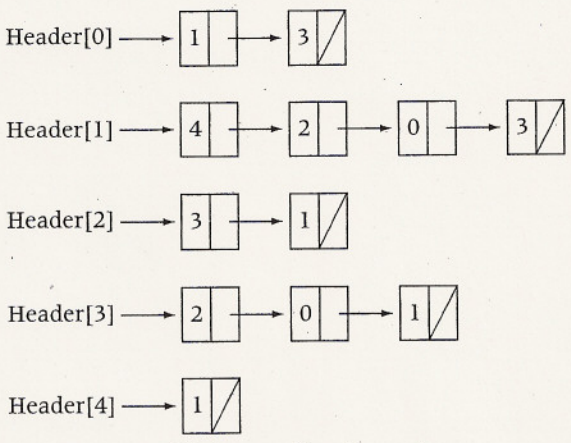
	e_1	e_2	e_3	e_4	e_5
a	1	1	1	0	1
b	1	0	0	1	0
c	0	1	0	1	0
d	0	0	1	0	1

(3) Adjacency List





(a) Linked list of header nodes

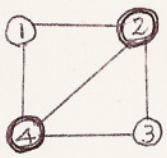


(b) Array of header nodes

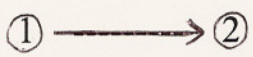
Theorem 1. $\sum_{v \in V} d(v) = 2m$ (* even number *)

Proof. each edge – 2 vertices
So each edge is counted twice in $\sum_{v \in V} d(v)$

Theorem 2. Number of vertices of odd degree is even.



Proof



predecessor successor

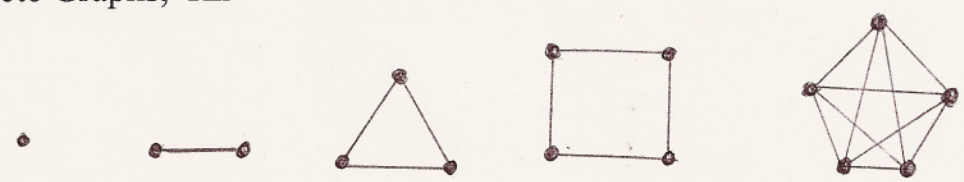
adjacent

path: 1—2—4—3 (simple path)

circuit: 1—2—3—4—1 (simple circuit = cycle)

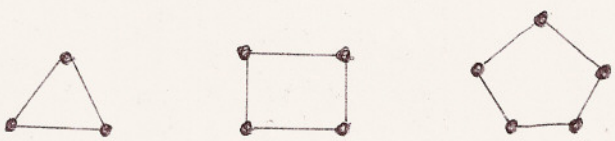
Hamiltonian cycle ≡ A cycle that contains every vertex in the graph

Complete Graphs, Kn



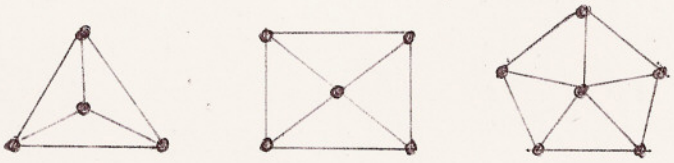
K_1 K_2 K_3 K_4 K_5

Cycles, C_n, $n > 3$



C_3 C_4 C_5

Wheels



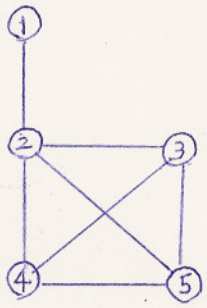
W W W

Clique

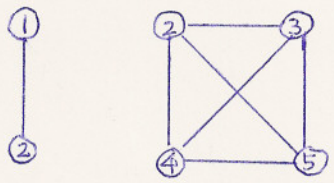
- a maximal complete subgraph of a graph
- not contained in any larger subset having the same property

clique of size p = complete subgraph on p vertices

clique number, $\omega(G)$ = largest clique size (* NPC *)



G



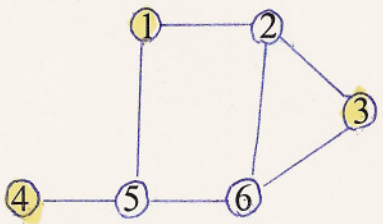
The cliques of G

Note. Turan's Theorem

$$\omega(G) \geq \lceil n^2 / (n^2 - 2m) \rceil$$

Independent Set (of vertices)

-no two vertices in the set are adjacent



U: { 1 3 4 } { 1 4 6 } { 2 5 } { 3 5 }

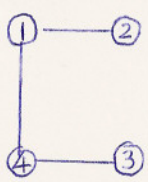
maximal independent set

Independent number, $\beta(G)$ - largest set size

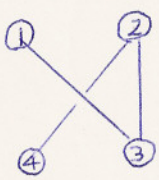
$\bar{G}$: complement of G

$$\beta(G) = \omega(\bar{G})$$

$$m + \bar{m} = \binom{n}{2} = \frac{n(n-1)}{2}$$



G



$\bar{G}$

Tree $\equiv$ connected undirected graph without cycles

Note. $m = n - 1$ (why?)

Rooted-tree

Forest $\equiv$ set of trees

Subgraph H of G: $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$

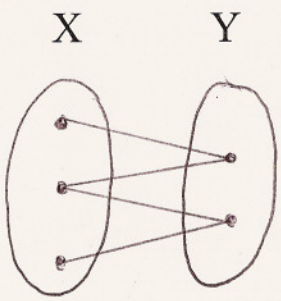
UNION of two graphs

EX

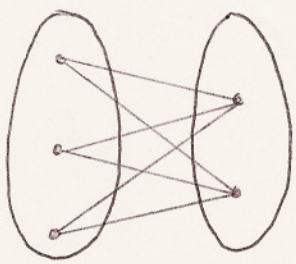
Subtree

Spanning tree $\equiv$ subgraph which is tree and contains every vertex

Bipartite

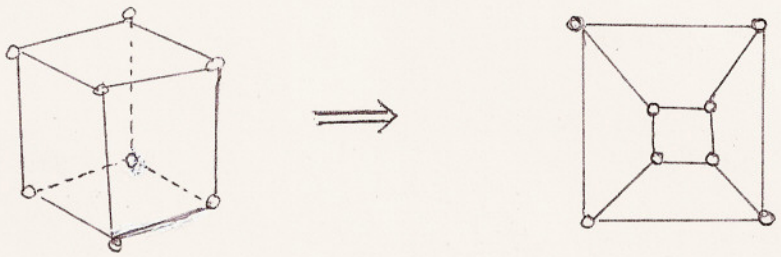


Complete bipartite

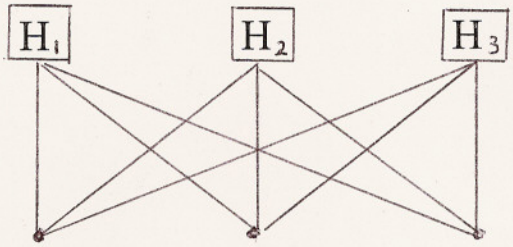


$K_{3,2}$

Planar

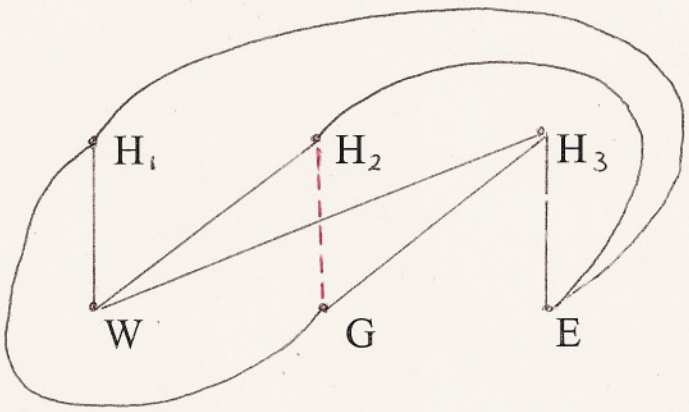


Three Houses and Three Utilities Problem



Gas Water Electricity

Is it possible to join three houses and utilities so that none of The connections cross? -- Planar graph



$K_{3,3}$

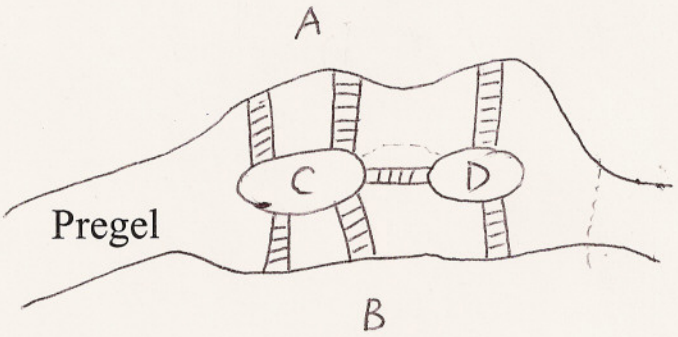
Kuratowski's Theorem

G is non-planar iff it contains a subgraph that is (isomorphic to) a subdivision of K_5 or $K_{3,3}$

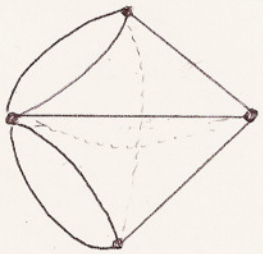
Graph Theory

Königsberg Bridge Problem (1736)

(East Prussia)



dual



Can you draw the above figure w/o lifting the pen from the paper and w/o retracing a line?

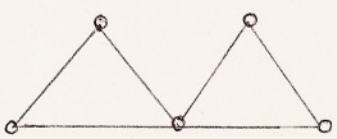
Leonhard Euler (1707-1783)

Euler (tour) circuit ≡ a simple circuit containing every edge of G

Euler path ≡ a simple path containing every edge of G

Chinese Postman Problem

Euler tour



No Euler tour

Four Color Problem

- four colors are enough to color counties in a map

1856 conjectured by Guthrie
1976 proved by Appel and Haken (University of Illinois graduate students) by computer

Note chromatic number ≡ minimum number of colors to properly color the vertices of G.

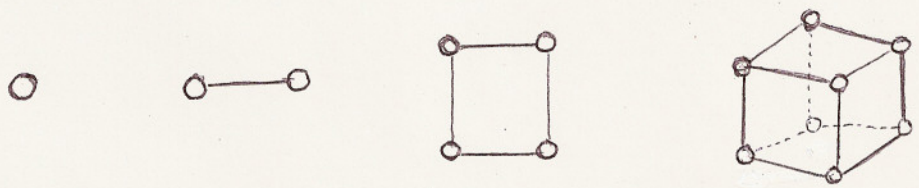
Interconnection Networks

criteria: small diameter
small maximum node degree

Example.

	complete graph	linear
diameter	1	n-1
max. deg	n	2

Hypercube – compromise



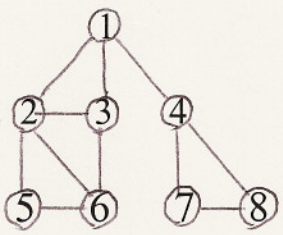
H_0 H_1 H_2 H_3

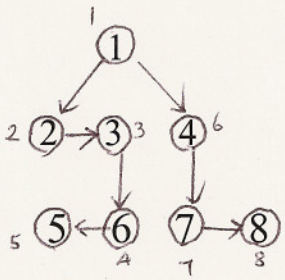
Depth-first Search (undirected)

```
procedure search (G)     // recursive //
begin
      procedure dfs(v)
        VISITED[v] := true
        for each node w adjacent to v do
          if not VISITED[w] then
             call dfs(w)
          endif
        endfor

    // main //
      for each v in V do
        VISITED[v]:= false
      endfor
      call dfs(s)   // s is the root
end
```

Ex.





DFS spanning tree

```
dfs(1)
   dfs(2)
     dfs(3)
        dfs(6)
          dfs(5)
   dfs(4)
     dfs(7)
      dfs(8)
```

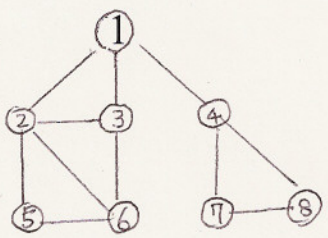
T(n,m) = O(n + m)

Breadth-first Search

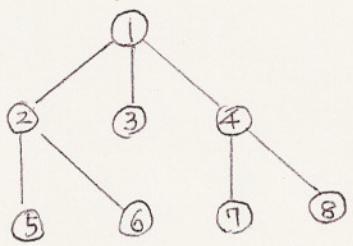
```
procedure search(G)        // non-recursive //
begin
      procedure bfs(v)
        Q:= ∅
        VISITED[v]:= true
        Q ← v    // insert v into Q //
         while Q not empty do
           delete w from Q
           for each w adjacent to v do
             if not VISITED[w] then
             begin
                  VISITED[w]:= true
                  Q ← Q ∪ w
             end
           endfor
         endwhile

     // main //
        for each v do VISITED[v]:= false
        call bfs(s)    // s is the root
  end {search}
```

Example.



Node visited	Q
1	2, 3, 4
2	3, 4, 5, 6
3	4, 5, 6
4	5, 6, 7, 8
5	6, 7, 8
6	7, 8
7	8
8	–

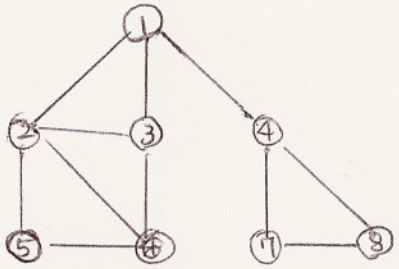


BFS tree

$T(n,m) = O(n+m)$

Note shortest path tree when each edge length is 1.

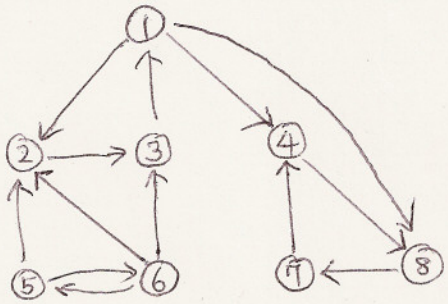
Articulation Point



① and ④

Biconnected ≡ no articulation point

Strongly Connected Components



{1, 2, 3}

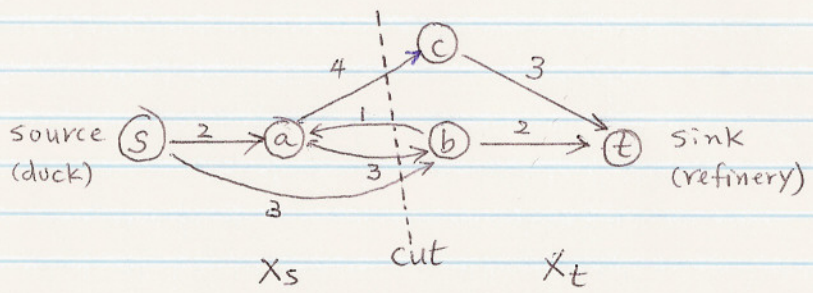
{4, 7, 8}

But {1,2,3,4,7,8} is not.
Because no ④→①

Strongly connected ≡ ∃ a path from u to v and v to u

Maximum Flow Problem

Find a way to send maximum-flow from source to sink in a capacitated graph.



Flow

① $0 \le f_{(i,j)} \le C_{(i,j)}$ for every edge (i,j) in G.

② $\sum_i f_{(i,j)} = \sum_i f_{(j,i)}$ for intermediate nodes,

$\sum_i f(s,i) - \sum_i f(i,s) = F$ (flow value) for s

$\sum_i f(t,i) - \sum_i f(i,t) = -F$ for t

Def cut ≡ set of edges that separate s and t.

$X_s \longrightarrow X_t$: capacity of the cut.

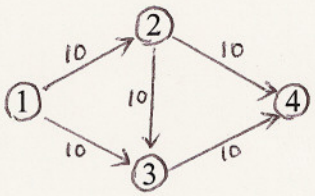
Note It is impossible to send more flow from s to t than the capacity of the cut (∵ every flow must accross the cut)
Therefore, max flow ≤ min. cut.

Algorithms

- Ford and Fulkerson (1956) : Labeling Algorithm
- Edmonds and Karp (1969) : $O(nm^2)$
- Dinic (1970) : $O(n^2m)$
- Karzanov (1974) : $O(n^3)$
- MKM (or three Indians) : $O(n^3)$

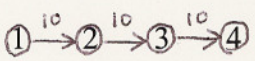
} Layered network approach.

Max-Flow

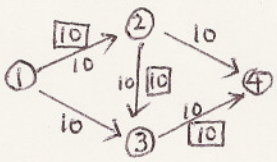


[Wrong Method]

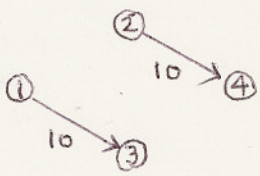
Augmenting-Path



$f_{12} = f_{23} = f_{34} = 10$

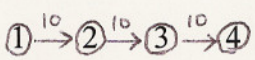


Modified Network

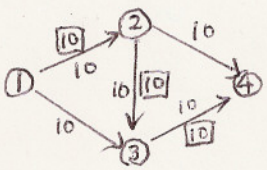


[Ford-Fulkerson]

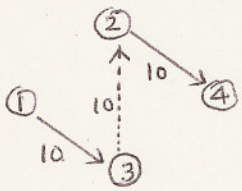
Augmenting-Path



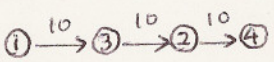
$f_{12} = f_{23} = f_{34} = 10$



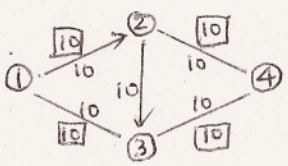
Modified Network



Augmenting path

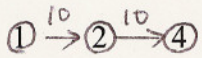


$f_{13} = f_{32} = f_{24} = 10$



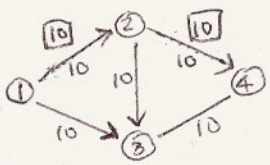
[Edmonds-Karp]

Augmenting-Path

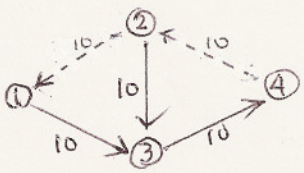


(Shortest Path)

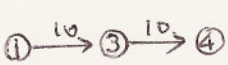
$f_{12} = f_{24} = 10$



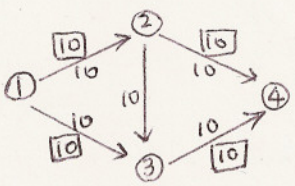
Modified Network



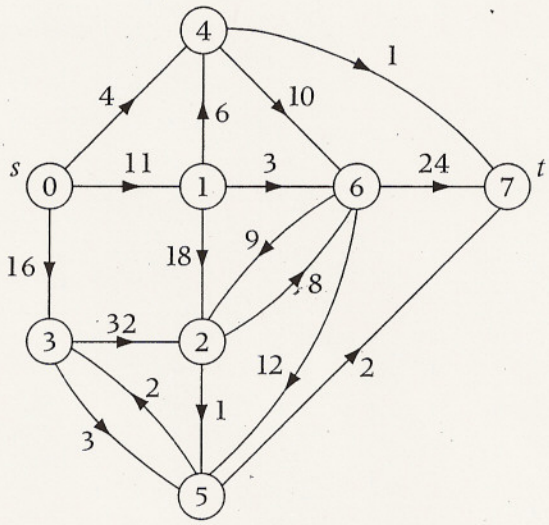
Augmenting path



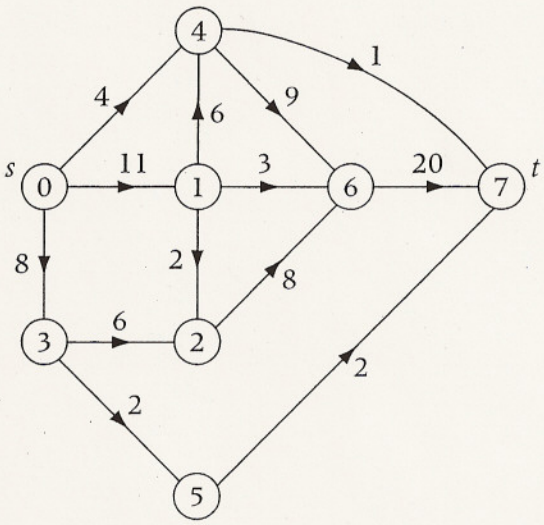
$f_{13} = f_{34} = 10$



MAX FLOW



A capacitated network N

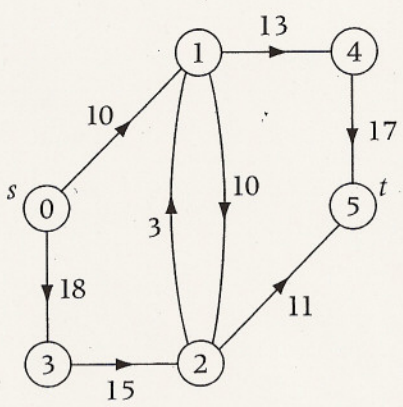


A flow f in N

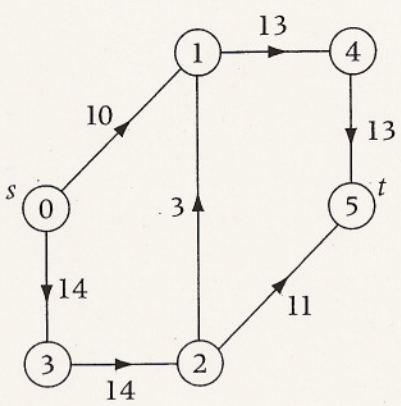
FIGURE 14.5
A sample capacitated network N and a flow f having value $val(f) = 23$.

FIGURE 14.7

A maximum flow f^* having value 24 in the capacitated network N of Figure 14.5.



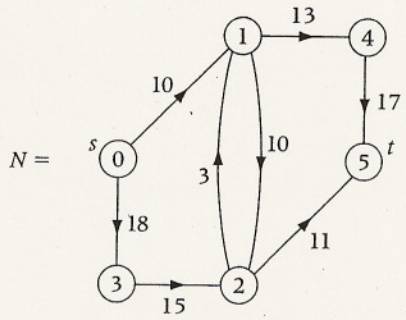
A capacitated network N

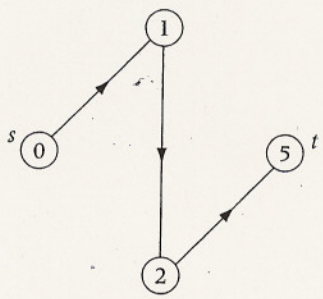


A maximum flow f^* on N

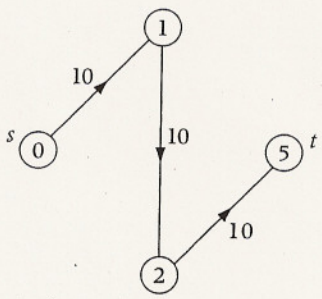
FIGURE 14.6

Action of naive attempt to find maximum flow for a sample capacitated network N, yielding flow $f = 10\chi_{P_1} + \chi_{P_2} + 3\chi_{P_3}$ having value 14. This flow is suboptimal because a maximum flow f^* shown in Figure 14.7 has value 24.

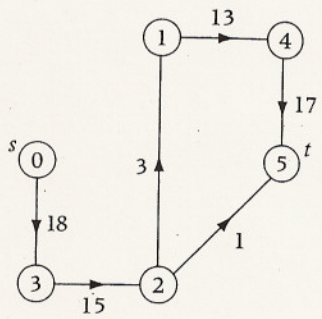




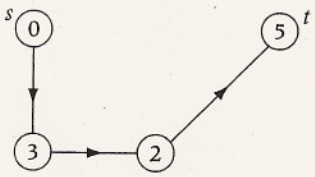
path P_1, $\lambda_1 = 10$



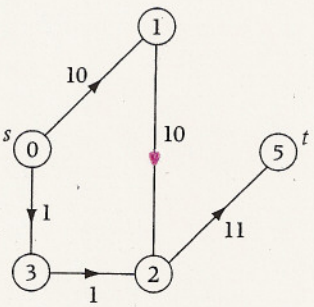
$f = 10\chi_{P_1}$



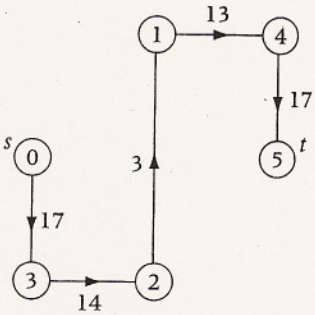
reduced N



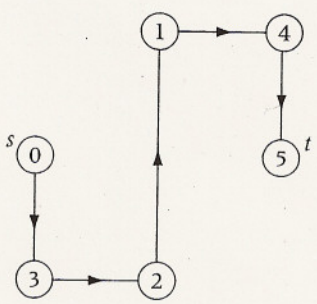
path P_2, $\lambda_2 = 1$

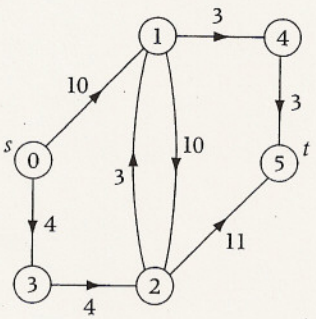


$f = 10\chi_{P_1} + \chi_{P_2}$



reduced N





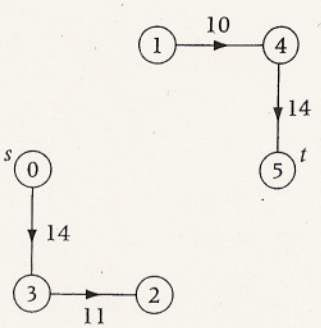


FIGURE 14.8

The f-derived network N_f for a sample network N and flow f.

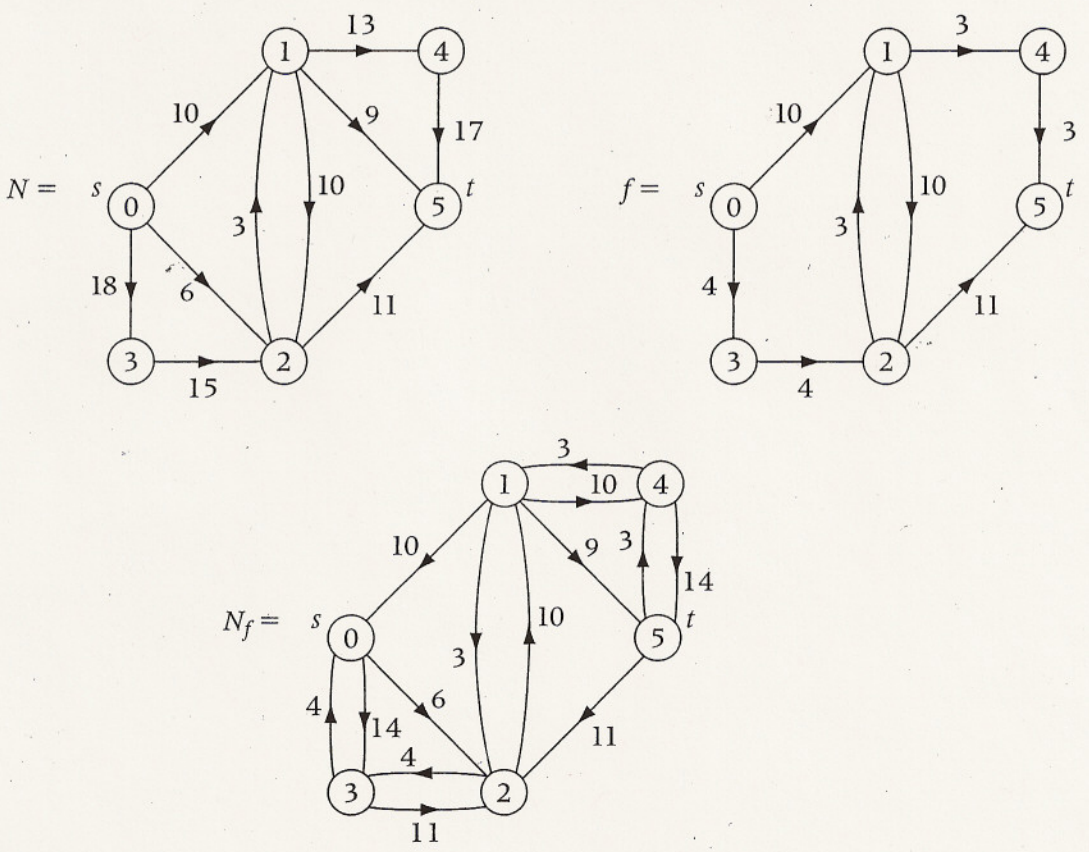


FIGURE 14.9

Action of the Edmonds-Karp algorithm for a sample capacitated network N.

Original flow network N with capacities c, and initial flow $f \equiv 0$:

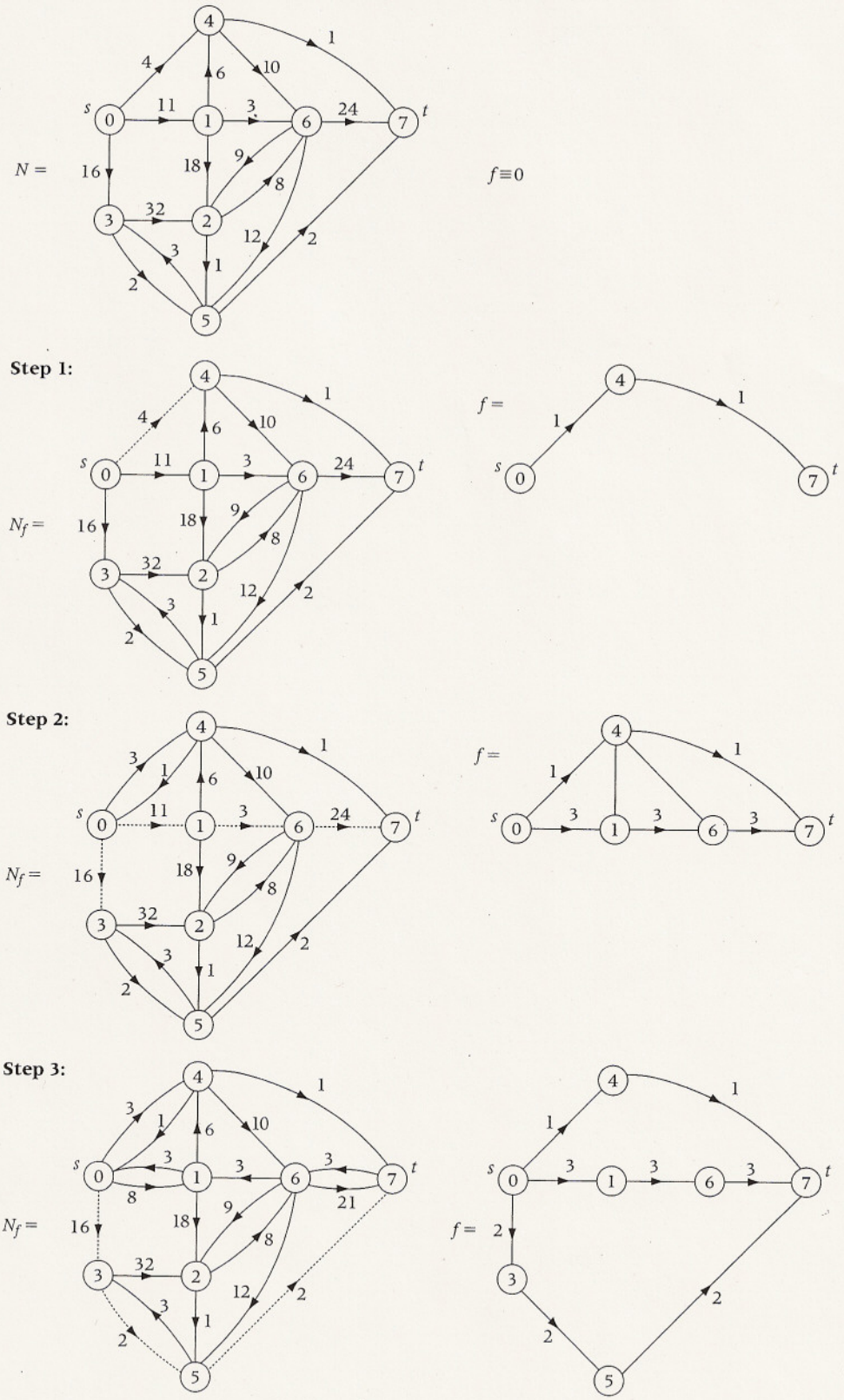
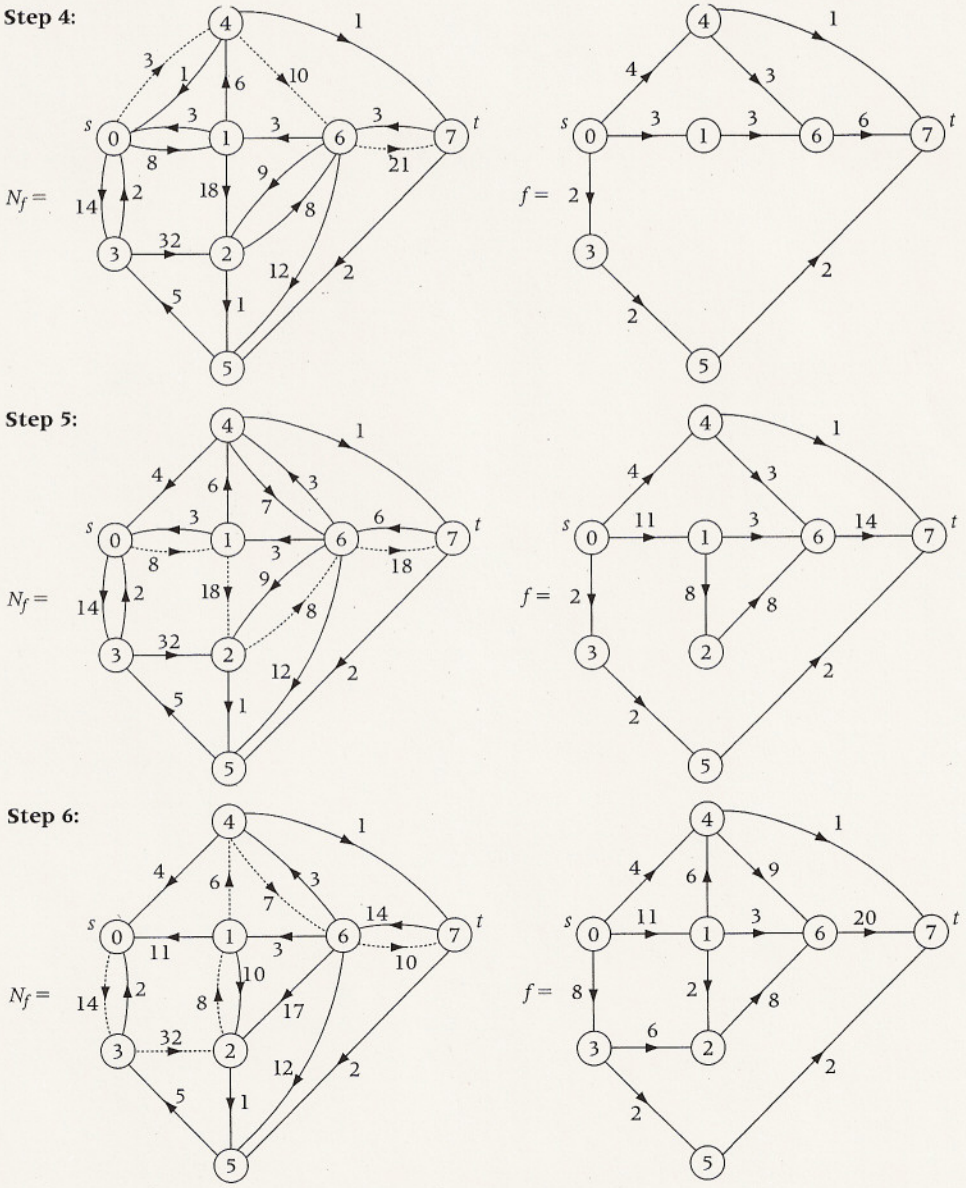
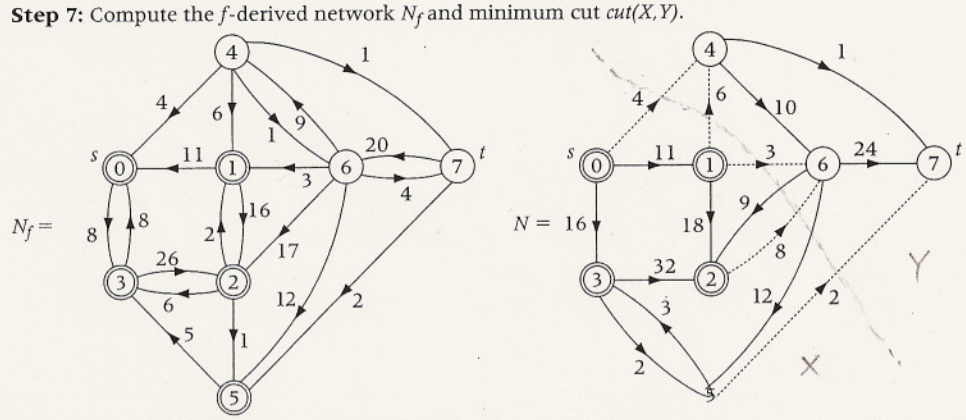


FIGURE 14.9
Continued



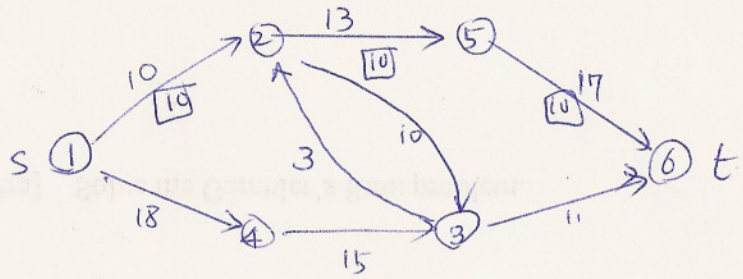
There are no more augmenting semipaths. The final flow f has value 23.

Step 7: Compute the f-derived network N_f and minimum cut $cut(X,Y)$.

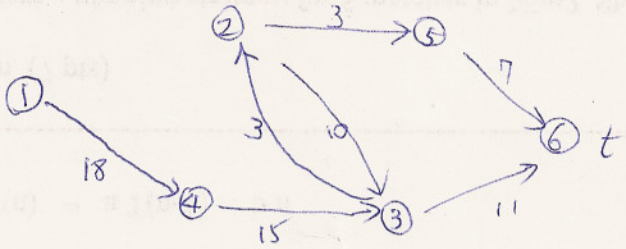


The set $X = \{1,2,3,4,6\}$ of vertices that are accessible in N_f from the source s (marked with ◯) and the set $Y = \{5,7,8\}$ of vertices that are not accessible from s determine a cut $\Gamma = cut(X,Y)$ of capacity $c(X,Y) = 4 + 6 + 3 + 8 + 2 = 23$. Hence, we have $val(f) = 23 = cap(\Gamma)$, so that f is a maximum flow Γ and is a minimum cut.

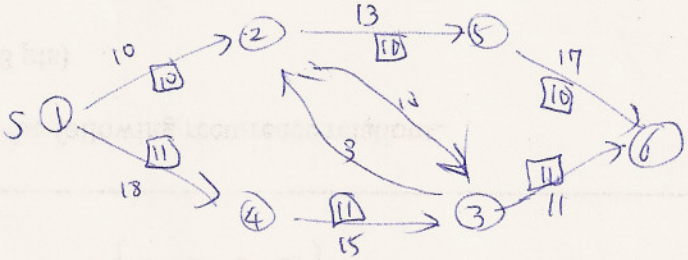
Ford-Fulkerson

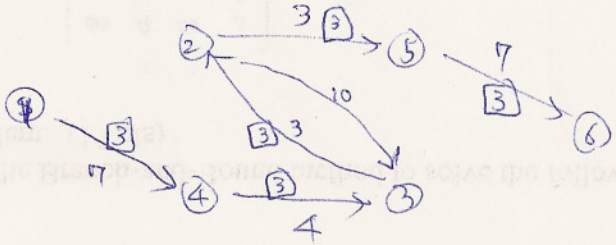


path: ① 10 → ② 13 → ⑤ 17 → ⑥ : flow = 10

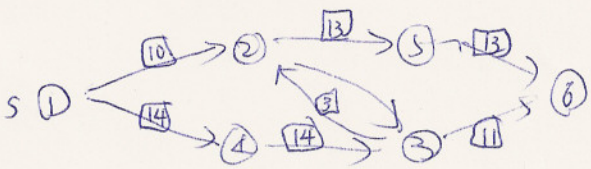


path: ① 18 → ④ 15 → ③ 11 → ⑥ : flow = 11





path: ① 7 → ④ 4 → ③ 3 → ② 3 → ⑤ 7 → ⑥ : flow = 3

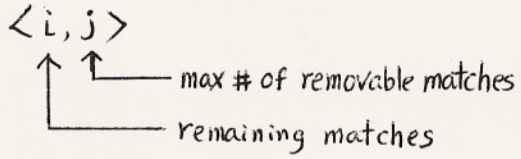


: max flow = 24

Graph and Games

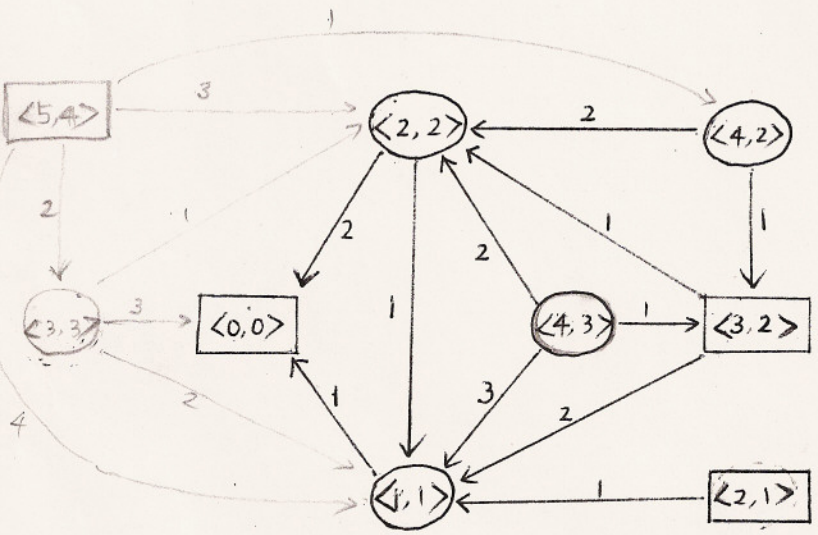
Nim (Marienbad Game)

"Nimm" -take



- win
- lose
- draw

1st player <n, 1..(n-1)>
next player : at least 1, at most twice the # of matches the opponent just took
The player who removes the last match wins.
No draws.



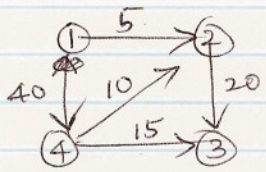
○ : win
□ : lose

Are there winning strategies with 5, 6, 7, or 8 matches?

2 : No
3 : No
4 : Yes
5 : No
6 :

Chapter 14

Network Representation



$|V| = n$

$|E| = m$

(1) Weight matrix

	1	2	3	4
1	0	5	0	40
2	0	0	20	0
3	0	0	0	0
4	0	10	15	0

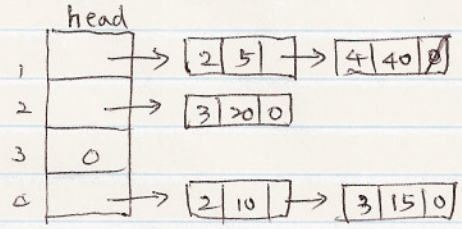
$S(n) = n^2$

(2) Edge List (Triplet)

FROM	TO	WT
1	2	5
1	4	40
2	3	20
4	2	10
4	3	15

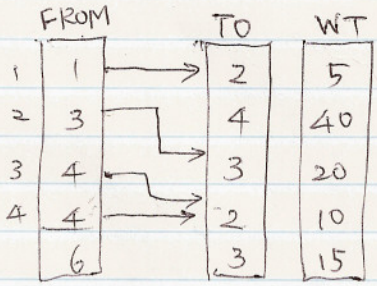
$S(n) = 3m$

(3) Linked list



$S(n) = n + 3m$

(4) Forward star



Note

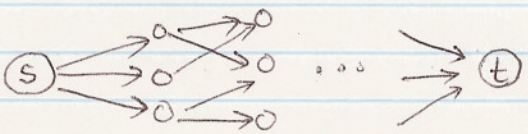
for e := FROM[u] to FROM[u+1] - 1 do
    V := TO[e]

$S(n) = (n+1) + 2m$

Dinic's algorithm (Dinitz's)

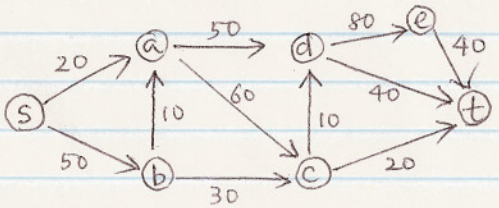
Idea: Push material from the source s progressively through to the sink t.
Then look at "usable" edges and "unsaturated edges" in the network and push some more.
Continue pushing until no more material can be pushed through.

Layered Network, $G_L = (V_L, E_L)$

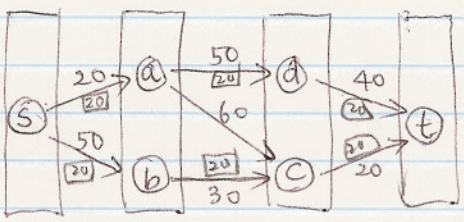


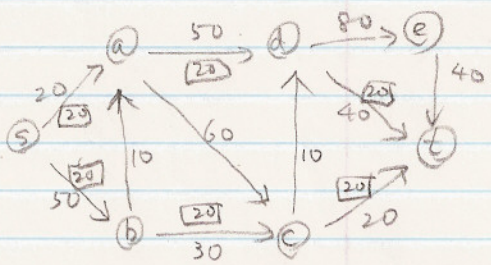
Layer V_1 V_2 V_3 ... V_ℓ

EXAMPLE



[First Layered Network]



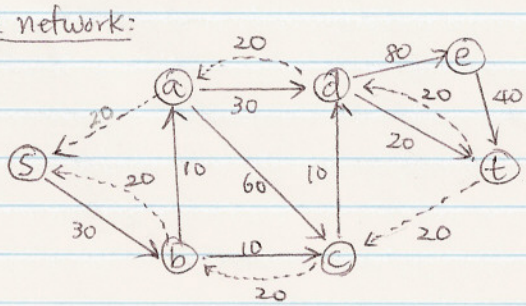


Saturating flow:

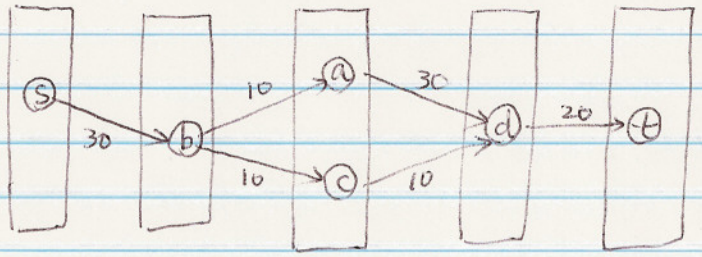
Suppose we get $f_{sa} = f_{ad} = f_{dt} = 20$

$f_{sb} = f_{bc} = f_{ct} = 20$

modified network:



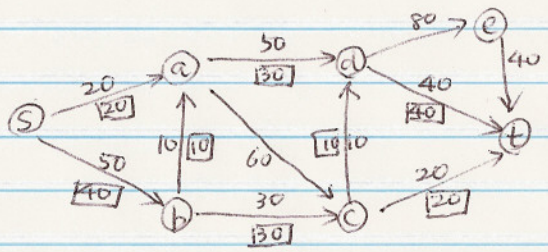
[Second Layered Network]



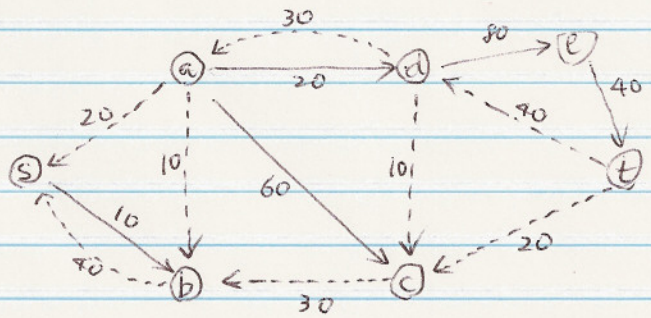
Saturating flow:

$f_{sb} = f_{dt} = 20$

$f_{ba} = f_{ad} = f_{bc} = f_{cd} = 10$



Modified Network



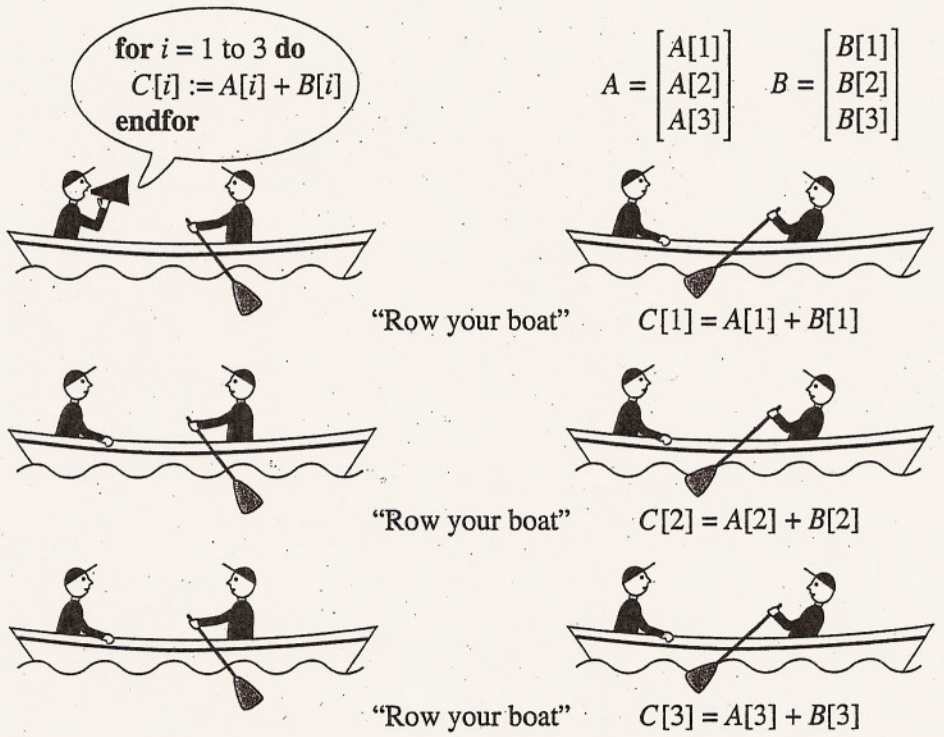
No s-t path.

Max Flow = 60

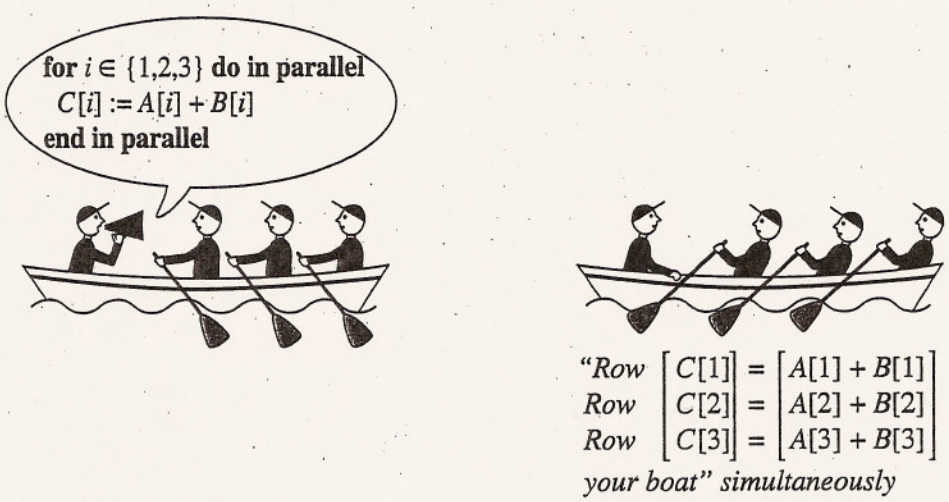
Max-Flow Algorithm.

```
for all (u,v) ∈ E do
    f(u,v) ← 0 ;
call LAYER  // obtain layered network //                    O(n²)
while stpath = true do                                      O(n)
    call SATURATE  // find saturating flow //               O(n²)
    for all (i,j) ∈ E do
        f(i,j) ← f(i,j) + f_L(i,j) ;   (f_L: layered)
    call LAYER
endwhile
```

Note. After each iteration, layer number is increased at least one.
Therefore, Time-complexity $= O(n^3)$



(a)



"Row, row, row your boat, gently down the stream ..."

(b)

Figure 5.1 (a) (Column) vector addition on a serial machine, one row after another; (b) (column) vector addition on a parallel machine, all rows done simultaneously

Classes of computer systems (Taxonomy)

[1] Flynn's classification (1966)

instruction stream – sequence of instructions that are executed in a processing unit
data stream – sequence of operands that are manipulated in a processor.

SISD – sequential (serial) computer.

MISD

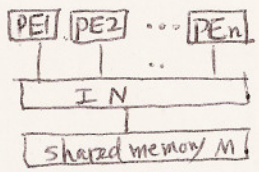
SIMD – array processor. hardware synchronization

Shared-memory SIMD { EREW, CREW, ERCW, CRCW

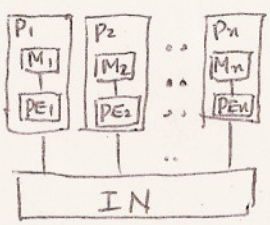
Interconnection-Network SIMD – linear, 2-D, Tree, perfect shuffle, Cube, ...

MIMD – multiprocessor. most general, software synchronization

Shared-memory multiprocessor (tightly-coupled)
shared memory through central switching mechanism



Distributed-memory multicomputer (loosely-coupled)
Shared memory is formed by combining the local memories.
message passing

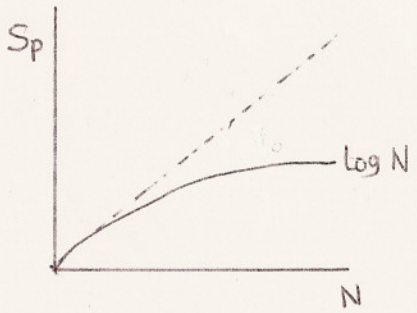


Note: Pipelined vector processor does not fit well in this scheme.
Hockney Jesshope – SIMD
Hwang and Briggs – SISD

Arguments against parallel computing

- **Minsky's conjecture**

  Speedup $\propto$ log N



- **Amdahl's law (1967)**

  The maximum speedup of a parallel processor is limited by the fraction of serial instructions it has to execute.

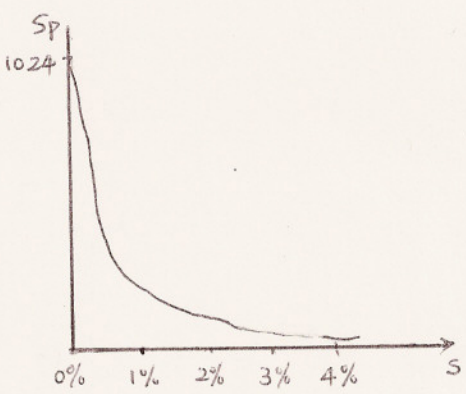
$$\text{Speedup} = \frac{S + P}{(S + \frac{P}{N})} = \frac{1}{(S + \frac{P}{N})}$$

Ex $S = \frac{1}{10}$ $N = 100$ $S_p \fallingdotseq 9.17$

$S = \frac{1}{100}$ $N = 100$ $S_p \fallingdotseq 50.25$

$S = \frac{1}{100}$ $N = 1000$ $S_p \fallingdotseq 91$

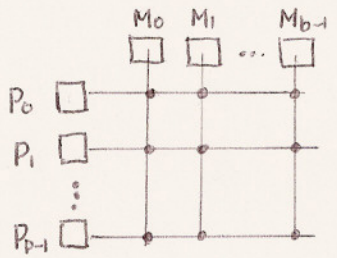
$S = \frac{2}{3}$ $N = 10$ $S_p \fallingdotseq 1.42$



Processor Interconnection

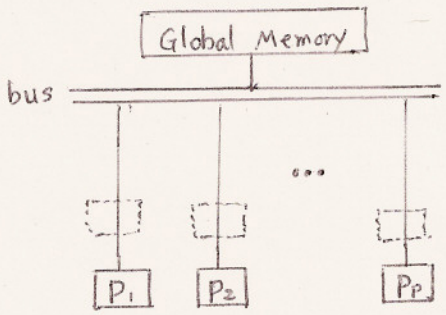
- **Shared Memory Architecture**
  - **dynamic interconnection networks**

. Crossbar Switching Network



CMU C.mmp
Cray Y/MP
Fujitsu VPP500

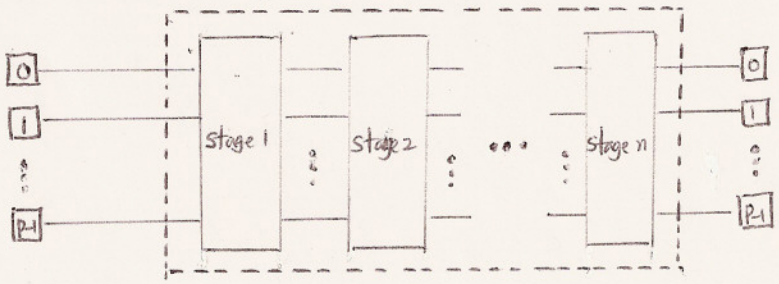
. Bus-based

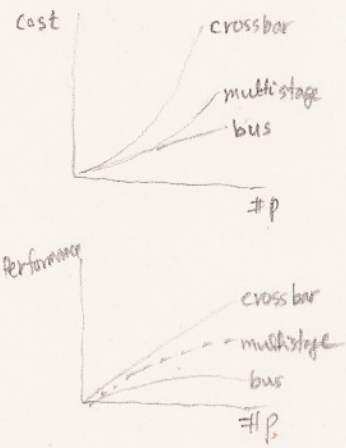


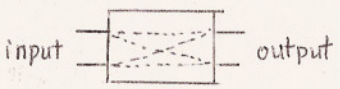
Sequent Symmetry

Note. Performance Vs. cost - tradeoff

. Multistage Interconnection Network







Switching box

control

Butterfly

BBN Butterfly

$B(x) = (b_1, b_{m-1}, \ldots, b_2, b_m)$

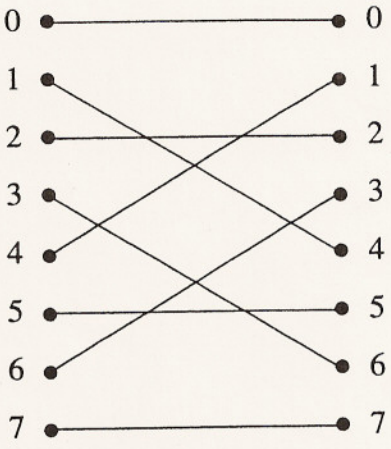
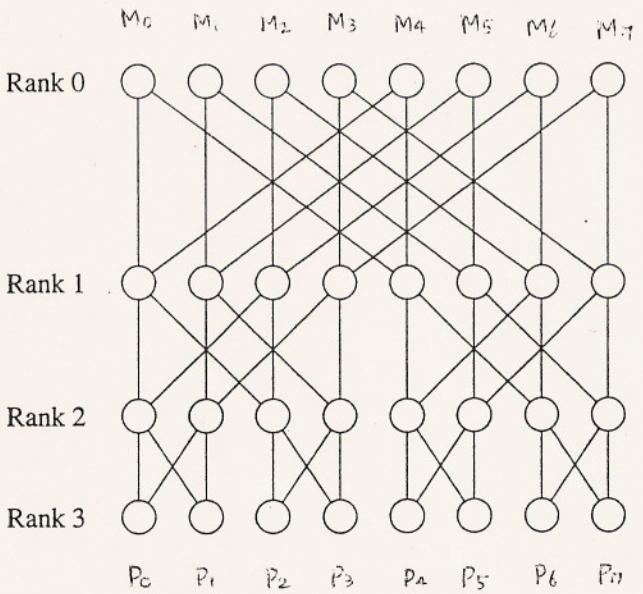
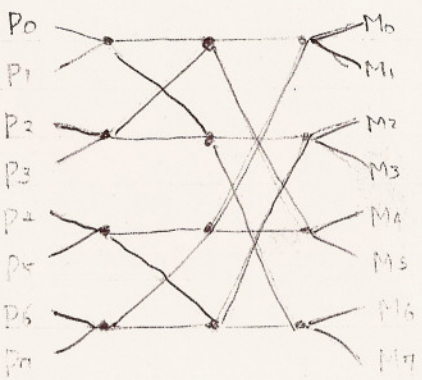


Figure 4.9 Butterfly interconnection.



Butterfly network with 32 nodes.

Another way to represent Butterfly network



Note for K=3, Butterfly = Bit-reversal

Ex Draw butterfly and Bit-reversal for K=4.

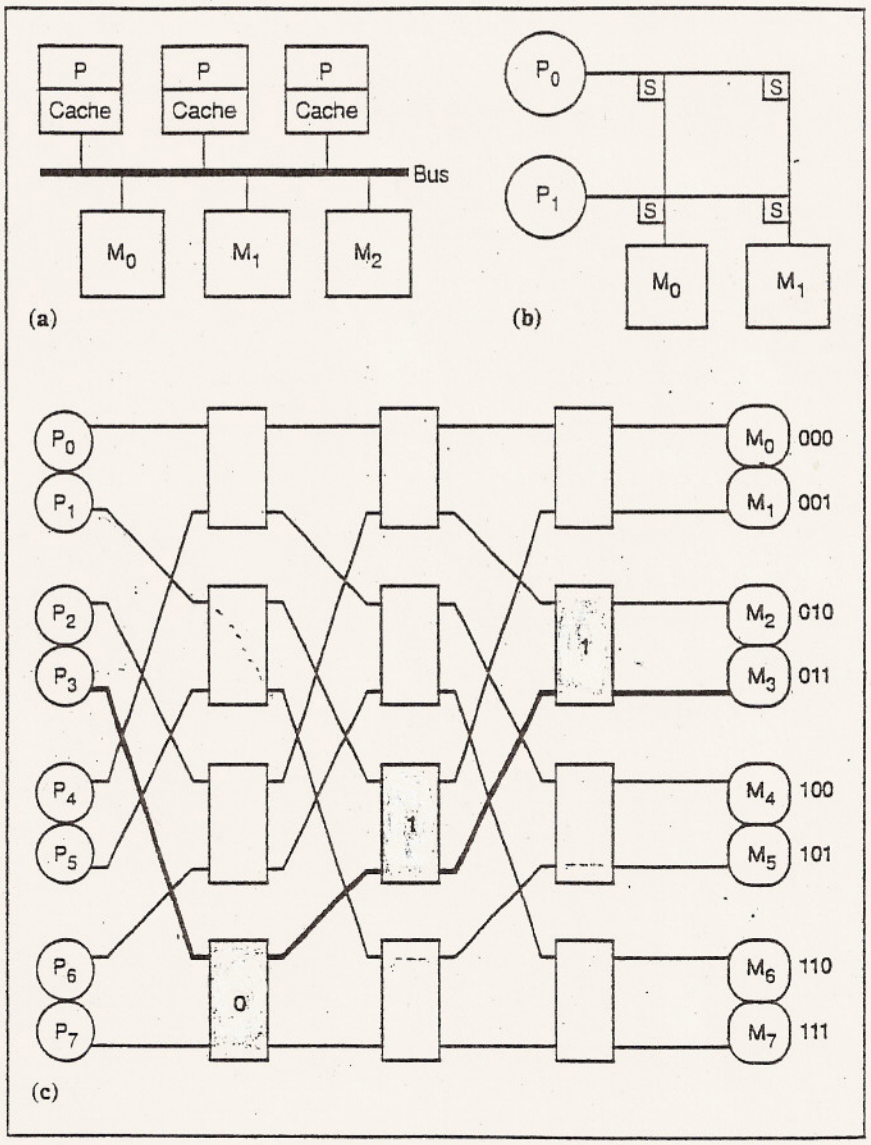
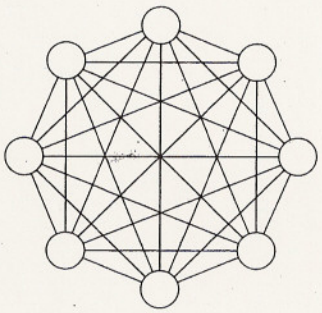


Figure 12. MIMD shared-memory interconnection schemes: (a) bus interconnection; (b) 2×2 crossbar; (c) 8×8 omega MIN routing a P_3 request to M_5.

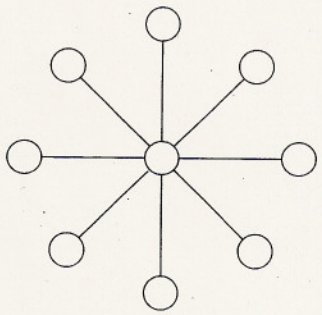
Static Interconnection Network

Completely-connected netwok



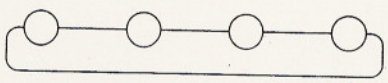
$\frac{n(n-1)}{2}$

Star-connected

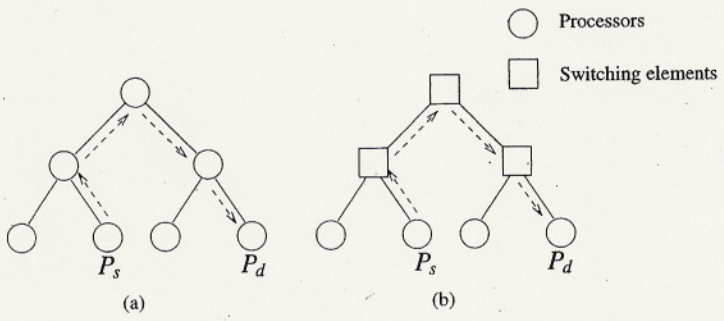


Linear (1-D)

Ring



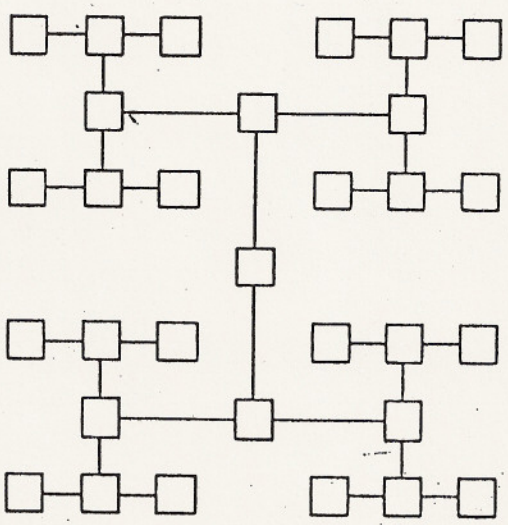
Tree Network

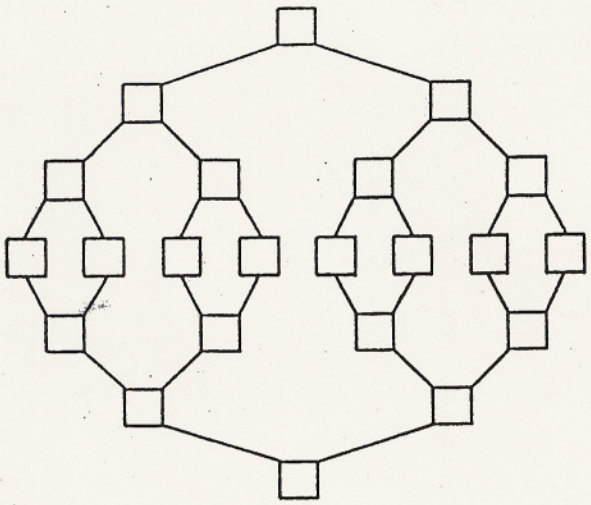


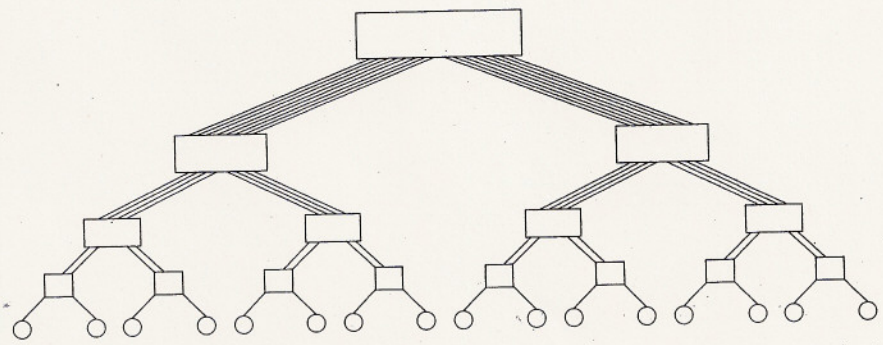
Processor communication : SEND / RECEIVE

Summation in log N steps

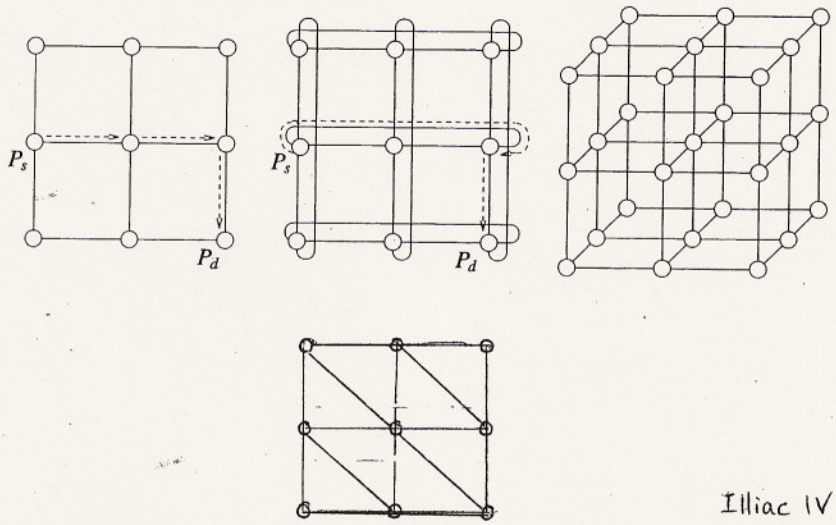
Layout





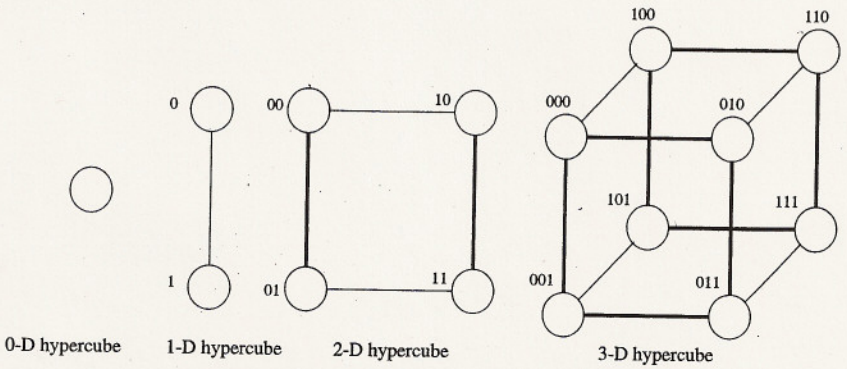


• 2-D Mesh

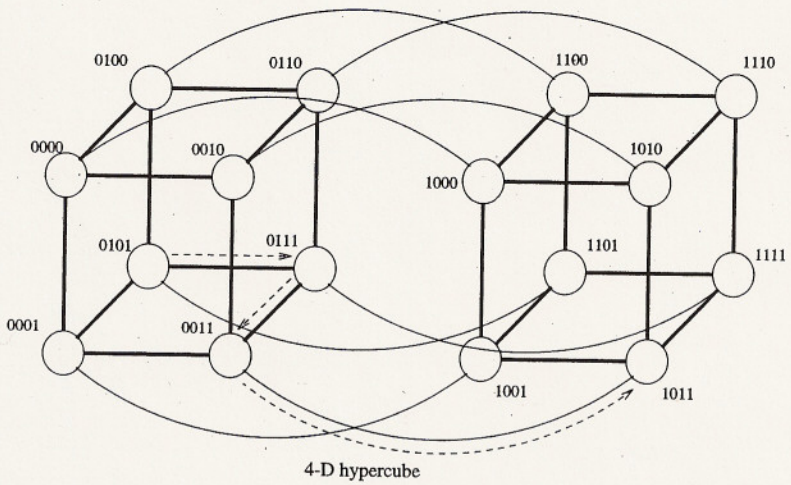


Illiac IV

• Hypercube



0-D hypercube 1-D hypercube 2-D hypercube 3-D hypercube



4-D hypercube

log N interconnections/processor

CM2: 64K
NCUBE2: 1K

Parallel Program Constructs

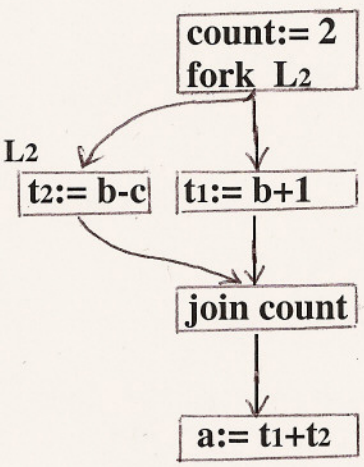
Thread – execution stream
Task – a procedure which can be executed with other procedures in parallel

[1] fork/join

Conway (1963)
Most primitive, flexible but not structured
PL/I, Unix

```
Fork L1      // new concurrent execution
Join  m,g    // m:= m -1; if (m = 0) then goto g
```

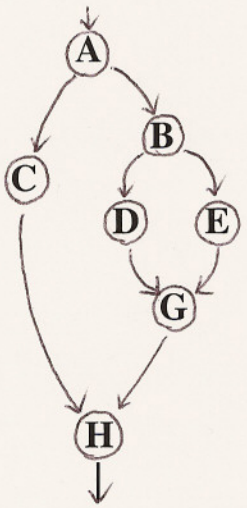
Ex. a:= (b + 1) * (b – c)



// count:= count – 1; if count > 0 wait
// if count = 0 then continue execution
// atomic operation

[2] parbegin/parend (cobegin/coend)

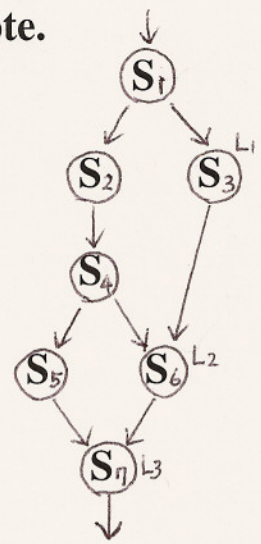
Dijkstra (1965)
High-level language construct
Degree of parallelism is static
Algol-68, CSP



precedence graph

```
A
parbegin
   C
   begin
      B
      parbegin
         D
         E
      parend
      G
   end
parend
H
```

Note.



fork/join

```
      S1
      count1 = 2
      fork L1
      S2
      S4
      count2 = 2
      fork L2
      S5
      goto L3
L1:   S3
L2:   join count1
      S6
L3:   join count2
      S7
```

Parbegin/parend

?

(3) forall (doall, pardo, doacross)

degree of parallelism – dynamic
implemented with fork/join
vector and matrix operation

Ex:
```
for i:= 1 to m do
  for j:= 1 to n do
    for k:= 1 to p do
      A[i,j,k]:= 0
    endfor
  endfor
endfor
```
→

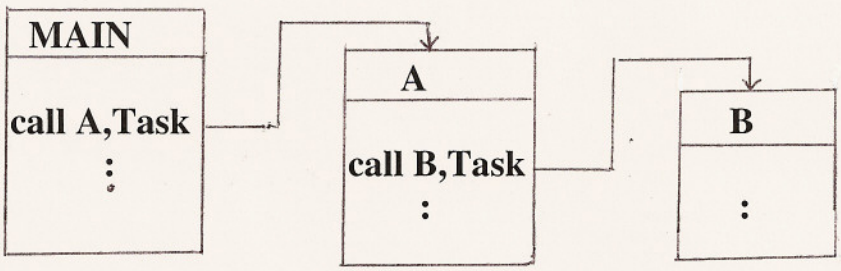
Fortran 95

```
forall (i=1:m, j=1:n, k=1:p)
    A(i,j,k) = 0
end forall
```
// construct

forall (i=1:m, j=1:m, k=1:p) A(i,j,k) = 0 // statement

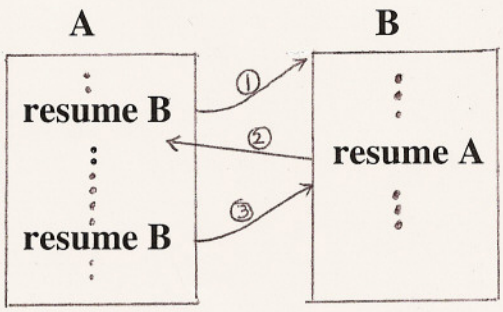
(4) parallel subprogram
declare processes, tasks, procedures, subroutines for
parallel execution

Ex. PL/I Task - coordination is difficult.



Note. Coroutine (Symmetric subprograms)

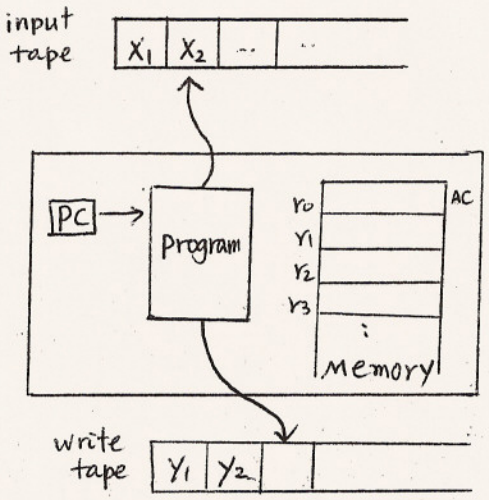
Simula 67
Modula 2



Note: single thread

Models of Parallel Computers

RAM model

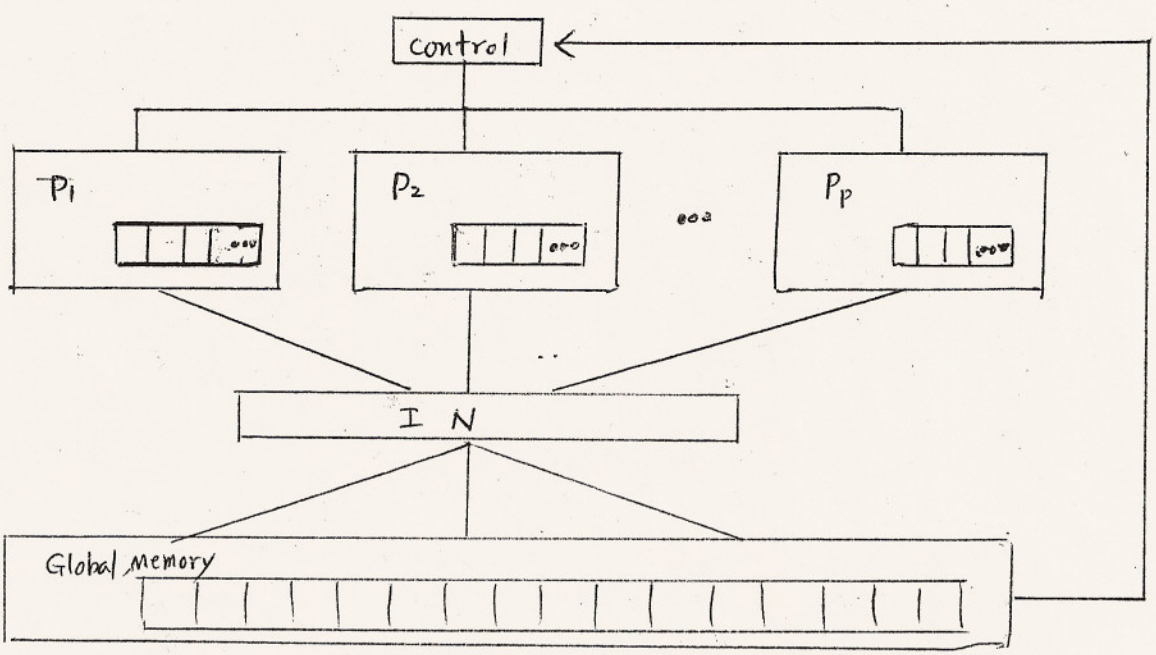


load
store
read
write
add/sub/mul/div
test
jump
halt

TM $\Longrightarrow$ RAM $\Longrightarrow$ RASP $\Longrightarrow$ computer

time-complexity : $f(n) = n^2$, where n is input size

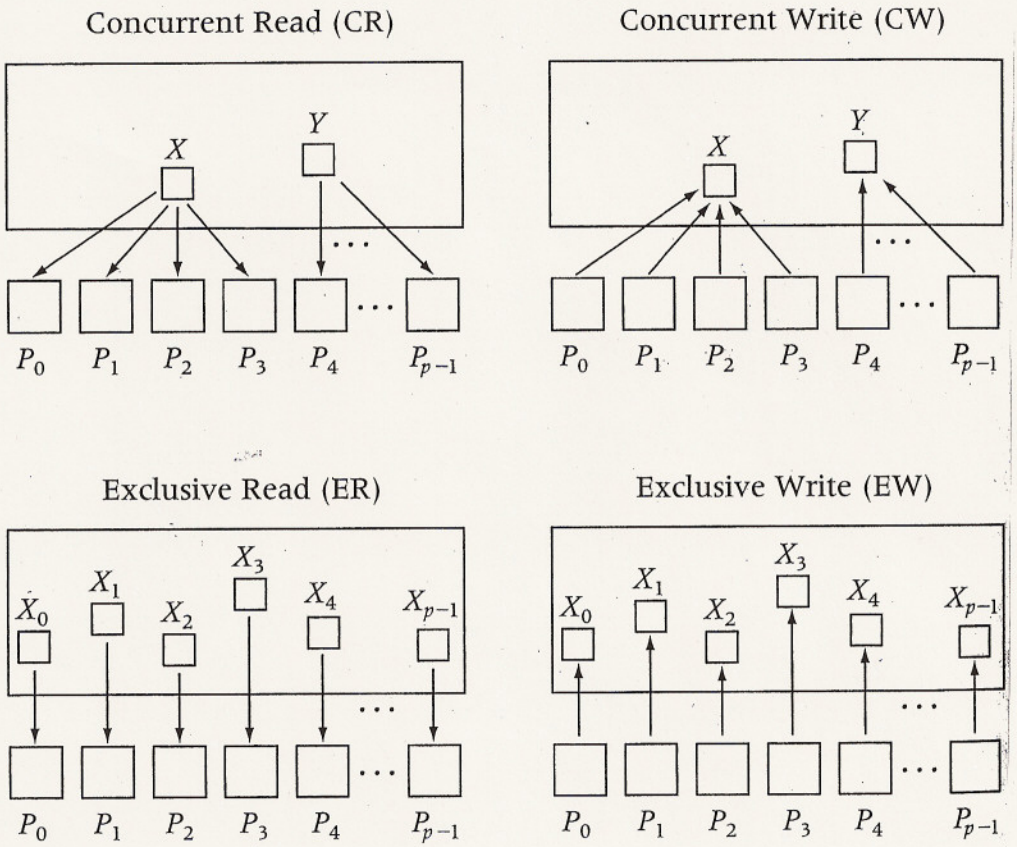
space-complexity : $S(n) = n+1$



Shared Memory **PRAM**

Def: cost = (parallel time complexity) X (#processors used)

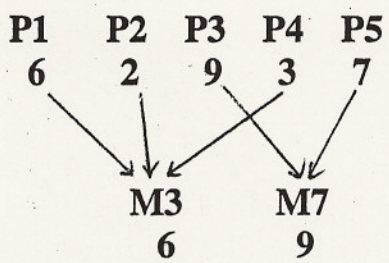
PRAM models



EREW
CREW
CRCW

1. **common – should be same value**
2. **arbitrary – randomly chosen**
3. **priority**

Ex. $P_i > P_{i+1}$



EREW

Ex. Searching

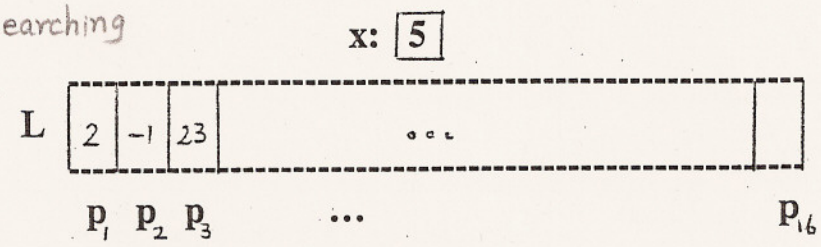
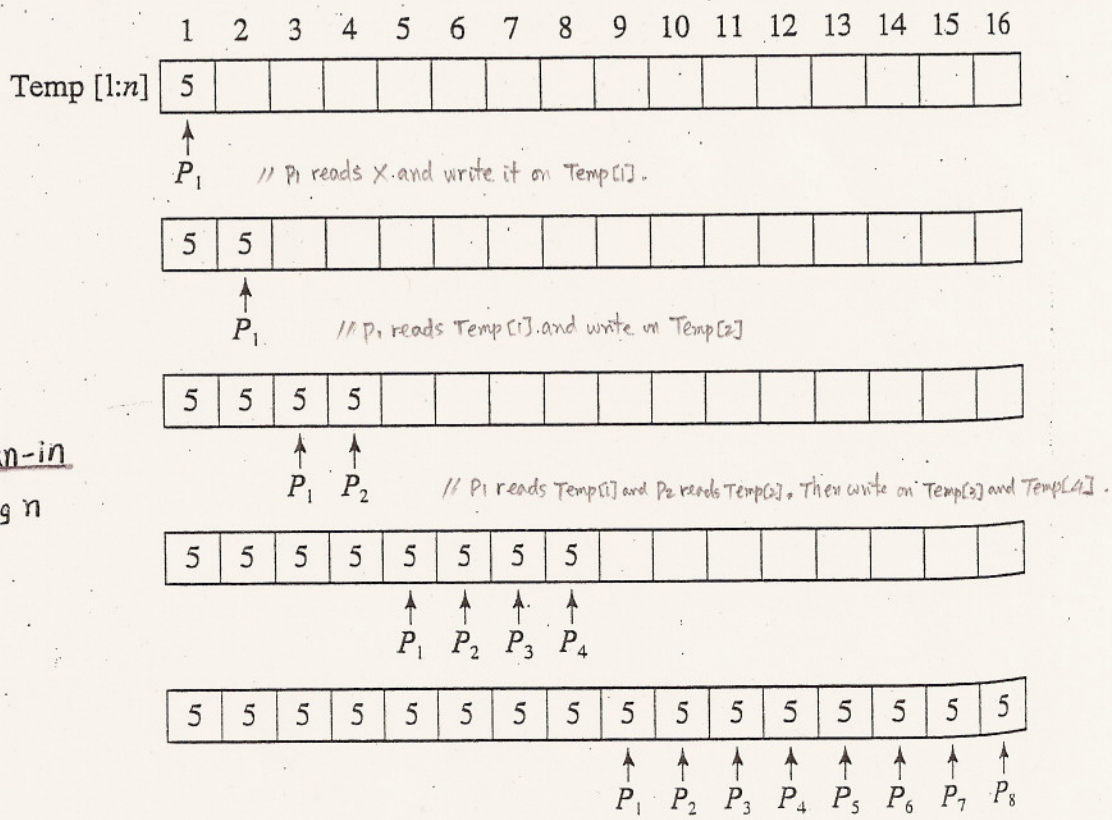


Figure 15.3 Broadcasting the value $X = 5$ in $\log_2 n$ steps



binary fan-in

$n \rightarrow \log n$

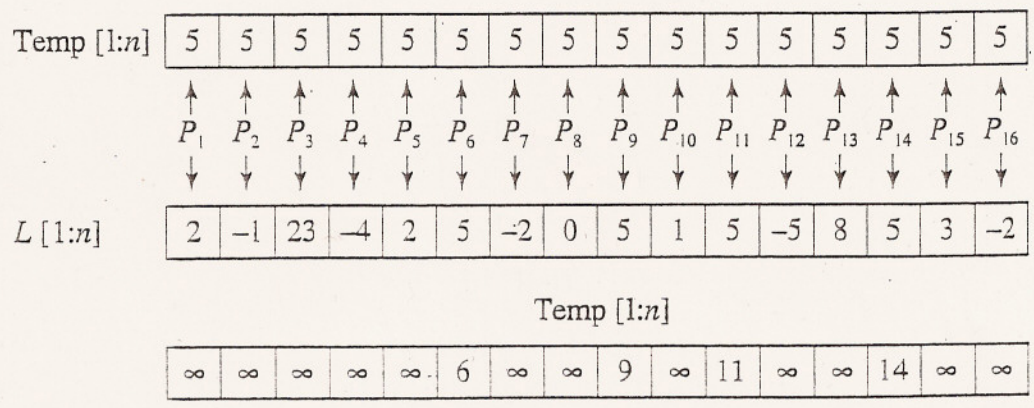


Figure 15.4 A single parallel comparison step between search elements and list elements

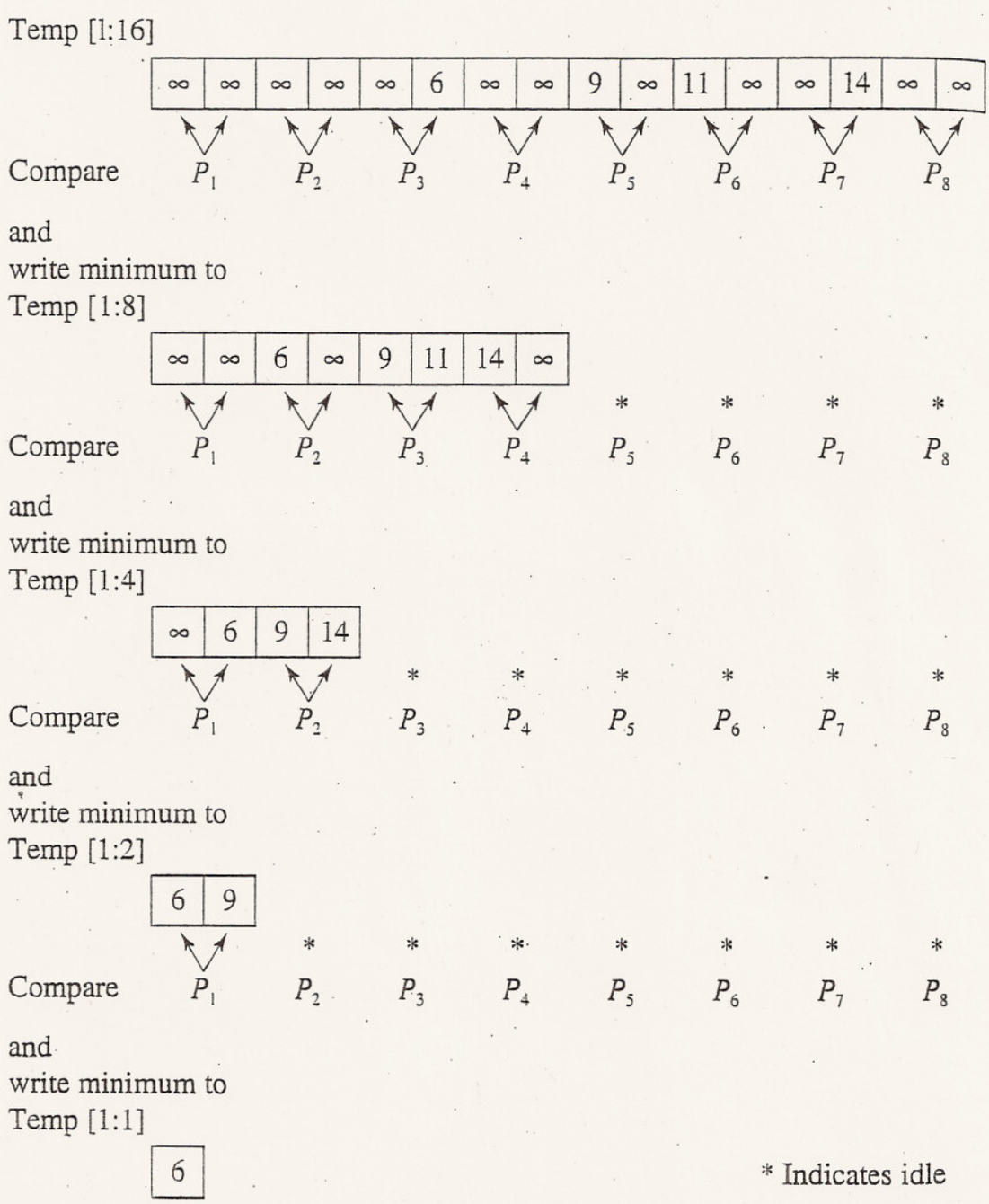
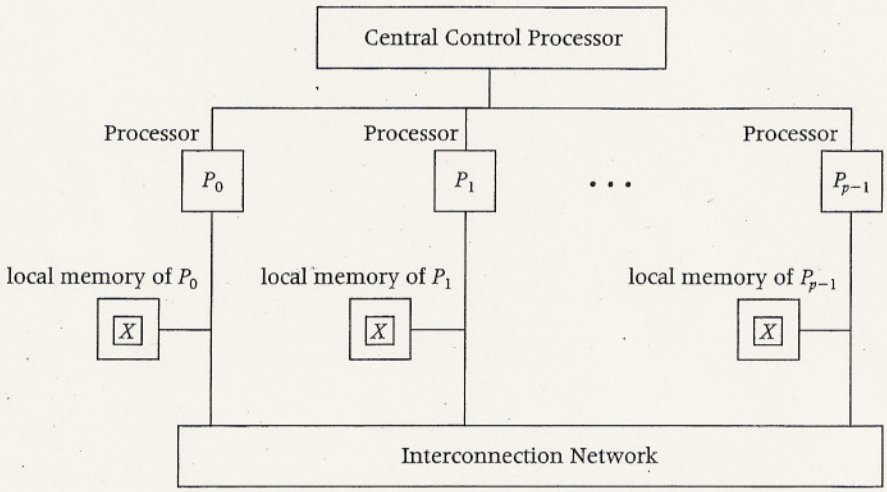


Figure 15.5 Binary fan-in technique for computing the minimum value in *Temp*[1:*n*] in $\log_2 n$ parallel comparison steps

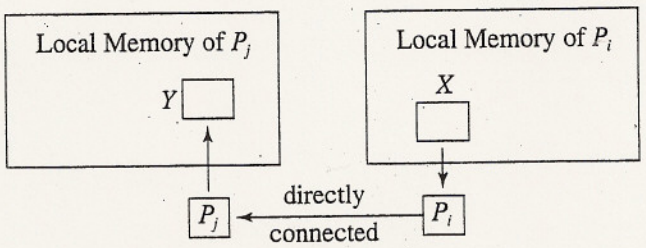
$$S_p = \frac{15}{4}$$

Distributed Memory: Interconnection Networks



- **message passing**
- **P: x // variable x in process P's (local) memory //**

$\mathbf{P}_j : Y \Leftarrow \mathbf{P}_i : X$



$\mathbf{P}_j : Y \Leftarrow \mathbf{P}_i : X$

15.2.7 Example: Searching on Meshes

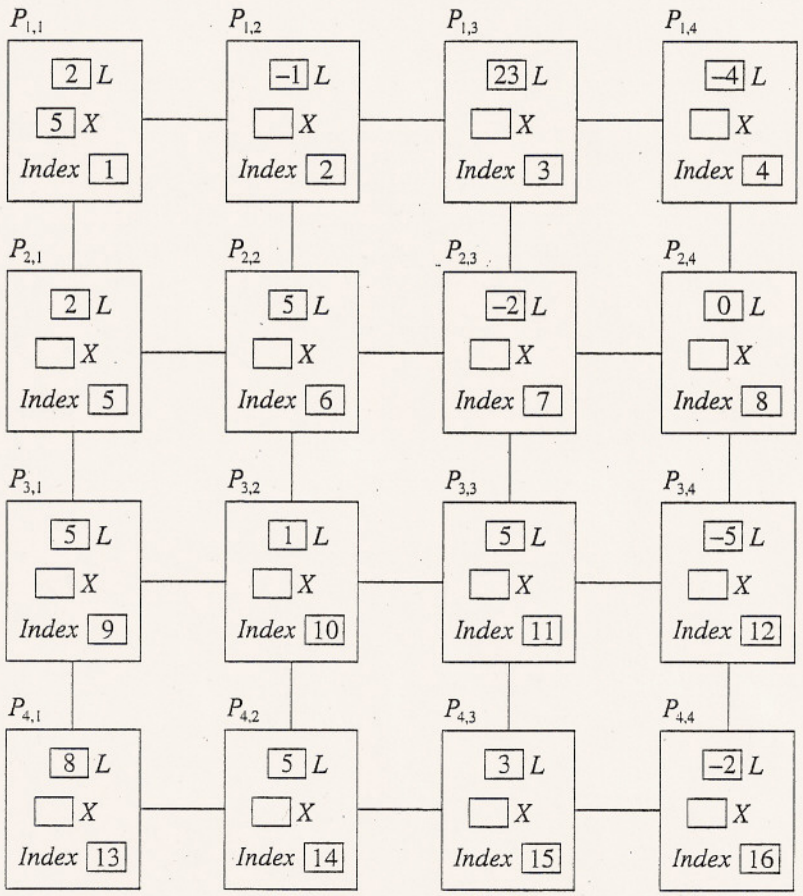
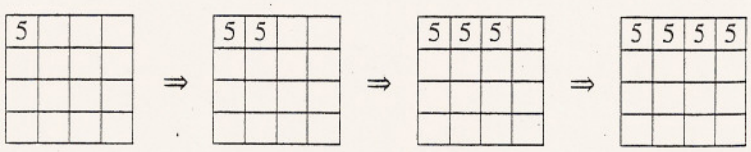
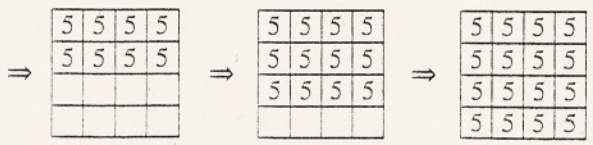


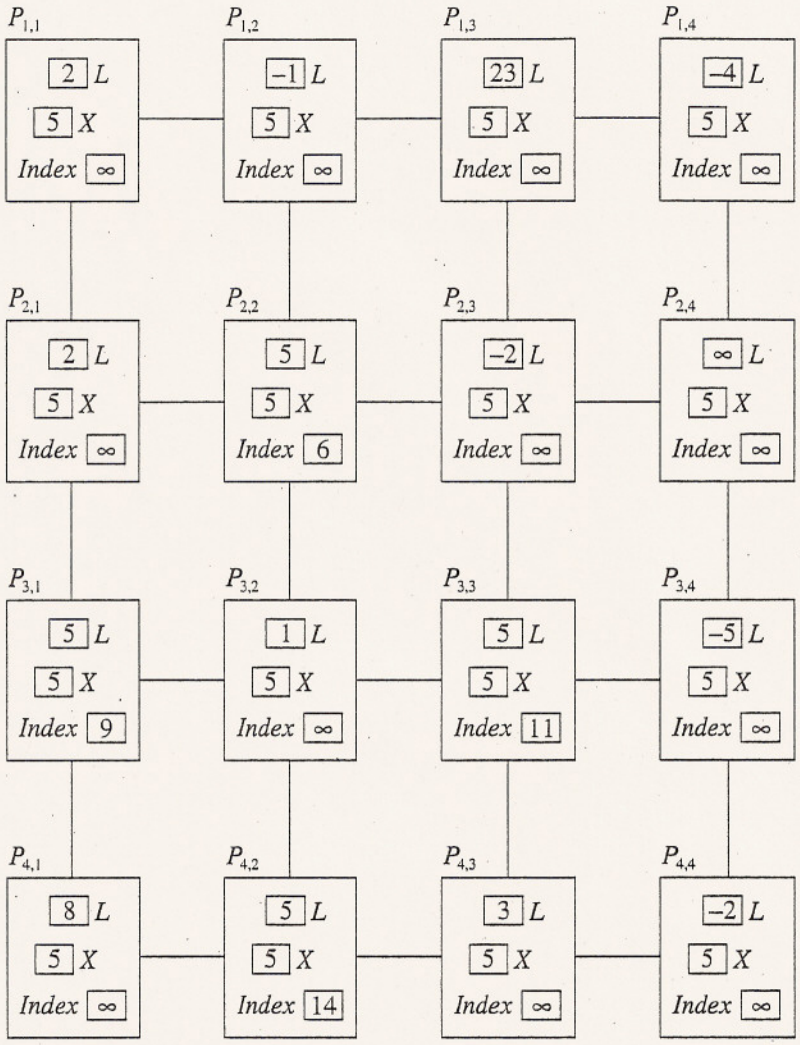
Figure 15.13 Broadcasting the value 5 throughout the distributed variable X on $M_{4,4}$

Phase 1: Broadcast 5 across the first row



Phase 2: Broadcast 5 down from the ith row to the $(i+1)$st row, $i = 1, \ldots, q-1$

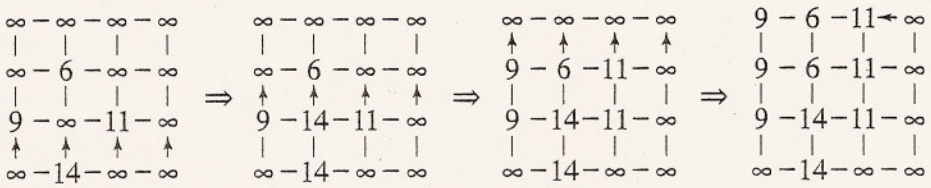




Figure

State of the distributed variables *L*, *X*, and *Index* after the value 5 has been broadcast throughout *X* and the single parallel comparison step between *X* and *Index* has been performed

Phase 1: Compute column minimums



Phase 2: Compute first row minimum

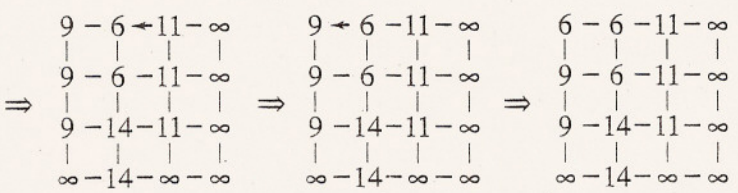


Figure 15.14

Computing the minimum on a two-dimensional mesh. Arrows indicate how the minimums are computed at each stage. The final result is in $P_{1,1}$.