DAA- Unit VI
Case Studies

By Prof. B.A.Khivsara
Assistant Professor
Department of Computer Engineering
SNJB’s KBJ COE, Chandwad
Distributed Operating Systems

Bully Algorithms

All Pair Shortest Path (Floyed Warshall's algo)

Process Termination (Diskstra Scolten)
Election Algorithms

- Many distributed algorithms such as mutual exclusion and deadlock detection require a *coordinator process*.
- When the coordinator process fails, the distributed group of processes must execute an *election algorithm* to determine a new coordinator process.
- These algorithms will assume that each active process has a unique *priority id*.
The Bully Algorithm

When any process, P, notices that the coordinator is no longer responding it initiates an election:

P sends an *election* message to all processes with higher id numbers.

If no one responds, P wins the election and becomes coordinator.

If a higher process responds, it takes over. Process P’s job is done.
The Bully Algorithm

At any moment, a process can receive an election message from one of its lower-numbered colleagues.

The receiver sends an OK back to the sender and conducts its own election.

Eventually only the bully process remains. The bully announces victory to all processes in the distributed group.
Bully Algorithm Example

- Process 4 notices 7 down.
- Process 4 holds an election.
- Process 5 and 6 respond, telling 4 to stop.
- Now 5 and 6 each hold an election.
Process 6 tells process 5 to stop.
Process 6 (the bully) wins and tells everyone.
If processes 7 comes up, starts elections again.
Distributed Operating Systems

Bully Algorithms

All Pair Shortest Path (Floyd-Warshall's algo)

Process Termination (Diskstra Scolten)
Algorithm All Pair shortest path (W,A)
{
    for i = 1 to n do {
        for j = 1 to n do {
    }
    For k=1 to n do{
        For i = 1 to n do{
            For j = 1 to n do{
            }
        }
    }
}
Example

\[ D^{(0)} = W \]

\[
\begin{array}{ccccc}
1 & 2 & 3 & 4 & 5 \\
1 & 0 & 3 & 8 & \infty & -4 \\
2 & \infty & 0 & \infty & 1 & 7 \\
3 & \infty & 4 & 0 & \infty & \infty \\
4 & 2 & \infty & -5 & 0 & \infty \\
5 & \infty & \infty & \infty & 6 & 0 \\
\end{array}
\]
Example

![Graph with labeled edges and a table]

\[ D^{(1)} \]

\[
\begin{array}{ccccc}
1 & 2 & 3 & 4 & 5 \\
1 & 0 & 3 & 8 & \infty & -4 \\
2 & \infty & 0 & \infty & 1 & 7 \\
3 & \infty & 4 & 0 & \infty & \infty \\
4 & 2 & 5 & -5 & 0 & -2 \\
5 & \infty & \infty & \infty & 6 & 0 \\
\end{array}
\]
Example

\[
\begin{array}{c|ccccc}
\hline
& 1 & 2 & 3 & 4 & 5 \\
\hline
1 & 0 & 3 & 8 & 4 & -4 \\
2 & \infty & 0 & \infty & 1 & 7 \\
3 & \infty & 4 & 0 & 5 & 11 \\
4 & 2 & 5 & -5 & 0 & -2 \\
5 & \infty & \infty & \infty & 6 & 0 \\
\hline
\end{array}
\]

\[D^{(2)}\]
Example

\[ D^{(3)} \]

\[
\begin{array}{ccccc}
1 & 2 & 3 & 4 & 5 \\
\hline
1 & 0 & 3 & 8 & 4 & -4 \\
2 & \infty & 0 & \infty & 1 & 7 \\
3 & \infty & \infty & 0 & 5 & 11 \\
4 & 2 & -1 & -5 & 0 & -2 \\
5 & \infty & \infty & \infty & 6 & 0 \\
\end{array}
\]
Example

\[ D^{(4)} \]

\[
\begin{array}{ccccc}
1 & 2 & 3 & 4 & 5 \\
1 & 0 & 3 & -1 & 4 & -4 \\
2 & 3 & 0 & -4 & 1 & -1 \\
3 & 7 & 4 & 0 & 5 & 3 \\
4 & 2 & -1 & -5 & 0 & -2 \\
5 & 8 & 5 & 1 & 6 & 0 \\
\end{array}
\]
Time complexity analysis

First double for loop takes $O(n^2)$

The nested 3 for loop takes $O(n^3)$

Thus, the whole algorithm takes $O(n^3)$ time
Distributed Operating Systems

- Bully Algorithms
- All Pair Shortest Path (Floyd-Warshall's algo)
- Process Termination (Diskstra Scolten)
Distributed Process Termination

Introduction

• A fundamental problem: To determine if a distributed computation has terminated.
• A non-trivial task since no process has complete knowledge of the global state, and global time does not exist.
• A distributed computation is globally terminated if every process is locally terminated and there is no message in transit between any processes.
• “Locally terminated” state is a state in which a process has finished its computation and will not restart any action unless it receives a message.
• In the termination detection problem, a particular process (or all of the processes) must infer when the underlying computation has terminated.
Distributed Process Termination-System Model

- At any given time, a process can be in only one of the two states: *active*, where it is doing local computation and *idle*, where the process has (temporarily) finished the execution of its local computation and will be reactivated only on the receipt of a message from another process.
- An active process can become idle at any time.
- An idle process can become active only on the receipt of a message from another process.
- Only active processes can send messages.
- A message can be received by a process when the process is in either of the two states, i.e., active or idle. On the receipt of a message, an idle process becomes active.
- The sending of a message and the receipt of a message occur as atomic actions.
Termination detection algorithm

[Dijkstra, Scholten]

- Simple stable property detection problem.
- Connected, undirected network graph \( G = (V,E) \).
- Assume:
  - Algorithm A begins with all nodes quiescent (only inputs enabled).
  - An input arrives at exactly one node.
  - Starting node need not be predetermined.
- From there, computation can “diffuse” throughout the network, or a portion of the network.
- At some point, the entire system may become quiescent:
  - No non-input actions enabled at any node.
  - No messages in channels.
- Termination Detection problem:
  - If A ever reaches a quiescent state then the starting node should eventually output “done”.
  - Otherwise, no one ever outputs “done”.
- To be solved by a monitoring algorithm Mon(A).
Dijkstra, Scholten Algorithm

- Augment A with extra pieces that construct and maintain a tree, rooted at the starting node, and including all the nodes currently active in A.
- Grows, shrinks, grows,…as nodes become active, quiescent, active,…
- Algorithm:
  - Execute A as usual, adding acks for all messages.
  - Messages of A treated like search messages in AsynchSpanningTree.
  - When a process receives an external input, it becomes the root, and begins executing A.
  - When any non-root process receives its first A message, it designates the sender as its parent in the tree, and begins participating in A.
  - Root process acks every message immediately.
  - Other processes ack all but the first message immediately.
  - Convergecast for termination:
    - If a non-root process finds its A-state quiescent and all its A-messages acked, then it cleans up: acks the first A-message, deletes all info about the termination protocol, becomes idle.
    - If it later receives another A message, it treats it like the first A message (defines a new parent, etc.), and resumes participating in A.
    - If root process finds A-state quiescent and all A-messages acked, reports done.
Example

- First, p1 gets awakened by an external A input, becomes the root, sends A messages to p2 and p4, p2 sends an A-message to p3, all set up parent pointers and start executing A.
- Next, p4 sends A message to p3, acked immediately.
- p4 sends A message to p1, acked immediately.
- p1, p2, p3, and p4 send A messages to each other for a while, everything gets acked immediately.
- Tree remains unchanged.
- Next, p2 and p3 quiesce locally; p3 cleans up, sends ack to p2, p2 receives ack, p2 cleans up, sends ack to p1.
- Next, p4 sends A messages to p2, p3, and p5, yielding a new tree:
- Etc.
Complexity

- **Messages:**
  - $2m$, where $m$ is the number of messages sent in $A$.

- **Time from quiescence of $A$ until output “done”:**
  - $O(md)$, where $d$ = upper bound on message delay, ignore local processing time
  - Time to clean up the spanning tree.

- **Bounds are most interesting if $m << n$.**
  - E.g., algorithms that involve only a limited computation in a small portion of a large network.
Outline

Distributed Operating System
Buddy Memory Management
Embedded Systems
Internet Of Things (IOT)
Software Engineering
Variable Size Partitions

- **Idea**
  - Allocate memory in small units
  - Give each job as many units as it needs

- **Key Challenges**
  - Keep track of free / allocated memory regions
  - Allocation policy to assign free regions to jobs
Maloc: Dynamic Allocation

**Idea**
- Allocate memory in small units
- Give each request as many units as it needs

**Key Challenges**
- Keep track of free / allocated memory regions
- Allocation policy to assign free regions to objects
Storage Allocation Policies

- **First fit**
  - Use first hole whose size is large enough
  - Rationale?

- **Best fit**
  - Use first exact size or smallest hole that is larger
  - Rationale?

- **Worst fit**
  - Use the largest available hole
  - Rationale?
Storage Allocation Policies

- **Best fit**
  - Produces the smallest leftover hole
  - Creates small holes that cannot be used

- **Worst Fit**
  - Produces the largest leftover hole
  - Difficult to run large programs

- **First Fit**
  - Creates average size holes

- First-fit and best-fit better than worst-fit in terms of speed and/or storage utilization
Fragmentation

- **Internal Fragmentation**
  - Allocated memory may be larger than requested memory
  - The extra memory is internal to a partition/block, but not being used

- **External Fragmentation**
  - Memory space exists to satisfy a request, but it is not contiguous
  - Reduce by compaction, buddy allocation
3. Deallocation Policies

- **Goals:** Given deallocation pointer:
  - Find size of the target object
  - Restore the memory to the “free list”
  - Minimize fragmentation

- **Common Mechanisms:**
  1. Object metadata to find size
  2. Binary buddy system to reduce external fragmentation
Binary Buddy Allocator

- Memory allocated using power-of-2 allocator
  - Satisfy requests in units of size power of 2
  - Request rounded up to next highest power of 2
  - When smaller allocation needed than is available, current chunk split into two buddies of next-lower power of 2
    - Continue until appropriate sized chunk available
Binary Buddy System

- **Approach**
  - Minimum allocation size = smallest frame
  - Use a *bitmap* to monitor frame use
  - Maintain *freelist* for each possible frame size
    - power of 2 frame sizes from min to max
  - Initially one block = entire buffer
  - If two neighboring frames ("buddies") are free, combine them and add to next larger freelist
Buddy System Example

128 Free
Buddy System Example

Process A requests 16

<table>
<thead>
<tr>
<th></th>
<th>128 Free</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>64 Free</td>
<td></td>
<td>64 Free</td>
</tr>
<tr>
<td>32 Free</td>
<td>32 Free</td>
<td>64 Free</td>
</tr>
<tr>
<td>16 A</td>
<td>16 Free</td>
<td>32 Free</td>
</tr>
</tbody>
</table>
Buddy System Example

Process B requests 32

16 A  |  16 Free  |  32 B  |  64 Free
Buddy System Example

Process C requests 8

<table>
<thead>
<tr>
<th>16 A</th>
<th>16 Free</th>
<th>32 B</th>
<th>64 Free</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 A</td>
<td>8</td>
<td>32 B</td>
<td>64 Free</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Buddy System Example

Process A exits

16 Free  8  8  32 B  64 Free
Buddy System Example

Process C exits

<table>
<thead>
<tr>
<th>16 Free</th>
<th>8</th>
<th>8</th>
<th>32 B</th>
<th>64 Free</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 Free</td>
<td>16 Free</td>
<td>32 B</td>
<td>64 Free</td>
<td></td>
</tr>
<tr>
<td>32 Free</td>
<td>32 B</td>
<td>64 Free</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Buddy System: Tradeoffs

- **Advantages**
  - Very fast search for Best Fit allocation policy
  - Fast compaction
  - Minimizes external fragmentation

- **Disadvantage**
  - Internal fragmentation when notalue 2^n request
Embedded Systems

- Introduction
- Scheduling in Embedded Systems
- Sorting in Embedded Systems
What is an Embedded System?

- Definition of an embedded computer system:
  - is a digital system.
  - uses a microprocessor (usually).
  - runs software for some or all of its functions.
  - frequently used as a controller.
Definition Embedded system

- An **embedded system** is a computer **system** with a dedicated function within a larger mechanical or electrical **system**, often with real-time computing constraints.
- It is **embedded** as part of a complete device often including hardware and mechanical parts.
- **Embedded systems** control many devices in common use today.
What an embedded system is NOT.

- Not a computer system that is used primarily for processing.
- Not a software system on a PC or Unix box.
- Not a traditional business or scientific application.
Examples of Embedded Systems

Medical instrument’s controls: CAT scanners, implanted heart monitors, etc.

Automotive systems: electronic dashboards, ABS brakes, transmission controls.

Controls for digital equipment: CD players, TV remote, programmable sprinklers, household appliances, etc.
Why ‘embedded’?

- Because the processor is ‘inside’ some other system.
- A microprocessor is ‘embedded’ into your TV, car, or appliance.
- The consumer does not think about performing processing,
- Considers running a machine or ‘making something work’.
Real-Time Systems

- Two types exist
  - Soft real-time
    - Tasks are performed as fast as possible
    - Late completion of jobs is undesirable but not fatal.
    - System performance degrades as more & more jobs miss deadlines
    - Example:
      - Online Databases
  - Hard real-time
    - Tasks have to be performed on time
    - Failure to meet deadlines is fatal
    - Example:
      - Flight Control System
Embedded & Real-Time Systems

- Execute tasks correctly and IN time.
- Systems with multiple tasks need scheduling
Definitions

- Ready time $r$ – task available
- Schedule
- Completed $C$
- Deadline $D$

From C.W. Mercer
Terms

- Periodic – aperiodic
- Fixed – variable computation time
- Predictable – unpredictable
- Preemptible – non preemptible
- Task → overall activity
- Jobs → individual computation
Terminologies (from J. A. Stankovic)

- **Job**
  - Each unit of work that is scheduled and executed by the system
- **Task**
  - A set of related jobs
  - For example, a periodic task Ti consists of jobs J1, J2, J3, … coming at every period
- **Release time**
  - Time instant at which a job becomes available for execution
  - It can be executed at any time at or after the release time
- **Deadline**
  - Time instant by which a job should be finished
  - Relative deadline: Maximum allowable response time
  - Absolute deadline = release time + relative deadline
Terminologies (from J. A. Stankovic)

- Periodic task $T_i$
  - Period $P_i$
  - Worst case execution time $C_i$
  - Relative deadline $D_i$

- Job $J_{ik}$
  - Absolute deadline = release time + relative deadline
  - Response time = finish time – release time

- Deadline miss if
  - Finish time > absolute deadline
  - Response time of $J_{ik} > D_i$
Real-Time Workload (from Insup Lee)

- Job (unit of work)
  - a computation, a file read, a message transmission, etc
- Attributes
  - Resources required to make progress
  - Timing parameters

![Diagram showing released, execution time, relative deadline, and absolute deadline](image-url)
Real-Time Task (from Insup Lee)

- **Task**: a sequence of similar jobs
- **Periodic task** $(p, e)$
  - Its jobs repeat regularly
  - Period $p = \text{inter-release time} \ (0 < p)$
  - Execution time $e = \text{maximum execution time} \ (0 < e < p)$
  - Utilization $U = e/p$
Periodic-Aperiodic Tasks

From C.W. Mercer
Aperiodic Non-predictable Task

Figure 4: Aperiodic, Unpredictable Task

From C.W. Mercer
Preemptible & non-preemptible Tasks

Figure 5: Preemptible Task

Figure 6: Non-preemptible Task
Commonly use real-time scheduling

- Clock-driven
- Weighted round-robin
- Priority driven
Scheduling Algorithms

- Clock Driven
  - All parameters about jobs (execution time/deadline) known in advance.
  - Schedule can be computed offline or at some regular time instances.
  - Minimal runtime overhead.
  - Not suitable for many applications.
Scheduling Algorithms

- **Weighted Round Robin**
  - Jobs scheduled in FIFO manner
  - Time quantum given to jobs is proportional to it’s weight
  - Example use: High speed switching network
    - QOS guarantee.
  - Not suitable for precedence constrained jobs.
    - Job A can run only after Job B. No point in giving time quantum to Job B before Job A.
Example task set

<table>
<thead>
<tr>
<th>Task</th>
<th>$t$</th>
<th>$p$</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>15 s</td>
<td>2 ms</td>
<td>smoke detector</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>15 s</td>
<td>2 ms</td>
<td>motion detector</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>40 ms</td>
<td>5 ms</td>
<td>digital audio intercom</td>
</tr>
<tr>
<td>$\tau_4$</td>
<td>40 ms</td>
<td>5 ms</td>
<td>digital telephone</td>
</tr>
<tr>
<td>$\tau_5$</td>
<td>20 ms</td>
<td>12 ms</td>
<td>CD quality audio</td>
</tr>
<tr>
<td>$\tau_6$</td>
<td>15 s</td>
<td>2 ms</td>
<td>toilet overflow detector</td>
</tr>
<tr>
<td>$\tau_7$</td>
<td>1 d</td>
<td>60 s</td>
<td>download newspaper</td>
</tr>
<tr>
<td>$\tau_8$</td>
<td>1 h</td>
<td>20 s</td>
<td>compile homework assignment</td>
</tr>
<tr>
<td>$\tau_9$</td>
<td>1 s</td>
<td>1 ms</td>
<td>keystroke</td>
</tr>
</tbody>
</table>

Table 1: Example Task Set

From C.W. Mercer
First in First out (FIFO)

Figure 12: FIFO Scheduling of Example Task Set

From C.W. Mercer
Round Robin scheduling

Figure 13: Round Robin Scheduling of Example Task Set

From C.W. Mercer
Cyclic executive

Figure 14: Timeline for Example Task Set (First Pass)

From C.W. Mercer
Cyclic executive

Figure 16: Timeline for Example Task Set (Third Pass)

Figure 17: Timeline for Example Task Set (Final Schedule)

From C.W. Mercer
Scheduling Algorithms

- Priority Scheduling
  - Processor never left idle when there are ready tasks
  - Processor allocated to processes according to priorities
- Priorities
  - Static - at design time
  - Dynamic - at runtime
Priority Scheduling

(from Dr. Chalermek Intanagonwiwat)

- Earliest Deadline First (EDF)
  - Process with earliest deadline given highest priority
- Least Slack Time First (LSF)
  - slack = relative deadline – execution left
- Rate Monotonic Scheduling (RMS)
  - For periodic tasks
  - Tasks priority inversely proportional to it’s period
Schedulability  *(from Insup Lee)*

- Property indicating whether a real-time system (a set of real-time tasks) can meet their deadlines

![Graph showing schedulability examples](image)
Real-Time Scheduling (from Insup Lee)

- Determines the order of real-time task executions
- Static-priority scheduling
- Dynamic-priority scheduling

![Diagram showing time slots and tasks with priorities (4,1), (5,2), and (7,2) at time points 5, 10, and 15 respectively.](image-url)
RM (Rate Monotonic)  
(from Insup Lee)

- Optimal static-priority scheduling
- It assigns priority according to period
- A task with a shorter period has a higher priority
- Executes a job with the shortest period
RM (Rate Monotonic)  *(from Insup Lee)*

- Executes a job with the shortest period
RM (Rate Monotonic)  
(from Insup Lee)

- Executes a job with the shortest period

Deadline Miss!
Response Time (from Insup Lee)

- Response time
- Duration from released time to finish time

\[ T_1(4,1) \]
\[ T_2(5,2) \]
\[ T_3(10,2) \]
Response Time (from Insup Lee)

- Response time
- Duration from released time to finish time

![Graph showing response times](image)
Response Time (from Insup Lee)

- Response Time ($r_i$) [Audsley et al., 1993]

- $\text{HP}(T_i):$ a set of higher-priority tasks than $T_i$
RM - Schedulability Analysis (from Insup Lee)

- Real-time system is schedulable under RM if and only if $r_i \leq p_i$ for all task $T_i(p_i,e_i)$

Joseph & Pandya,
“Finding response times in a real-time system”,
RM – Utilization Bound \textit{(from Insup Lee)}

- Real-time system is schedulable under RM if
  \[\sum U_i \leq n \left(2^{1/n} - 1\right)\]

Liu & Layland,

RM – Utilization Bound \( \text{(from Insup Lee)} \)

- Real-time system is schedulable under RM if 
  \[ \sum U_i \leq n \left( 2^{1/n} - 1 \right) \]

- Example: \( T_1(4,1), T_2(5,1), T_3(10,1), \)

  \[
  \sum U_i = \frac{1}{4} + \frac{1}{5} + \frac{1}{10} \\
  = 0.55 \\
  3 \left( 2^{1/3} - 1 \right) \approx 0.78
  
  \]

  Thus, \( \{T_1, T_2, T_3\} \) is schedulable under RM.
RM – Utilization Bound (from Insup Lee)

- Real-time system is schedulable under RM if
  \[ \sum U_i \leq n \left(2^{1/n} - 1\right) \]
Rate monotonic schedulable bound

<table>
<thead>
<tr>
<th>Task set size (n)</th>
<th>Schedulable bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>.828</td>
</tr>
<tr>
<td>3</td>
<td>.780</td>
</tr>
<tr>
<td>4</td>
<td>.757</td>
</tr>
<tr>
<td>5</td>
<td>.743</td>
</tr>
<tr>
<td>6</td>
<td>.735</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>∞</td>
<td>ln(2)</td>
</tr>
</tbody>
</table>

Table 2: Rate Monotonic Schedulable Bound

<table>
<thead>
<tr>
<th>Task</th>
<th>Priority</th>
<th>T</th>
<th>C</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>τ₁</td>
<td>0</td>
<td>20 ms</td>
<td>12 ms</td>
<td>CD quality audio</td>
</tr>
<tr>
<td>τ₂</td>
<td>1</td>
<td>40 ms</td>
<td>5 ms</td>
<td>digital audio intercom</td>
</tr>
<tr>
<td>τ₃</td>
<td>1</td>
<td>40 ms</td>
<td>5 ms</td>
<td>digital telephone</td>
</tr>
<tr>
<td>τ₄</td>
<td>2</td>
<td>15 s</td>
<td>2 ms</td>
<td>smoke detector</td>
</tr>
<tr>
<td>τ₅</td>
<td>2</td>
<td>15 s</td>
<td>2 ms</td>
<td>motion detector</td>
</tr>
<tr>
<td>τ₆</td>
<td>2</td>
<td>15 s</td>
<td>2 ms</td>
<td>toilet overflow detector</td>
</tr>
</tbody>
</table>

Table 3: Example Task Set with Rate Monotonic Priority Assignment

Table 4: Example Task Set Exact Criterion Results

\[ \sum_{i=1}^{n} \frac{C_i}{T_i} = \frac{12}{20} + \frac{5}{40} + \frac{5}{40} + \frac{2}{15000} + \frac{2}{15000} + \frac{2}{15000} = .8504 \]
EDF (Earliest Deadline First) (from Insup Lee)

- Optimal dynamic priority scheduling
- A task with a shorter deadline has a higher priority
- Executes a job with the earliest deadline
EDF (Earliest Deadline First) (from Insup Lee)

- Executes a job with the earliest deadline

T₁(4,1)  
T₂(5,2)  
T₃(7,2)
EDF (Earliest Deadline First) (from Insup Lee)

- Executes a job with the earliest deadline
EDF (Earliest Deadline First)  

- Executes a job with the earliest deadline
EDF (Earliest Deadline First) (from Insup Lee)

- Optimal scheduling algorithm
- If there is a schedule for a set of real-time tasks, EDF can schedule it.
Processor Demand Bound (from Insup Lee)

- Demand Bound Function: $dbf(t)$
  - the maximum processor demand by workload over any interval of length $t$
EDF - Schedulability Analysis *(from Insup Lee)*

- Real-time system is schedulable under EDF if and only if \( \text{dbf}(t) \leq t \) for all interval \( t \)

Baruah et al.

- Demand Bound Function : \( \text{dbf}(t) \)
  - the maximum processor demand by workload over any interval of length \( t \)
EDF – Utilization Bound (from Insup Lee)

- Real-time system is schedulable under EDF if and only if

\[ \sum U_i \leq 1 \]

Liu & Layland,

EDF – Overload Conditions (from Insup Lee)

- Domino effect during overload conditions
- Example: $T_1(4,3)$, $T_2(5,3)$, $T_3(6,3)$, $T_4(7,3)$

Deadline Miss!

Better schedules:
Least Slack Time First (LSF)

- slack = relative deadline – execution left
Embedded Systems

Introduction

Scheduling in Embedded Systems

Sorting in Embedded Systems
optimal sorting algorithm for embedded systems has these characteristics:

- It must sort in place.
- The algorithm must not be recursive.
- Its best, average and worst case running times should be of similar magnitude.
- Its code size should be commensurate with the problem.
- Its running time should increase linearly or logarithmically with the number of elements to be sorted.
- Its implementation must be ‘clean’ – i.e. free of breaks and returns in the middle of a loop.
Different sorting algorithms suitable for Embedded Systems

- Insertion sort
- Bubble sort
- Selection sort
Insertion Sort

- while some elements unsorted:
  - Using linear search, find the location in the sorted portion where the 1\(^{st}\) element of the unsorted portion should be inserted
  - Move all the elements after the insertion location up one position to make space for the new element

the fourth iteration of this loop is shown here
An insertion sort partitions the array into two regions
An insertion sort of an array of five integers
Insertion Sort Algorithm

public void insertionSort(Comparable[] arr) {
    for (int i = 1; i < arr.length; ++i) {
        Comparable temp = arr[i];
        int pos = i;
        // Shuffle up all sorted items > arr[i]
        while (pos > 0 &&
               arr[pos-1].compareTo(temp) > 0) {
            arr[pos] = arr[pos-1];
            pos--;
        } // end while
        // Insert the current item
        arr[pos] = temp;
    }
}
public void insertionSort(Comparable[] arr) {
    for (int i = 1; i < arr.length; ++i) {
        Comparable temp = arr[i];
        int pos = i;
        // Shuffle up all sorted items > arr[i]
        while (pos > 0 &&
            arr[pos-1].compareTo(temp) > 0) {
            arr[pos] = arr[pos-1];
            pos--;
        }
        // Insert the current item
        arr[pos] = temp;
    }
}
### Insertion Sort: Number of Comparisons

<table>
<thead>
<tr>
<th># of Sorted Elements</th>
<th>Best case</th>
<th>Worst case</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>n-1</td>
<td>1</td>
<td>n-1</td>
</tr>
<tr>
<td></td>
<td>n-1</td>
<td>n(n-1)/2</td>
</tr>
</tbody>
</table>

Remark: we only count comparisons of elements in the array.
Insertion Sort: Cost Function

- 1 operation to initialize the outer loop
- The outer loop is evaluated $n-1$ times
  - 5 instructions (including outer loop comparison and increment)
  - Total cost of the outer loop: $5(n-1)$
- How many times the inner loop is evaluated is affected by the state of the array to be sorted
- Best case: the array is already completely sorted so no “shifting” of array elements is required.
  - We only test the condition of the inner loop once (2 operations = 1 comparison + 1 element comparison), and the body is never executed
  - Requires $2(n-1)$ operations.
Insertion Sort: Cost Function

- Worst case: the array is sorted in reverse order (so each item has to be moved to the front of the array)
  - In the \(i\)-th iteration of the outer loop, the inner loop will perform \(4i+1\) operations
  - Therefore, the total cost of the inner loop will be \(2n(n-1)+n-1\)
- Time cost:
  - Best case: \(7(n-1)\)
  - Worst case: \(5(n-1)+2n(n-1)+n-1\)
- What about the number of moves?
  - Best case: \(2(n-1)\) moves
  - Worst case: \(2(n-1)+n(n-1)/2\)
Insertion Sort: Average Case

- Is it closer to the best case \((n \text{ comparisons})\)?
- The worst case \((n \times (n-1) / 2) \text{ comparisons}\)?
- It turns out that when random data is sorted, insertion sort is usually closer to the worst case
  - Around \(n \times (n-1) / 4\) comparisons
  - Calculating the average number of comparisons more exactly would require us to state assumptions about what the “average” input data set looked like
  - This would, for example, necessitate discussion of how items were distributed over the array
- Exact calculation of the number of operations required to perform even simple algorithms can be challenging
  (for instance, assume that each initial order of elements has the same probability to occur)
Bubble Sort

- Simplest sorting algorithm
- **Idea:**
  1. Set flag = false
  2. Traverse the array and compare pairs of two consecutive elements
     - 1.1 If \( E_1 \leq E_2 \) -> OK (do nothing)
     - 1.2 If \( E_1 > E_2 \) then Swap(\( E_1, E_2 \)) and set flag = true
  3. repeat 1. and 2. while flag=true.
**Bubble Sort**

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
|1  | 1 | 23| 2 | 56| 9 | 8 | 10|=100
|2  | 1 | 2 | 23| 56| 9 | 8 | 10|=100
|3  | 1 | 2 | 23| 9 | 56| 8 | 10|=100
|4  | 1 | 2 | 23| 9 | 8 | 56| 10|=100
|5  | 1 | 2 | 23| 9 | 8 | 10| 56|=100

---- finish the first traversal ----

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
|1  | 1 | 2 | 23| 9 | 8 | 10| 56|=100
|2  | 1 | 2 | 9 | 23| 8 | 10| 56|=100
|3  | 1 | 2 | 9 | 8 | 23| 10| 56|=100
|4  | 1 | 2 | 9 | 8 | 10| 23| 56|=100

---- finish the second traversal ----

...
Bubble Sort

```java
public void bubbleSort (Comparable[] arr) {
    boolean isSorted = false;
    while (!isSorted) {
        isSorted = true;
        for (i = 0; i<arr.length-1; i++)
            if (arr[i].compareTo(arr[i+1]) > 0) {
                Comparable tmp = arr[i];
                arr[i] = arr[i+1];
                arr[i+1] = tmp;
                isSorted = false;
            }
    }
}
```
Bubble Sort: analysis

- After the first traversal (iteration of the main loop) – the maximum element is moved to its place (the end of array)
- After the $i$-th traversal – largest $i$ elements are in their places

- Time cost, number of comparisons, number of moves -> Assignment 4
Internet Of Things

**Introduction**

- Algorithms in routing
- Clustering
- Context management
- Identity management and trust management
Evaluation Of Internet
Evaluation Of Internet
The Internet of Things allows people and things to be connected anytime, anyplace, with anything and anyone, ideally using any path/network and any service.
Internet Of Things (IoT)

The Internet of Things (IoT), also called Internet of Everything or Network of Everything, is the network of physical objects or "things" embedded with electronics, software, sensors, and connectivity to enable objects to exchange data with the production, operator and/or other connected devices based on the infrastructure of International Telecommunication Union's Global Standards Initiative.

The Internet of Things allows objects to be sensed and controlled remotely across existing network infrastructure, creating opportunities for more direct integration between the physical world and computer-based systems, and resulting in improved efficiency, accuracy and economic benefit.
Internet Of Things (IoT)

Each thing is uniquely identifiable through its embedded computing system but is able to interoperate within the existing Internet infrastructure.

Typically, IoT is expected to offer advanced connectivity of devices, systems, and services that goes beyond machine-to-machine communications (M2M) and covers a variety of protocols, domains, and applications.
Elements of Internet Of Things (IoT)

- **Sensors**
- Storage
- Cloud based Capture and Computing
- Delivery of Information (Devices, Communication and Processing)
- Identification
- Localization and Tracking
IoT Architecture

Three-Layer Arch

Technology
- WPAN, WLAN
- WMAN, WWAN

Protocol
- Non-IP Based (Zigbee)
- IP-Based (IPV6)
Application of IoT

- Smart Cities
- Smart Environment
- Smart Water
- Industrial Control
- Smart Agriculture
- Smart Animal Farming
- Domotic & Home Automation
- eHealth
Layered Structure of Sensor Network
Internet Of Things

Introduction

Algorithms in routing

clustering

context management

identity management and trust management
IoT routing

- LLN – Low Power and Lossy Networks
  - Connections are by default unreliable, low transmission speed, high packet loss rate
    - Small capacity antennas, unfriendly environment (rain, snow, frost), interferences, mobility
  - The goal is energy efficiency, not communication efficiency
    - No problem is no continuous connection or packets are lost
    - Unattended operation for years – self-configuration, self-management
IoT routing

- **LLN**, link failures are usual, but transient
  - If we would react to them, the network would become unstable (too much control plane overhead)

- In traditional networks big data traffic (video, VoIP), no possibility for buffering if a link is broken

- In **LLNs** low data traffic, a transient loss can be easily handled with buffering or local redirection
  - No need to reconfigure the entire topology
  - In traditional networks static metrics, to ensure path stability

- In **LLNs** **dynamic metrics**, changing in time
  - The networks can adapt
LLN vs. WSN

- A wireless sensor network is a special type of LLN

<table>
<thead>
<tr>
<th>WSN</th>
<th>LLN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Homogeneous network, similar sensors</td>
<td>Heterogeneous network, different nodes</td>
</tr>
<tr>
<td>Devices deployed with a specific goal, based on the needs of a specific application</td>
<td>Devices with different tasks, cooperating to find a gateway to the Internet</td>
</tr>
<tr>
<td>P2MP or MP2P communication (between sink and sensors)</td>
<td>P2MP, MP2P or P2P communication (between two IoT devices)</td>
</tr>
<tr>
<td>Usually no IP support</td>
<td>IP based communication</td>
</tr>
</tbody>
</table>
RPL – IPv6 Routing Protocol for Low Power Lossy Networks

- IPv6 distance vector routing protocol
  - Builds a DODAG-ot (Destination Oriented Directed Acyclic Graph)
  - Using an objective function (OF) that takes into account several metrics and/or constraints
  - Several different OFs might be used in the same time, each building its own DODAG
    E.g., 1) The smallest ETX*, but only over nodes supporting encryption
    2) The lowest delay, but only over nodes that have solar panels

* ETX – Expected Transmission Count
  - Shows the quality of a radio connection – how many times should I probably send a packet to be sure that it is received. Value between 1 and $\infty$, expected value based on past experience
Building a DODAG

- Starts at the root node – LBR (LLN Border Router)
  - Gateway to the Internet
  - Many LBRs can exist in the same network

- New ICMPv6 control messages for RPL
  - DIO – DODAG Information Object
  - DAO – DODAG Destination Advertisement Object
  - DIS – DODAG Information Solicitation
Building the DODAG

- Edges – LLN connections
- Values – e.g. ETX
  - Can change in time
  - Using an average value some stability can be ensured
- Goal (OF) – minimize ETX
Building the DODAG

- The LBR sends a DIO message to its neighbors
  - Link local multicast
- Nodes A, B, C receive and handle it
  - Many DIOs can be received in the same time, from different LBRs
  - Based on the OF and other criteria, they decide to join the DODAG or not
  - If yes, the LBR is marked as parent
Building the DODAG

- The DIO timer of node C expires
- C sends a multicast DIO message to its neighbors
- The LBR ignores it, as it come from a higher ranked node
  - Rank(LBR) = 0, Rank(C) = 1
  - Needed to avoid loops
- B marks C as an alternative parent
- E joins the DODAG, marks C as a parent
Building the DODAG

- The DIO timer of node A expires
- Sends a multicast DIO message to its neighbors
- The LBR ignores it
- B marks A as an alternative parent
- As the OF (ETX) value is better if B connects through A, B deletes the LBR and C from its parents
Building the DODAG

- The construction of the DODAG is continuous

- The DODAG is continuously maintained
MP2P (Multi-Point to Point) traffic

- MP2P traffic along the DODAG, from each node to the LBR
  - UPWARD routing
- The LBR connects to the Internet
- A node can participate in parallel in several DODAGs
P2MP (Point to Multi-Point) traffic

- **DOWNWARD routing**
  - Routing information has to be built

- **DAO (DODAG Destination Advertisement) messages**
  - If a node joins a DODAG, sends a DAO to its parents
  - Can be initiated by the LBR as well, or any intermediate node in the DODAG
    - Marked in the DIO message going downwards
    - The DAO timers (DelayDAO) are set so as to expire first at higher ranked nodes in the DODAG
  - The advertise in the DAO message the network prefix reachable through them
P2MP (Point to Multi-Point) traffic

- LLN nodes sometimes are not able to store routing entries

- If yes – **storing mode**
  - If possible, it aggregates the prefix of its children with its own prefix, and send this aggregated value forward

- If not – **non-storing mode**
  - Source-routing: the DAO message contains the path

- Either storing mode or non-storing, the hybrid solution was not standardized
  - Even if it was included in the first drafts

- Advantages end drawbacks
  - In storing mode routing entries have to be stored, but short messages
  - In non-storing mode nothing has to be stored but long messages, increased energy consumption
P2P (Point-to-Point) traffic

- Between any two S (Source) and D (Destination) nodes
  - If non-storing mode:
    - From S, we have to go towards the LBR along the DODAG
    - Then source routing from the LBR to D
  - If storing mode:
    - Shortcut at the first common ancestor
DODAG maintenance

- **Grounded DODAG**
  - A DODAG that corresponds to an OF and the given constraints
- **Floating DODAG**
  - A DODAG that does not meet the necessary criteria
    - Transient state, marked in the DIO message by setting the G (Grounded) bit
  - What happens if the connection breaks between B and D?
    - D deletes B from its parents
    - As D does not have other parents, it becomes the root of a floating DODAG
D sends a multicast DIO message to its neighbors, letting them know about the change.

- Node I has an alternative parent (E), still connected through it to the grounded DODAG.

- As D is not part of the DODAG anymore, I deletes it from its parents.
F does not have an alternative parent, so it stays in the floating DODAG of D.

F sends a multicast DIO message to its neighbors.

G and H do the same and follow F in the floating DODAG of D.
A floating DODAG was formed, with D being the root
They delete all the paths related to LBR
The floating DODAG tries to rejoin the grounded DODAG...
I sends a multicast DIO message

D receives it, learns that it could join the LBR DODAG through I

D starts a timer related to node I

- The setting of the timer depends on the rank of node I
- D wants to be as close to LBR as possible
- Suppose that a radio link is established between A and F
- A sends a multicast DIO message
- F receives it, starts a timer related to node A
The timer of node F expires
F joins the grounded DODAG through A, deletes D from its parents
F sends a multicast DIO message
G and H join the grounded DODAG through F
DODAG maintenance

- D sees that it could join the grounded DODAG through F
- D starts a timer related to F, besides the already running time related to I
The timer related to F expires first.

D joins the grounded DODAG through F.

The floating DODAG disappears, the problem was handled without forming cycles.
Internet Of Things

- Introduction
- Algorithms in routing
- Clustering
- Context management
- Identity management and trust management
Clustering In IoT

Data clustering refers to grouping of data based on specific features and their values. It is unsupervised machine learning.

Data clustering can be applied however on a new data set without really knowing much about it in advance.

The number of clusters is usually given as an input (e.g., the number of activity levels the motion data should be divided into), but there are also algorithms that can automatically categorize data in the most optimal way.

Data clustering maybe not be used directly in IoT applications, but in many cases it can be an intermediate step for identifying patterns from the collected data.
Cluster

- Partitioning Methods
- Hierarchical Methods
Partitioning Algorithms: Basic Concept

- **Partitioning method**: Construct a partition of a database \( D \) of \( n \) objects into a set of \( k \) clusters
- Given a \( k \), find a partition of \( k \) clusters that optimizes the chosen partitioning criterion
  - \( k\)-means: Each cluster is represented by the center of the cluster
  - \( k\)-medoids or PAM (Partition around medoids): Each cluster is represented by one of the objects in the cluster
The *K-Means* Clustering Method

- Given $k$, the *k-means* algorithm is implemented in 4 steps:
  - Partition objects into $k$ nonempty subsets
  - Compute seed points as the centroids of the clusters of the current partition. The centroid is the center (mean point) of the cluster.
  - Assign each object to the cluster with the nearest seed point.
  - Go back to Step 2, stop when no more new assignment.
The *K-Means* Clustering Method

- **Example**

![Diagram](image)
The *K-Medoids* Clustering Method

- Find *representative* objects, called *medoids*, in clusters
- *PAM* (Partitioning Around Medoids)
  - starts from an initial set of medoids and iteratively replaces one of the medoids by one of the non-medoids if it improves the total distance of the resulting clustering
- *PAM* works effectively for small data sets, but does not scale well for large data sets
- *CLARA*
- *CLARANS*
Cluster

- Partitioning Methods
- Hierarchical Methods
Hierarchical Clustering

- Use distance matrix as clustering criteria. This method does not require the number of clusters $k$ as an input, but needs a termination condition.

![Diagram showing hierarchical clustering steps](image_url)
AGNES (Agglomerative Nesting)

- Introduced in Kaufmann and Rousseeuw (1990)
- Implemented in statistical analysis packages, e.g., Splus
- Use the Single-Link method and the dissimilarity matrix.
- Merge nodes that have the least dissimilarity
- Go on in a non-descending fashion
- Eventually all nodes belong to the same cluster
DIANA (Divisive Analysis)

- Introduced in Kaufmann and Rousseeuw (1990)
- Implemented in statistical analysis packages, e.g., Splus
- Inverse order of AGNES
- Eventually each node forms a cluster on its own
More on Hierarchical Clustering Methods

- Major weakness of agglomerative clustering methods
  - do not scale well: time complexity of at least $O(n^2)$, where $n$ is the number of total objects
- Integration of hierarchical with distance-based clustering
  - BIRCH: uses CF-tree and incrementally adjusts the quality of sub-clusters
  - CURE: selects well-scattered points from the cluster and then shrinks them towards the center of the cluster by a specified fraction
  - CHAMELEON: hierarchical clustering using dynamic modeling
Internet Of Things

Introduction

Algorithms in routing

clustering

context management

identity management and trust management
What is meant by “Context”

“Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application.”

Synonyms: “Circumstance, situation, phase, position, posture, attitude, place, point, status surroundings, environment, location.”
Context Management

Context Management allows users to choose a subject once in one application, and have all other applications containing information on that same subject 'tune' to the data they contain, thus obviating the need to redundantly select the subject in the varying applications.
Context management Cont...

Context information is represented by values assigned to attributes that characterize the entities relevant to your application.

Context is about entities and their attributes.
Entities and Attributes

**Entity**: Object in the real world

**Attributes**: Describe each entity

**Entity** = Customer

- Customer ID
- First Name
- Surname
- Date of birth
- Address
- Phone no.

These are the ‘attributes’ for the entity ‘customer’
If your application is about weather measurement based on sensors deployed in the streets and parks of your city.

**The entities**: sensors

**The attributes**: are the different weather parameters a sensor may measure (such as temperature, humidity, luminance, atmospheric pressure, etc.) along with other operational parameters (such as the sensor location, battery level, etc.)
Values of context management

- **Simplicity:**
  - Everything is about entities and attributes
  - No complex modeling needed
  - No complex data relationships or complicated SQL statements to get your data
  - Modeling your application in terms of entities and attributes is generally easy

- **Flexibility:**
  - Context is a rather generic concept, so it is suited for many applications
Architecture of Context Management

Context Manager

- Context Broker
  - Query Manager
  - Subscription Manager

- Context Provider
  - Physical Sensor
  - Virtual Sensor

- Context Consumer
  - Context Subscriber

- Working Context
- Context History
Actors in context management

- **Context Producers**
  - Context producers are the sources of context, the ones that create or update context information.
  - Sensor is context producers

- **Context Consumers**
  - They are the sinks for context, the ones that receive context information
Context management - single sign on

- Context management can be used in conjunction with single sign on (SSO) to further improve workflow efficiency.
- Single sign-on (also known as Enterprise Single Sign On or ESSO) is a property of access control of multiple, related, but independent software systems.
- Single sign on allows the user to enter one username/password, which then grants access to multiple applications within the system.
- SSO may grant access to all applications, or if additional layers of security are deemed necessary
Context management- **CCOW**

- CCOW is a vendor independent **HL7** standard that enables context management when switching between different clinical applications.
- CCOW works for both client-server and web-based applications.
- The goal of CCOW is to create a clinical user’s experience of interacting with a single system, when the user is actually using multiple, independent applications from many different systems.
- Contexts that can be shared by CCOW include patient (name and identifying numbers), user, and encounter.
- CCOW context management architecture has three main components
  - Clinical applications (context participants)
  - Context manager to coordinate and synchronize the applications
  - Mapping agents to map equivalent identifiers, allowing applications to interoperate without sharing the same identification method for a particular patient or user.
Context management - **CCOW**

- **Advantages of CCOW**
  - Greater purchasing flexibility.
  - Rapid, unified access to patient data from different applications helps insure an efficient workflow environment for end-users.
  - Single sign-on management improves workflow efficiency
  - Context oriented workflow improves both efficiency and safety.

- **Disadvantages of CCOW**
  - An organization may have applications already in use that are not CCOW compliant.
  - Requires server side crosswalk table for application login password adjudication
Software Engineering

Boyer Moore string Matching Algorithm

KMP Algorithm
Boyer-Moore algorithm

- The Boyer-Moore algorithm is considered the most efficient string-matching algorithm.
- The algorithm scans the characters of the pattern from right to left beginning with the rightmost character.
- During the testing of a possible placement of pattern P against text T, a mismatch of text character $T[i] = c$ with the corresponding pattern character $P[j]$ is handled as follows:
  1. If $c$ is not contained anywhere in P, then shift the pattern P completely past $T[i]$.
  2. Otherwise, shift P until an occurrence of character $c$ in P gets aligned with $T[i]$.
- This technique is likely to avoid lots of needless comparisons by significantly shifting pattern relative to text.
We define a function last(c) that takes a character c from the alphabet and specifies how far may shift the pattern P if a character equal to c is found in the text that does not match the pattern.

For example consider:

- T: 0 1 2 3 4 5 6 7 8 9
  a b a c a a b a c c
- P: a b a c a a b
  0 1 2 3 4 5
Last(c)

- last(a) is the index of the last (rightmost) occurrence of 'a' in P, which is 4.
- last(c) is the index of the last occurrence of c in P, which is 3
- 'd' does not exist in the pattern there we have last(d) = -1.
- The complete last(c) function

<table>
<thead>
<tr>
<th>c</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>last(c)</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>-1</td>
</tr>
</tbody>
</table>
Algorithm Boyer-Moore

- **BOYER_MOORE.Matcher** (T, P)
- **Input:** Text with n characters and Pattern with m characters
- **Output:** Index of the first substring of T matching P

**Compute function last**

\[i \leftarrow m - 1\]
\[j \leftarrow m - 1\]

Repeat

- If \(P[j] = T[i]\) then
  - if \(j = 0\) then
    - return \(i\)  // we have a match
  - else
    - \(i \leftarrow i - 1\)
    - \(j \leftarrow j - 1\)
  - else
    - \(i \leftarrow i + m - \text{Min}(j, 1 + \text{last}[T[i]])\)
    - \(j \leftarrow m - 1\)

until \(i > n - 1\)

Return "no match"
Analysis

- The computation of the last function takes $O(m + |\Sigma|)$ time and actual search takes $O(mn)$ time. Therefore the worst case running time of Boyer-Moore algorithm is $O(nm + |\Sigma|)$. Implies that the worst-case running time is quadratic, in case of $n = m$, the same as the naïve algorithm.
Software Engineering

Boyer Moore string Matching Algorithm

KMP Algorithm
The Knuth-Morris-Pratt Algorithm

Knuth, Morris and Pratt proposed a linear time algorithm for the string matching problem.

A matching time of $O(n)$ is achieved by avoiding comparisons with elements of ‘S’ that have previously been involved in comparison with some element of the pattern ‘p’ to be matched. i.e., backtracking on the string ‘S’ never occurs.
Components of KMP algorithm

- **The prefix function, \( \Pi \)**
  The prefix function, \( \Pi \) for a pattern encapsulates knowledge about how the pattern matches against shifts of itself. This information can be used to avoid useless shifts of the pattern ‘\( p \)’. In other words, this enables avoiding backtracking on the string ‘\( S \)’.

- **The KMP Matcher**
  With string ‘\( S \)’, pattern ‘\( p \)’ and prefix function ‘\( \Pi \)’ as inputs, finds the occurrence of ‘\( p \)’ in ‘\( S \)’ and returns the number of shifts of ‘\( p \)’ after which occurrence is found.
The prefix function, $\Pi$

Following pseudocode computes the prefix function, $\Pi$:

**Compute-Prefix-Function (p)**

1. $m \leftarrow \text{length}[p]$ // ‘p’ pattern to be matched
2. $\Pi[1] \leftarrow 0$
3. $k \leftarrow 0$
4. for $q \leftarrow 2$ to $m$
5.   do while $k > 0$ and $p[k+1] \neq p[q]$
6.     do $k \leftarrow \Pi[k]$
7.     if $p[k+1] = p[q]$
8.         then $k \leftarrow k + 1$
9.     $\Pi[q] \leftarrow k$
10. return $\Pi$
Example: compute $\Pi$ for the pattern ‘p’ below:

\[
\begin{array}{cccccccc}
p & a & b & a & b & a & c & a \\
\end{array}
\]

Initially: $m = \text{length}[p] = 7$
- $\Pi[1] = 0$
- $k = 0$

Step 1: $q = 2$, $k = 0$
- $\Pi[2] = 0$

Step 2: $q = 3$, $k = 0$
- $\Pi[3] = 1$

Step 3: $q = 4$, $k = 1$
- $\Pi[4] = 2$
Step 4: q = 5, k = 2
\[ \Pi[5] = 3 \]

Step 5: q = 6, k = 3
\[ \Pi[6] = 1 \]

Step 6: q = 7, k = 1
\[ \Pi[7] = 1 \]

After iterating 6 times, the prefix function computation is complete:
The **KMP Matcher**

The KMP Matcher, with pattern ‘p’, string ‘S’ and prefix function ‘Π’ as input, finds a match of p in S.

**KMP-Matcher(S,p)**

\[
\begin{align*}
\text{n} & \leftarrow \text{length}[S] \\
\text{m} & \leftarrow \text{length}[p] \\
\Pi & \leftarrow \text{Compute-Prefix-Function}(p) \\
\text{q} & \leftarrow 0 \quad // \text{number of characters matched} \\
\text{for} \ i \leftarrow 1 \ \text{to} \ n & \quad // \text{scan S from left to right} \\
\text{do while} \ q > 0 \ \text{and} \ p[q+1] \neq S[i] & \\
\text{\quad do} \ q \leftarrow \Pi[q] \quad // \text{next character does not match} \\
\text{\quad if} \ p[q+1] = S[i] & \\
\text{\quad \quad then} \ q \leftarrow q + 1 \quad // \text{next character matches} \\
\text{\quad if} \ q = m & \quad // \text{is all of p matched?} \\
\text{\quad \quad then} \ \text{print} \ "\text{Pattern occurs with shift}" \ i - m \\
\text{\quad q} & \leftarrow \Pi[q] \quad // \text{look for the next match}
\end{align*}
\]
**Illustration:** given a String ‘S’ and pattern ‘p’ as follows:

\[
\begin{array}{cccccccc}
\text{S} & b & a & c & b & a & b & a & b & a & b & a & b & a & c & c & a & c \\
\end{array}
\]

\[
\begin{array}{cccccccc}
\text{p} & a & b & a & b & a & c & a \\
\end{array}
\]

Let us execute the KMP algorithm to find whether ‘p’ occurs in ‘S’.

*For ‘p’ the prefix function, \( \Pi \) was computed previously and is as follows:*

\[
\begin{array}{cccccccc}
q & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
p & a & b & A & b & a & c & a \\
\Pi & 0 & 0 & 1 & 2 & 3 & 1 & 1 \\
\end{array}
\]
Initially: \( n = \text{size of } S = 15; \)
\[ m = \text{size of } p = 7 \]

Step 1: \( i = 1, \ q = 0 \)
comparing \( p[1] \) with \( S[1] \)

\[
S = \text{b a c b a b a b a b a b a c a a a b}
\]
\[
p = \text{a b a b a b a c a a a}
\]

\( P[1] \) does not match with \( S[1] \). ‘\( p \)’ will be shifted one position to the right.

Step 2: \( i = 2, \ q = 0 \)

\[
S = \text{b a c b a b a b a b a b a c a a a b}
\]
\[
p = \text{a b a b a b a c a a a}
\]

\( P[1] \) matches \( S[2] \). Since there is a match, \( p \) is not shifted.
Step 3: $i = 3, q = 1$


```
S: bacbababababcabaab

p: ababababaca
```

Backtracking on $p$, comparing $p[1]$ and $S[3]$

Step 4: $i = 4, q = 0$


```
S: bacbababababcabaab

p: ababababaca
```

Step 5: $i = 5, q = 0$


```
S: bacbababababcabaab

p: ababababaca
```
Step 6: $i = 6$, $q = 1$

Step 7: $i = 7$, $q = 2$

Step 8: $i = 8$, $q = 3$
Step 9: $i = 9, q = 4$


Step 10: $i = 10, q = 5$


Step 11: $i = 11, q = 4$

Step 12: $i = 12$, $q = 5$

Step 13: $i = 13$, $q = 6$

Pattern ‘p’ has been found to completely occur in string ‘S’. The total number of shifts that took place for the match to be found are: $i - m = 13 - 7 = 6$ shifts.
Running - time analysis

Compute-Prefix-Function ($\Pi$)-
The running time of compute prefix function is $\Theta(m)$.

KMP Matcher-
The running time of matching function is $\Theta(n)$. 