Graph Algorithms (Chapter 10)

Alexandre David
B2-206

Today

- Recall on graphs.
- Minimum spanning tree (Prim's algorithm).
- Single-source shortest paths (Dijkstra's algorithm).
- All-pair shortest paths (Floyd's algorithm).
- Connected components.

Graphs – Definition

- A graph is a pair (V,E)
  - V finite set of vertices.
  - E finite set of edges.
    $e \in E$ is a pair (u,v) of vertices.
    Ordered pair → directed graph.
    Unordered pair → undirected graph.

V=
E=

V=
E=

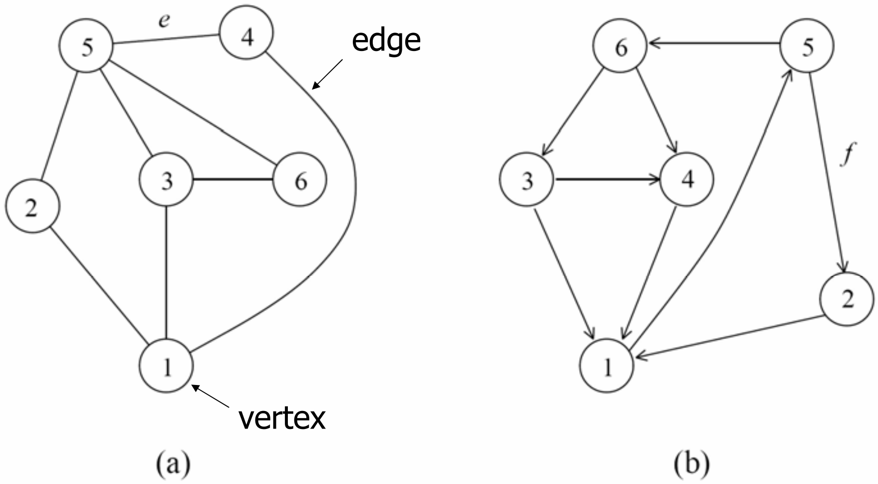


Figure 10.1 (a) An undirected graph and (b) a directed graph.

Graphs – Edges

- Directed graph:
  - $(u,v) \in E$ is incident from u and incident to v.
  - $(u,v) \in E$: vertex v is adjacent to u.
- Undirected graph:
  - $(u,v) \in E$ is incident on u and v.
  - $(u,v) \in E$: vertices u and v are adjacent to each other.

4 adjacent to 6

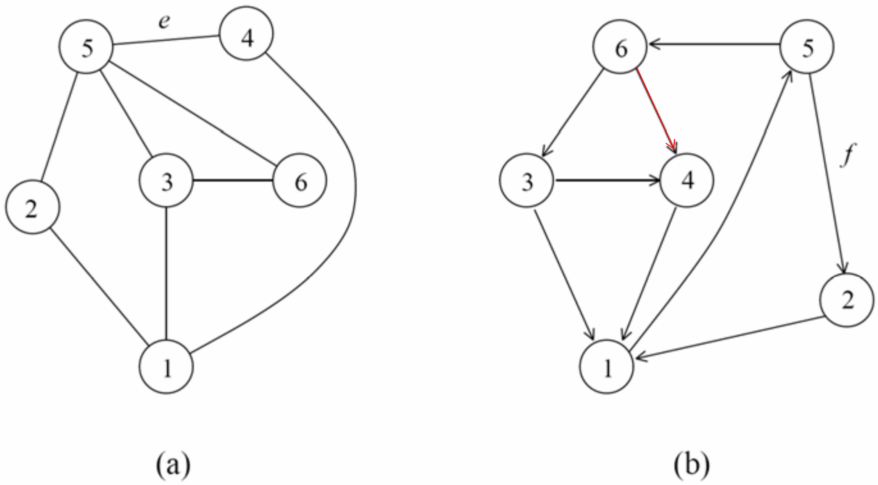


Figure 10.1 (a) An undirected graph and (b) a directed graph.

Graphs – Paths

- A path is a sequence of adjacent vertices.
  - Length of a path = number of edges.
  - Path from v to $u \Rightarrow u$ is reachable from v.
  - Simple path: All vertices are distinct.
  - A path is a cycle if its starting and ending vertices are the same.
  - Simple cycle: All intermediate vertices are distinct.

Simple path:
Simple cycle:
Non simple cycle:

Simple path:
Simple cycle:
Non simple cycle:

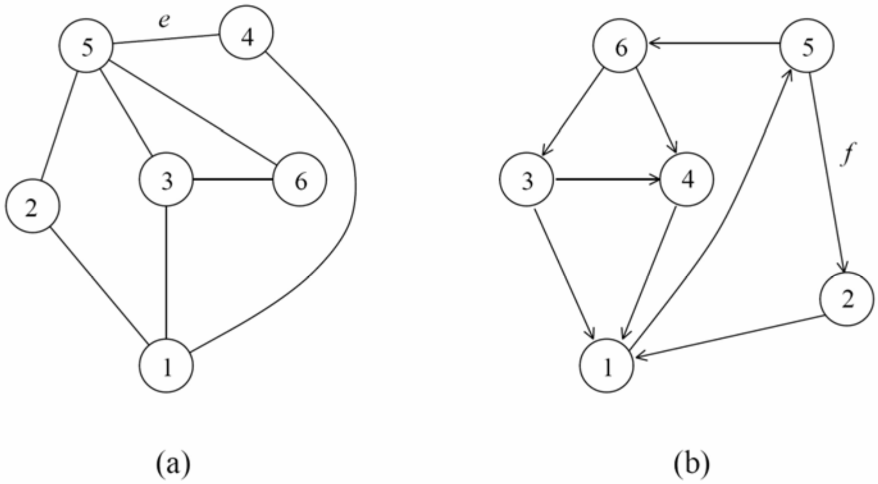


Figure 10.1 (a) An undirected graph and (b) a directed graph.

Graphs

- Connected graph: $\exists$ path between any pair.
- $G'=(V',E')$ sub-graph of $G=(V,E)$ if $V' \subseteq V$ and $E' \subseteq E$.
- Sub-graph of G induced by V': Take all edges of E connecting vertices of $V' \subseteq V$.
- Complete graph: Each pair of vertices adjacent.
- Tree: connected acyclic graph.

Sub-graph:
Induced sub-graph:

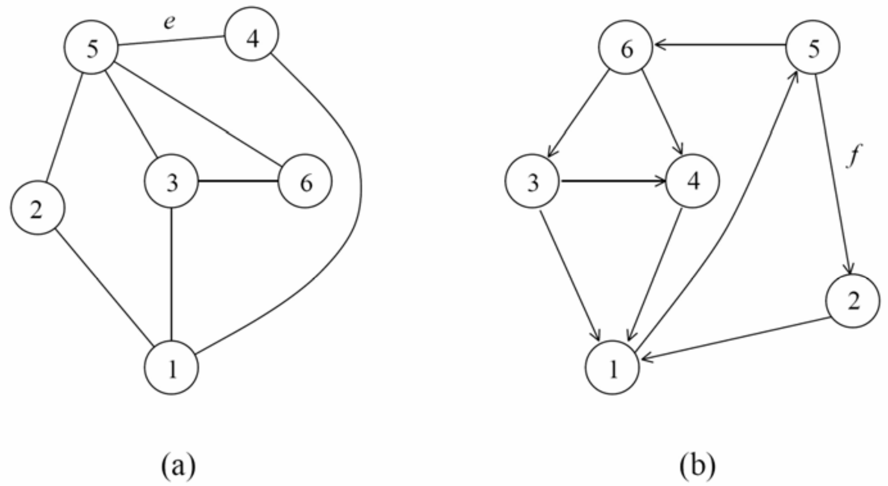
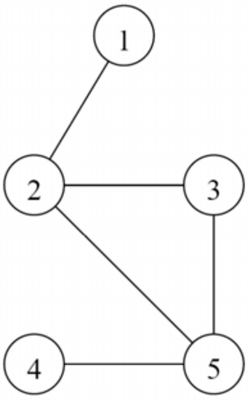


Figure 10.1 (a) An undirected graph and (b) a directed graph.

Graph Representation

- Sparse graph ($|E|$ much smaller than $|V|^2$):
  - Adjacency list representation.
- Dense graph:
  - Adjacency matrix.
- For weighted graphs (V,E,w): weighted adjacency list/matrix.

$$a_{i,j} = \begin{cases} 1 & if\ (v_i, v_j) \in E \\ 0 & otherwise \end{cases}$$



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

|V|

$|V|^2$ entries

Figure 10.2 An undirected graph and its adjacency matrix representation.

Undirected graph ⇒ symmetric adjacency matrix.

|V|+|E| entries

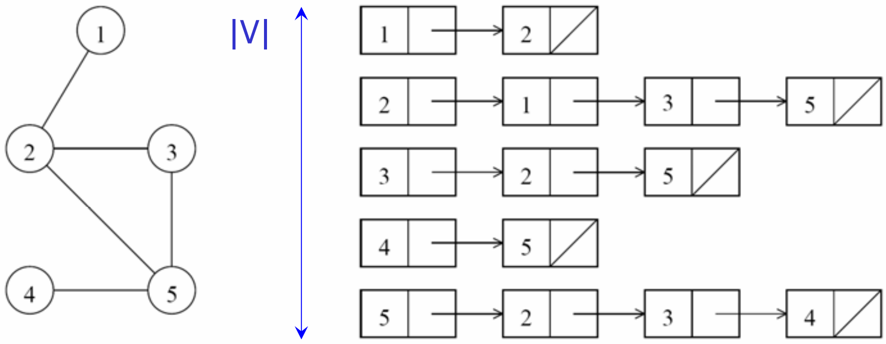


Figure 10.3 An undirected graph and its adjacency list representation.

Minimum Spanning Tree

- We consider undirected graphs.
- Spanning tree of (V,E) = sub-graph
  - being a tree and
  - containing all vertices V.
- Minimum spanning tree of (V,E,w) = spanning tree with minimum weight.
- Example: minimum length of cable to connect a set of computers.

Spanning Trees

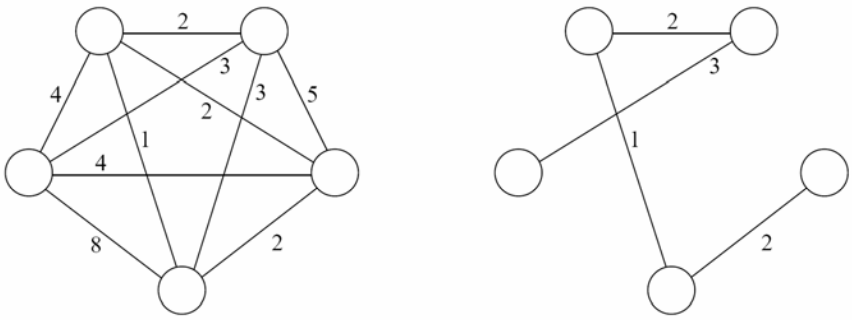


Figure 10.4 An undirected graph and its minimum spanning tree.

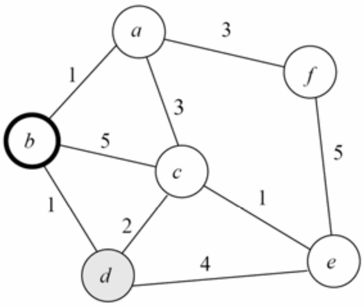
Prim's Algorithm

- Greedy algorithm:
  - Select a vertex.
  - Choose a new vertex and edge guaranteed to be in a spanning tree of minimum cost.
  - Continue until all vertices are selected.

```
procedure PRIM_MST(V, E, w, r)
begin
    V_T := {r};                                    Vertices of minimum spanning tree.
    d[r] := 0;
    for all v ∈ (V − V_T) do                       Weights from V_T to V.
        if edge (r, v) exists set d[v] := w(r, v);
        else set d[v] := ∞;
    while V_T ≠ V do
    begin
select  find a vertex u such that d[u] := min{d[v]|v ∈ (V − V_T)};
add     V_T := V_T ∪ {u};
update  for all v ∈ (V − V_T) do
            d[v] := min{d[v], w(u, v)};
    endwhile
end PRIM_MST
```

Algorithm 10.1 Prim's sequential minimum spanning tree algorithm.

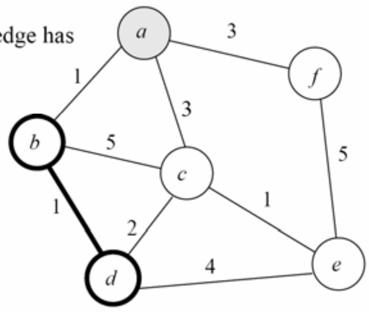
(a) Original graph



	a	b	c	d	e	f
d[]	1	0	5	1	∞	∞

	a	b	c	d	e	f
a	0	1	3	∞	∞	3
b	1	0	5	1	∞	∞
c	3	5	0	2	1	∞
d	∞	1	2	0	4	∞
e	∞	∞	1	4	0	5
f	2	∞	∞	∞	5	0

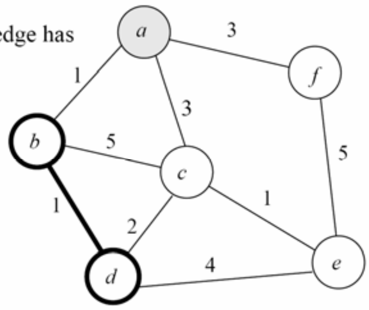
(b) After the first edge has been selected



	a	b	c	d	e	f
d[]	1	0	2	1	4	∞

	a	b	c	d	e	f
a	0	1	3	∞	∞	3
b	1	0	5	1	∞	∞
c	3	5	0	2	1	∞
d	∞	1	2	0	4	∞
e	∞	∞	1	4	0	5
f	2	∞	∞	∞	5	0

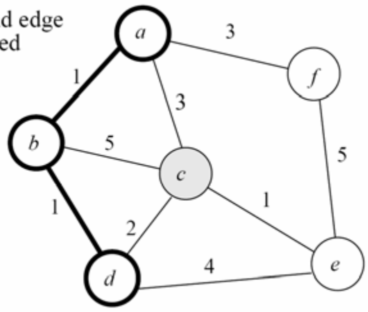
(b) After the first edge has been selected



	a	b	c	d	e	f
d[]	1	0	2	1	4	∞

	a	b	c	d	e	f
a	0	1	3	∞	∞	3
b	1	0	5	1	∞	∞
c	3	5	0	2	1	∞
d	∞	1	2	0	4	∞
e	∞	∞	1	4	0	5
f	2	∞	∞	∞	5	0

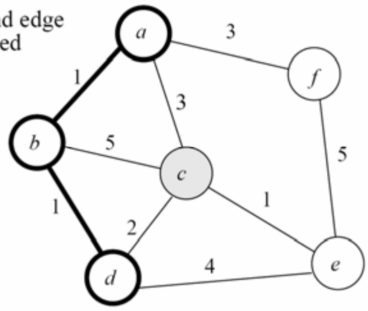
(c) After the second edge has been selected



	a	b	c	d	e	f
d[]	1	0	2	1	4	3

	a	b	c	d	e	f
a	0	1	3	∞	∞	3
b	1	0	5	1	∞	∞
c	3	5	0	2	1	∞
d	∞	1	2	0	4	∞
e	∞	∞	1	4	0	5
f	2	∞	∞	∞	5	0

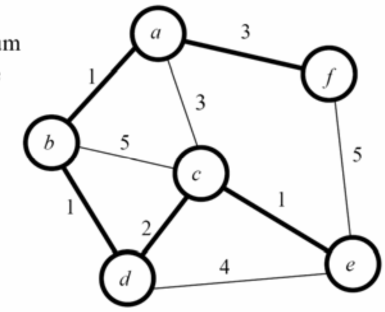
(c) After the second edge has been selected



	a	b	c	d	e	f
d[]	1	0	2	1	4	3

	a	b	c	d	e	f
a	0	1	3	∞	∞	3
b	1	0	5	1	∞	∞
c	3	5	0	2	1	∞
d	∞	1	2	0	4	∞
e	∞	∞	1	4	0	5
f	2	∞	∞	∞	5	0

(d) Final minimum spanning tree



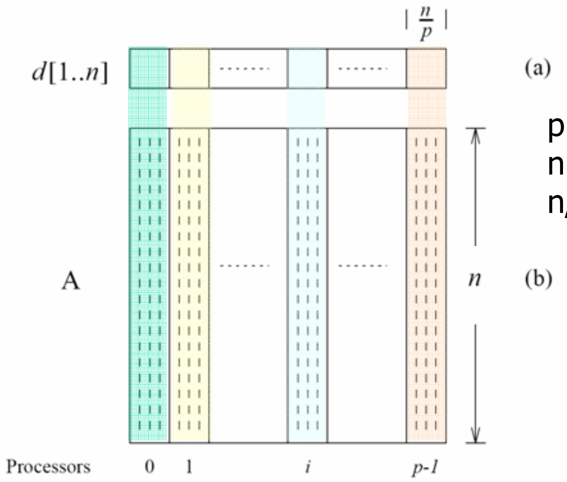
	a	b	c	d	e	f
d[]	1	0	2	1	1	3

	a	b	c	d	e	f
a	0	1	3	∞	∞	3
b	1	0	5	1	∞	∞
c	3	5	0	2	1	∞
d	∞	1	2	0	4	∞
e	∞	∞	1	4	0	5
f	2	∞	∞	∞	5	0

Prim's Algorithm

- Complexity $\Theta(n^2)$.
- Cost of the minimum spanning tree: $\sum_{v \in V} d[v]$
- How to parallelize?
  - Iterative algorithm.
  - Any d[v] may change after every loop.
  - But possible to run each iteration in parallel.

1-D Block Mapping



p processes
n vertices
n/p vertices per process

22

Parallel Prim's Algorithm

1-D block partitioning: V_i per P_i.
For each iteration:

- P_i computes a local min $d_i[u]$.
- All-to-one reduction to P_0 to compute the global min.
- One-to-all broadcast of u.
- Local updates of d[v].

Every process needs a column of the adjacency matrix to compute the update.
$\Theta(n^2/p)$ space per process.

Analysis

- The cost to select the minimum entry is $O(n/p + \log p)$.
- The cost of a broadcast is $O(\log p)$.
- The cost of local update of the d vector is $O(n/p)$.
- The parallel run-time per iteration is $O(n/p + \log p)$.
- The total parallel time (n iterations) is given by $O(n^2/p + n \log p)$.

Analysis

- Efficiency = Speedup/# of processes: $E=S/p=1/(1+\Theta((p \log p)/n)$.
- Maximal degree of concurrency = n.
- To be cost-optimal we can only use up to $n/\log n$ processes.
- Not very scalable.

max at $n^2/p = \Theta(n \log p)$, with bound $p=O(n)$

27-04-2007 Alexandre David, MVP'07 25

Keep cost optimality: $p \log p=O(n)$, $\log p+\log\log p=O(\log p)=O(\log n) \rightarrow p=O(n/\log n)$.

$pT_P=T_S+T_0 \rightarrow T_0=O(pn \log p)=O((p \log p)^2)$.

Single-Source Shortest Paths: Dijkstra's Algorithm

- For (V,E,w), find the shortest paths from a vertex to all other vertices.
  - Shortest path=minimum weight path.
  - Algorithm for directed & undirected with non negative weights.
- Similar to Prim's algorithm.
  - Prim: store d[u] minimum cost edge connecting a vertex of V_T to u.
  - Dijkstra: store l[u] minimum cost to reach u from s by a path in V_T.

Parallel formulation: Same as Prim's algorithm.

```
procedure DIJKSTRA_SINGLE_SOURCE_SP(V, E, w, s)
begin
    V_T := {s};
    for all v ∈ (V − V_T) do
        if (s, v) exists set l[v] := w(s, v);
        else set l[v] := ∞;
    while V_T ≠ V do
    begin
        find a vertex u such that l[u] := min{l[v] | v ∈ (V − V_T)};
        V_T := V_T ∪ {u};
        for all v ∈ (V − V_T) do
            l[v] := min{l[v], l[u] + w(u, v)};
    endwhile
end DIJKSTRA_SINGLE_SOURCE_SP
```

Algorithm 10.2 Dijkstra's sequential single-source shortest paths algorithm.

All-Pairs Shortest Paths

- For (V,E,w), find the shortest paths between all pairs of vertices.
  - Dijkstra's algorithm: Execute the single-source algorithm for *n* vertices → $\Theta(n^3)$.
  - Floyd's algorithm.

All-Pairs Shortest Paths – Dijkstra – Parallel Formulation

- Source-partitioned formulation: Each process has a set of vertices and compute the

  Up to n processes. Solve in $\Theta(n^2)$.

  - No communication, E=1, but maximal degree of concurrency = n. Poor scalability.
- Source-parallel formulation (p>n):
  - Partition the processes (p/n processes/subset),

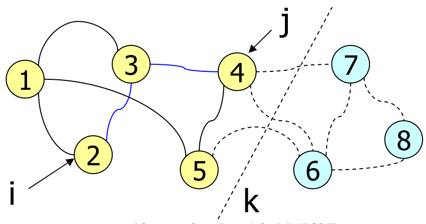
    Up to n^2 processes, $n^2/\log n$ for cost-optimal, in which case solve in $\Theta(n \log n)$.

  - In parallel: n single-source problems.

Floyd's Algorithm

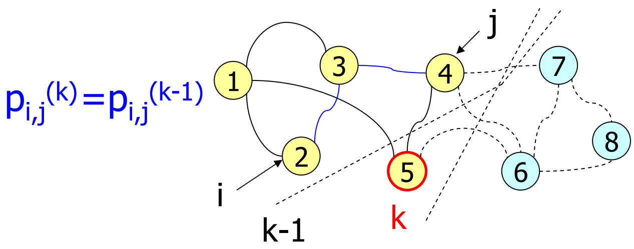
- For any pair of vertices v_i, $v_j \in V$, consider all paths from v_i to v_j whose intermediate vertices belong to the set $\{v_1,v_2,\ldots,v_k\}$.
- Let $p_{i,j}^{(k)}$ (of weight $d_{i,j}^{(k)}$) be the minimum-weight path among them.

$p_{i,j}^{(k)}$



Floyd's Algorithm

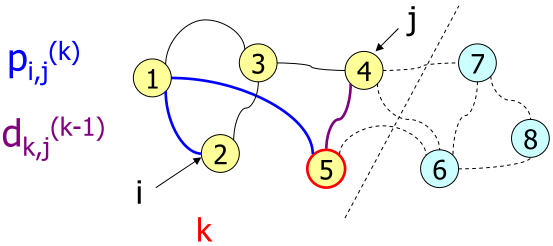
- If vertex v_k is not in the shortest path from v_i to v_j, then $p_{i,j}^{(k)} = p_{i,j}^{(k-1)}$.



Floyd's Algorithm

- If v_k is in $p_{i,j}^{(k)}$, then we can break $p_{i,j}^{(k)}$ into two paths - one from v_i to v_k and one from v_k to v_j . Each of these paths uses vertices from $\{v_1,v_2,\ldots,v_{k-1}\}$.

$$d_{i,j}^{(k)}=d_{i,k}^{(k-1)}+d_{k,j}^{(k-1)}$$



Floyd's Algorithm

- Recurrence equation:

$$d_{i,j}^{(k)} = \begin{cases} w(v_i, v_j) & if\ k = 0 \\ \min\left(d_{i,j}^{(k-1)}, d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)}\right) & if\ k \geq 1 \end{cases}$$

- Length of shortest path from v_i to v_j = $d_{i,j}^{(n)}$. Solution set = a matrix.

Floyd's Algorithm

How to parallelize?

$\Theta(n^3)$

Also works *in place*.

```
procedure FLOYD_ALL_PAIRS_SP(A)
begin
    D^(0) = A;
    for k := 1 to n do
        for i := 1 to n do
            for j := 1 to n do
                d_{i,j}^(k) := min(d_{i,j}^(k-1), d_{i,k}^(k-1) + d_{k,j}^(k-1));
end FLOYD_ALL_PAIRS_SP
```

$$d_{i,j}^{(k)} := \min\left(d_{i,j}^{(k-1)}, d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)}\right);$$

Algorithm 10.3 Floyd's all-pairs shortest paths algorithm. This program computes the all-pairs shortest paths of the graph $G = (V, E)$ with adjacency matrix A.

Parallel Formulation

- 2-D block mapping:
  - Each of the p processes has a sub-matrix $(n/\sqrt{p})^2$ and computes its $D^{(k)}$.
  - Processes need access to the corresponding k row and column of $D^{(k-1)}$.
  - k^{th} iteration: Each processes containing part of the k^{th} row sends it to the other processes in the same column. Same for column broadcast on rows.

2-D Mapping

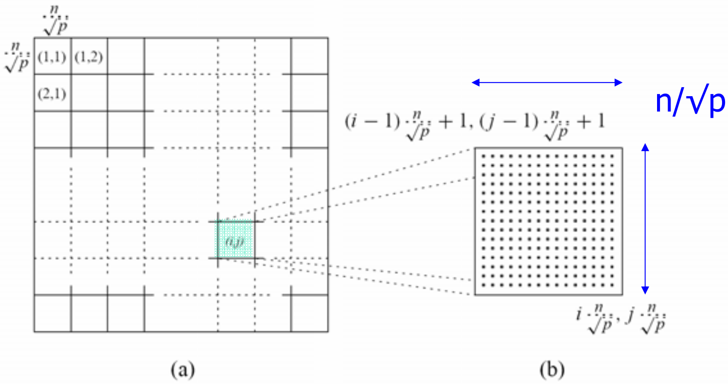


Figure 10.7 (a) Matrix $D^{(k)}$ distributed by 2-D block mapping into $\sqrt{p} \times \sqrt{p}$ subblocks, and (b) the subblock of $D^{(k)}$ assigned to process $P_{i,j}$.

Communication

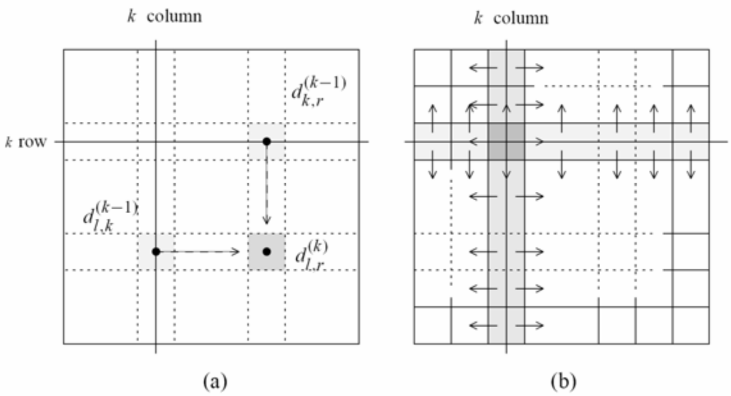


Figure 10.8 (a) Communication patterns used in the 2-D block mapping. When computing $d_{i,j}^{(k)}$, information must be sent to the highlighted process from two other processes along the same row and column. (b) The row and column of $\sqrt{p}$ processes that contain the k^{th} row and column send them along process columns and rows.

Parallel Algorithm

procedure FLOYD_2DBLOCK($D^{(0)}$)
begin
  for $k := 1$ **to** n **do**
  begin
    each process $P_{i,j}$ that has a segment of the k^{th} row of $D^{(k-1)}$;
      broadcasts it to the $P_{*,j}$ processes;
    each process $P_{i,j}$ that has a segment of the k^{th} column of $D^{(k-1)}$;
      broadcasts it to the $P_{i,*}$ processes;
    each process waits to receive the needed segments;
    each process $P_{i,j}$ computes its part of the $D^{(k)}$ matrix;
  end
end FLOYD_2DBLOCK

Algorithm 10.4 Floyd's parallel formulation using the 2-D block mapping. $P_{*,j}$ denotes all the processes in the j^{th} column, and $P_{i,*}$ denotes all the processes in the i^{th} row. The matrix $D^{(0)}$ is the adjacency matrix.

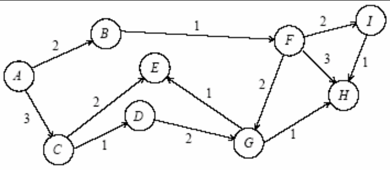
Analysis

$$T_P = \overbrace{\Theta\left(\frac{n^3}{p}\right)}^{\text{computation}} + \overbrace{\Theta\left(\frac{n^2}{\sqrt{p}}\log p\right)}^{\text{communication}}.$$

- E=1/(1+Θ(($\sqrt{p}$ log*p*)/*n*).
- Cost optimal if up to O((*n*/ log*n*)^2) processes.
- Possible to improve: pipelined 2-D block mapping: No broadcast, send to neighbour. Communication: Θ(n), up to O(n^2) processes & cost optimal.

All-Pairs Shortest Paths: Matrix Multiplication *Based* Algorithm

- Multiplication of the weighted adjacency matrix with itself – except that we replace multiplications by additions, and additions by minimizations.
- The result is a matrix that contains shortest paths of length 2 between any pair of nodes.
- It follows that A^n contains all shortest paths.



$$A^1 = \begin{pmatrix} 0 & 2 & 3 & \infty & \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty & \infty & 1 & \infty & \infty & \infty \\ \infty & \infty & 0 & 1 & 2 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & 0 & \infty & \infty & 2 & \infty & \infty \\ \infty & \infty & \infty & \infty & 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 & 2 & 3 & 2 \\ \infty & \infty & \infty & \infty & 1 & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 1 & 0 \end{pmatrix}$$

$$A^2 = \begin{pmatrix} \vdots \\ \infty & \infty & \infty & \infty & 1 & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 1 & 0 \end{pmatrix}$$

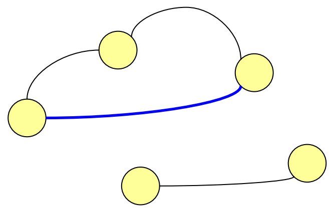
Serial algorithm not optimal but we can use $n^3/\log n$ processes to run in $O(\log^2 n)$.

$$A^4 = \begin{pmatrix} 0 & 2 & 3 & 4 & 5 & 3 & 5 & 6 & 5 \\ \infty & 0 & \infty & \infty & 4 & 1 & 3 & 4 & 3 \\ \infty & \infty & 0 & 1 & 2 & \infty & 3 & 4 & \infty \\ \infty & \infty & \infty & 0 & 3 & \infty & 2 & 3 & \infty \\ \infty & \infty & \infty & \infty & 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 3 & 0 & 2 & 3 & 2 \\ \infty & \infty & \infty & \infty & 1 & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 1 & 0 \end{pmatrix}$$

$$A^8 = \begin{pmatrix} 0 & 2 & 3 & 4 & 5 & 3 & 5 & 6 & 5 \\ \infty & 0 & \infty & \infty & 4 & 1 & 3 & 4 & 3 \\ \infty & \infty & 0 & 1 & 2 & \infty & 3 & 4 & \infty \\ \infty & \infty & \infty & 0 & 3 & \infty & 2 & 3 & \infty \\ \infty & \infty & \infty & \infty & 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 3 & 0 & 2 & 3 & 2 \\ \infty & \infty & \infty & \infty & 1 & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 1 & 0 \end{pmatrix}$$

Transitive Closure

- Find out if any two vertices are connected.
- $G^*=(V,E^*)$ where $E^*=\{(v_i,v_j)|\exists$ a path from v_i to v_j in $G\}$.



Transitive Closure

- Start with D=($a_{i,j}$ or ∞).
- Apply one all-pairs shortest paths algorithm.
- Solution:

$$a^*_{i,j} = \begin{cases} \infty & if\ d_{i,j} = \infty \\ 1 & if\ d_{i,j} > 0\ or\ i = j \end{cases}$$

Also possible to modify Floyd's algorithm by replacing + by logical or and min by logical and.

Connected Components

- Connected components of $G=(V,E)$ are the maximal disjoint sets $C_1,\ldots,C_k$ s.t. $V=\bigcup C_k$ and $u,v \in C_i$ iff u reachable from v and v reachable from u.

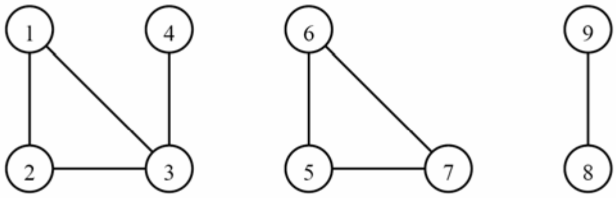


Figure 10.10 A graph with three connected components: {1, 2, 3, 4}, {5, 6, 7}, and {8, 9}.

DFS Based Algorithm

- DFS traversal of the graph → forest of (DFS) spanning trees.

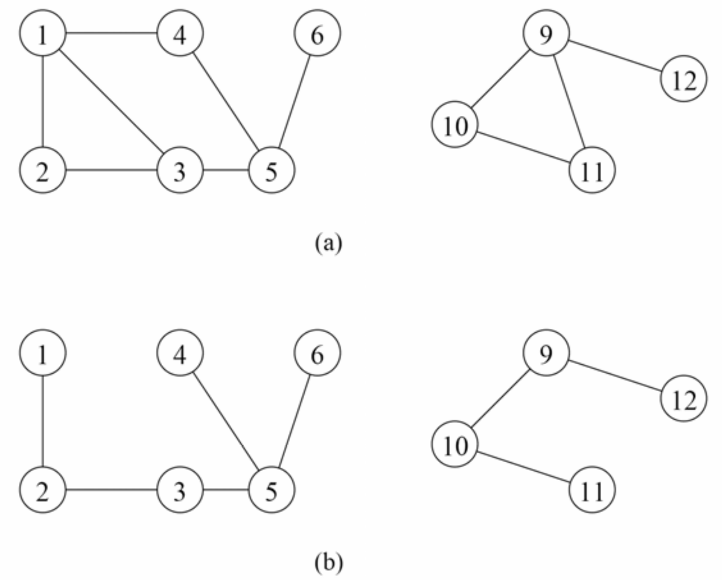
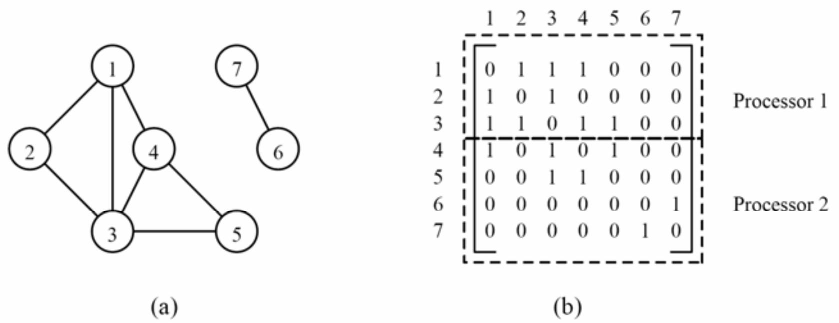


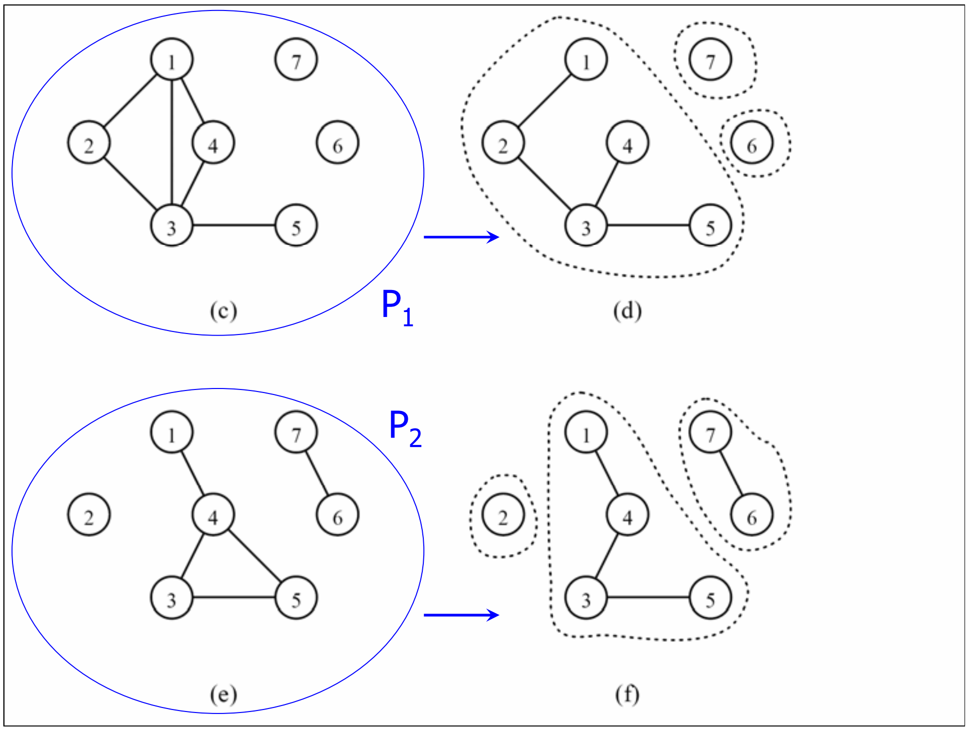
Figure 10.11 Part (b) is a depth-first forest obtained from depth-first traversal of the graph in part (a). Each of these trees is a connected component of the graph in part (a).

Parallel Formulation

- Partition G into p sub-graphs. P_i has $G_i=(V,E_i)$.
  - Each P_i computes the spanning forest of G_i.
  - Merge the forests pair-wise.
- Each merge possible in $\Theta(n)$.
  - Not described in the book – out of scope.
  - Find if an edge of A has its vertices in B:
    - no for all $\rightarrow$ union of 2 disjoint sets.
    - yes for one $\rightarrow$ merge.



Partition the adjacency matrix.
1-D partitioning in p stripes of n/p consecutive rows.


1
7
2
4
6
3
5
(c)
P1
1
7
2
4
6
3
5
(d)
1
7
P2
2
4
6
3
5
(e)
1
7
2
4
6
3
5
(f)

Analysis

$$T_P = \overbrace{\Theta\left(\frac{n^2}{p}\right)}^{\text{local computation}} + \overbrace{\Theta(n \log p)}^{\text{forest merging}}.$$

- E=1/(1+Θ((p log$p$)/n).
- Up to O(n/ log$n$) to be cost-optimal.
- Performance similar to Prim's algorithm.