EE3980 Algorithms

演算法

EE/NTHU

Mar. 3, 2020

Algorithms

- An example
  - Brute-force approach
  - Improving the performance
  - Taking advantage of input sparsity
  - Improving worst-case performance
  - Average-case performance
- Course information

Programming and Algorithm

- Programming uses computer (or any mechanism) to solve problems:
  Given a set of input, perform necessary processing to find the right output.
- An example
  - Problem: find the number of 1s in a bit string.
  - Input: n bit binary string, $B = b_n b_{n-1} \cdots b_2 b_1,\ b_i \in \{0, 1\}$, $1 \le i \le n$.
  - Output: c is the number of 1s in B.
  - Example: an instance of the problem is
    - Input: $n = 8,\ B = 11010001$.
    - Output: $c = 4$.
  - A brute-force approach can be used to solve this problem.

Brute-force Approach – CountOne_A

Algorithm 0.0.1.

```
   // Count the number of 1s in a bit string B.
   // Input: B = b_n b_{n-1} ··· b_2 b_1, int n > 0
   // Output: c, number of 1s in B.
1 Algorithm CountOne_A(B, n)
2 {
3     c := 0; // Init c to 0
4     for i := 1 to n step 1 do
5         c := c + b_i; // Count every bit.
6     return c;
7 }
```

- Lines 4-5, loop is executed n times
  - Loop body consists of 1 operation: addition
  - Addition is executed n times.
- A straightforward brute force approach.
  - Efficiency can be improved.

Modified Approach – CountOne_B

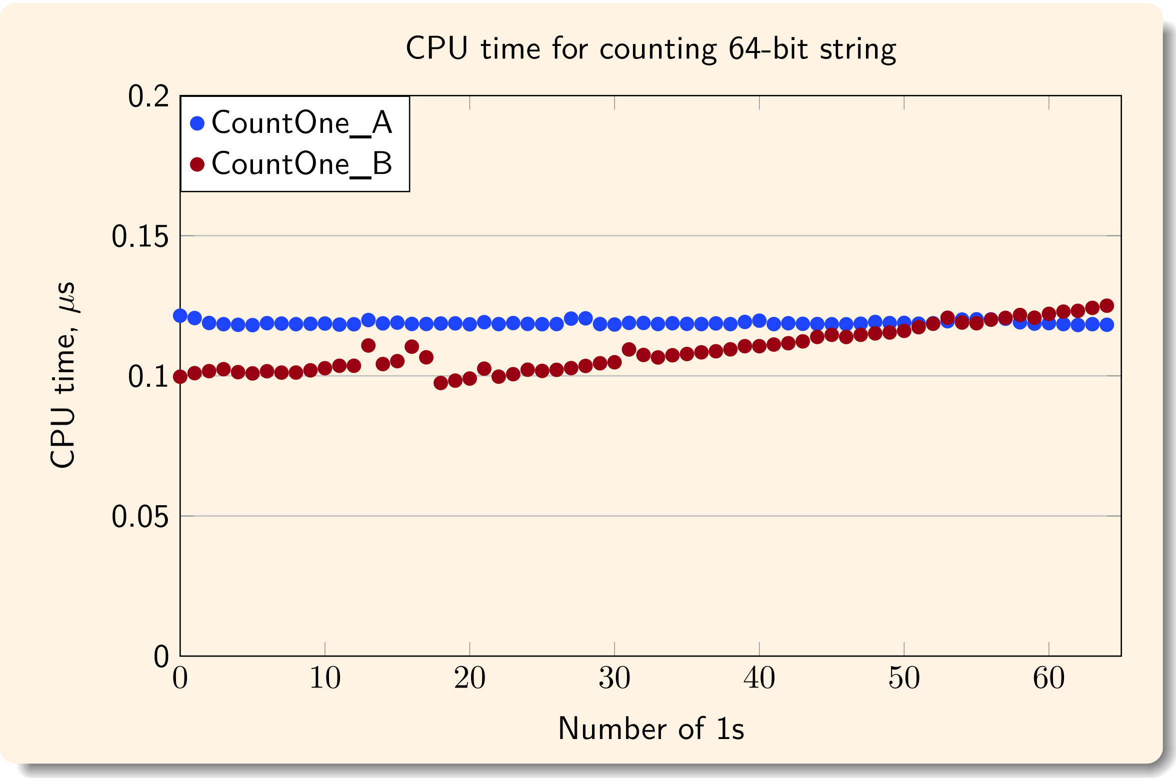
- The preceding algorithm can be modified as the following.

Algorithm 0.0.2.

```
  // Count the number of 1s in a bit string B.
  // Input: B = b_n b_{n-1} ··· b_2 b_1, int n > 0
  // Output: c, number of 1s in B.
1 Algorithm CountOne_B(B, n)
2 {
3     c := 0; // Init c to 0
4     for i := 1 to n step 1 do
5         if (b_i = 1) c := c + 1; // Add only necessary.
6     return c;
7 }
```

- Lines 4-5, loop is still executed n times
  - Loop body consists of 2 operations: equality check and addition.
  - Equality check executed n times and addition c times.
  - If addition takes more time than equality check, then CPU time can be reduced.
- This is still a brute-force approach.

Comparing Two Approaches



- CountOne_B is, indeed, faster for smaller c.
  - But for large c, it can be slower – worst-case scenario.

A Better Approach – CountOne_C

- A more efficient approach

Algorithm 0.0.3.

```
   // Count the number of 1s in a bit string B.
   // Input: B = b_n b_{n-1} ··· b_2 b_1, int n > 0
   // Output: c, number of 1s in B.
1 Algorithm CountOne_C(B, n)
2 {
3       c := 0; // Init c to 0
4       while (B ≠ 0) do {
5             c := c + 1;
6             B := B & (B − 1); // Remove one 1 in B; & is bit-wise AND
7       }
8       return c;
9 }
```

- Lines 4-7, loop is executed c times, $c \leq n$.
  - Loop body consists of 3 operations
    - 1 addition, 1 subtraction, 1 bitwise AND
  - If B is sparse, few 1s, then this algorithm is very efficient.
  - If B is mostly ones, then it might be slower than the preceding algorithms.

Algorithm CountOne_C Example

- Algorithm CountOne_C execution example
  $B = 1101,0001$

Iteration 1

c:	1
B:	1101,0001
$B-1$:	1101,0000
$B \,\&\, (B-1)$:	1101,0000

Iteration 2

c:	2
B:	1101,0000
$B-1$:	1100,1111
$B \,\&\, (B-1)$:	1100,0000

Iteration 3

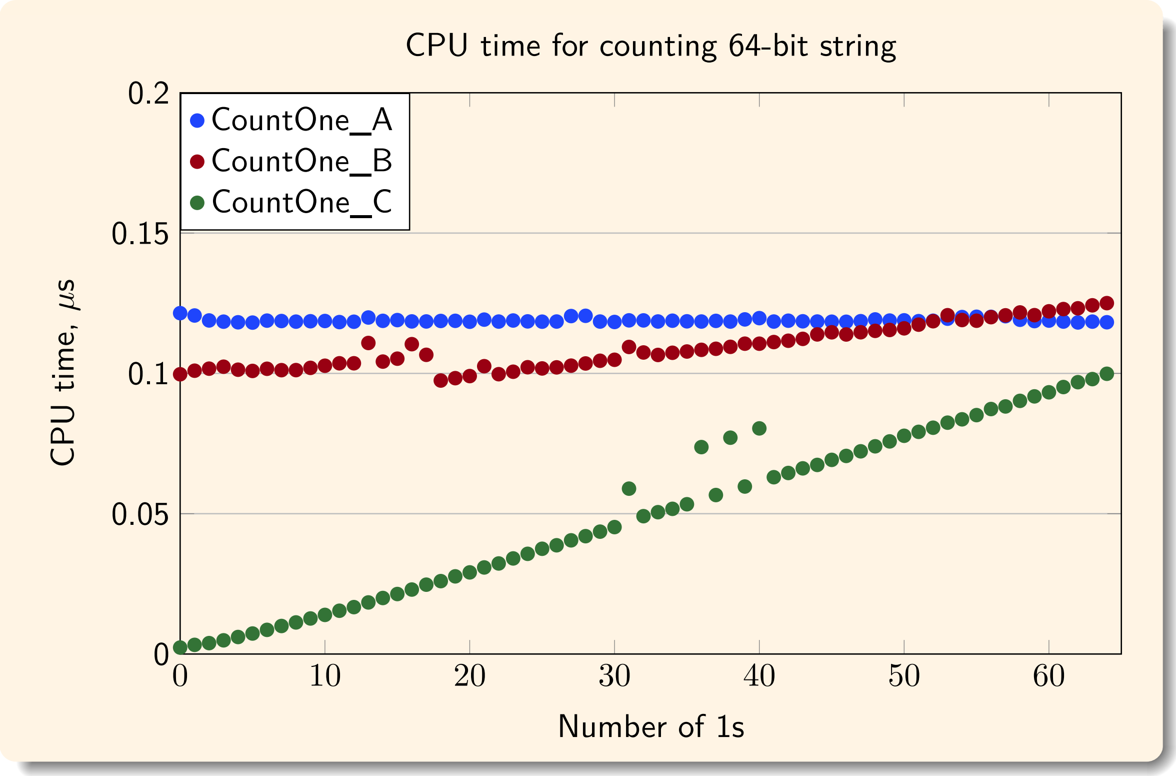
c:	3
B:	1100,0000
$B-1$:	1011,1111
$B \,\&\, (B-1)$:	1000,0000

Iteration 4

c:	4
B:	1000,0000
$B-1$:	0111,1111
$B \,\&\, (B-1)$:	0000,0000

- Each iteration of the loop eliminates one 1 in B.

Comparisons of First 3 Approaches



- CountOne_C is shown to be the most efficient, especially for small c.
- On some computers, the worst case $(c = n)$ CPU time is larger than the first two approaches.

Counting 1s in a Bit String – Algorithm D

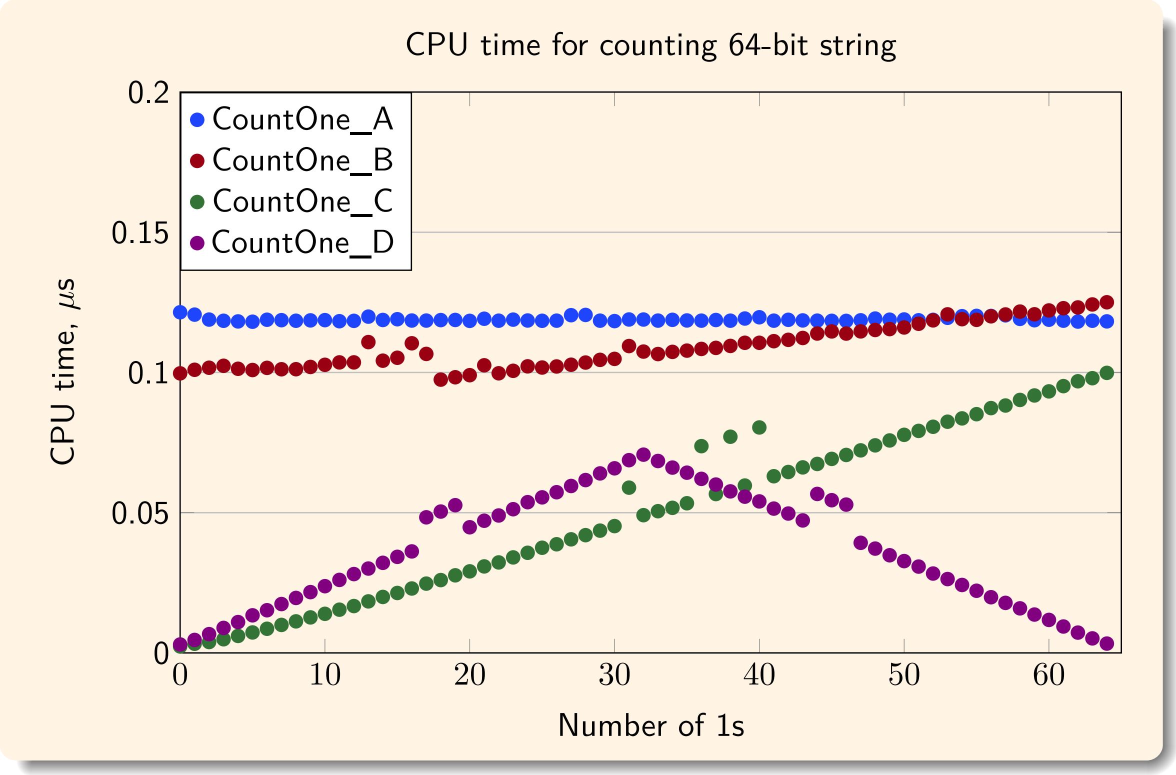
- The preceding algorithm can be modified to avoid worst-case scenario.

Algorithm 0.0.4.

```
    // Count the number of 1s in a bit string B.
    // Input: B = b_n b_{n-1} ··· b_2 b_1, int n > 0
    // Output: c, number of 1s in B.
Algorithm CountOne_D(B, n)
{
    BB := ~B; // BB is B's complement.
    c := 0; // Init c to 0
    while (B ≠ 0 and BB ≠ 0) do {
        c := c + 1;
        B := B & (B − 1); // Remove one 1 in B
        BB := BB & (BB − 1); // Remove one 1 in BB
    }
    if (BB = 0) c = n − c; // Fewer 0, c is number of 0 in B
    return c;
}
```

- Use BB to count the number of 0s in B.
- Algorithm stops when all 1s or 0s have been counted.

Comparisons of 4 Approaches



- `CountOne_D` appears to be the most efficient algorithm
  - Or is it?

Analyses of `CountOne_C` and `CountOne_D`

- `Algorithm CountOne_D`
  - Lines 5-9, loop is executed $\min\{c, n-c\}$ times
    - Loop body consists of 5 operations: 1 addition, 2 subtractions, 2 bit-wise ANDs
    - Maximum $\frac{5n}{2}$ total operations
- `Algorithm CountOne_C`
  - Maximum $3n$ operations
- Memory space needed
  - `Algorithm CountOne_C`
    - Local variable c is needed.
    - $B-1$ needs to be stored.
  - `Algorithm CountOne_D`
    - Local variable c is needed.
    - $B-1$ needs to be stored.
    - In addition, $BB = \sim B$ and $BB-1$ are needed.
    - Larger memory space requirement.

Comparison, Average-Case Performance

- Worst-case scenario, CountOne_D is faster than CountOne_C
- To compare average execution time for all possible input patterns
- Example, $n = 4$

	Loop iterations		Total #operations		
c	CountOne_C	CountOne_D	CountOne_C	CountOne_D	Frequency
0	0	0	0	0	1
1	1	1	3	5	4
2	2	2	6	10	6
3	3	1	9	5	4
4	4	0	12	0	1
Total	32	20	96	100	16
Ave.	2	1.25	6	6.25	

- Average-case execution time
  - Algorithm CountOne_D is a little slower than Algorithm CountOne_C.
- Need to consider which scenario is more important in a real application.
  - Worst-case, average-case, or best-case CPU time.

Most Efficiency Approach – CountOne_E

- A faster algorithm

Algorithm 0.0.5.

// Count the number of 1s in a bit string B.
// Input: $B = b_n b_{n-1} \cdots b_2 b_1$; int $n = 2^k$
// Output: B, number of 1s in bit string

Algorithm CountOne_E(B, n)
{
    $D_1 := 01010101 \cdots 0101$; // alternatinve 1 and 0.
    $D_2 := 00110011 \cdots 0011$; // two consecutive bits are 1s or 0s.
    $D_4 := 00001111 \cdots 1111$; // four consecutive bits are 1s or 0s.
    $\cdots\cdots\cdots$
    $D_k := 00000000 \cdots 1111$; // (n/2) 1s followed by (n/2) 0s.
    for $i := 1$ to k step 1 do {
        $B := (B \,\&\, D_i) + ((B >> 2^{i-1}) \,\&\, D_i)$; // >>: right shift
    }
    return B;
}

Algorithm CountOne_E Example

- Lines 8-10, loop is executed $k = \lg n$ times
  - Loop body consists of 4 operations
    - 1 right-shift, 2 bitwise AND, 1 addition
- For large n, this algorithm is the most efficient
- Execution example of Algorithm `CountOne_E`

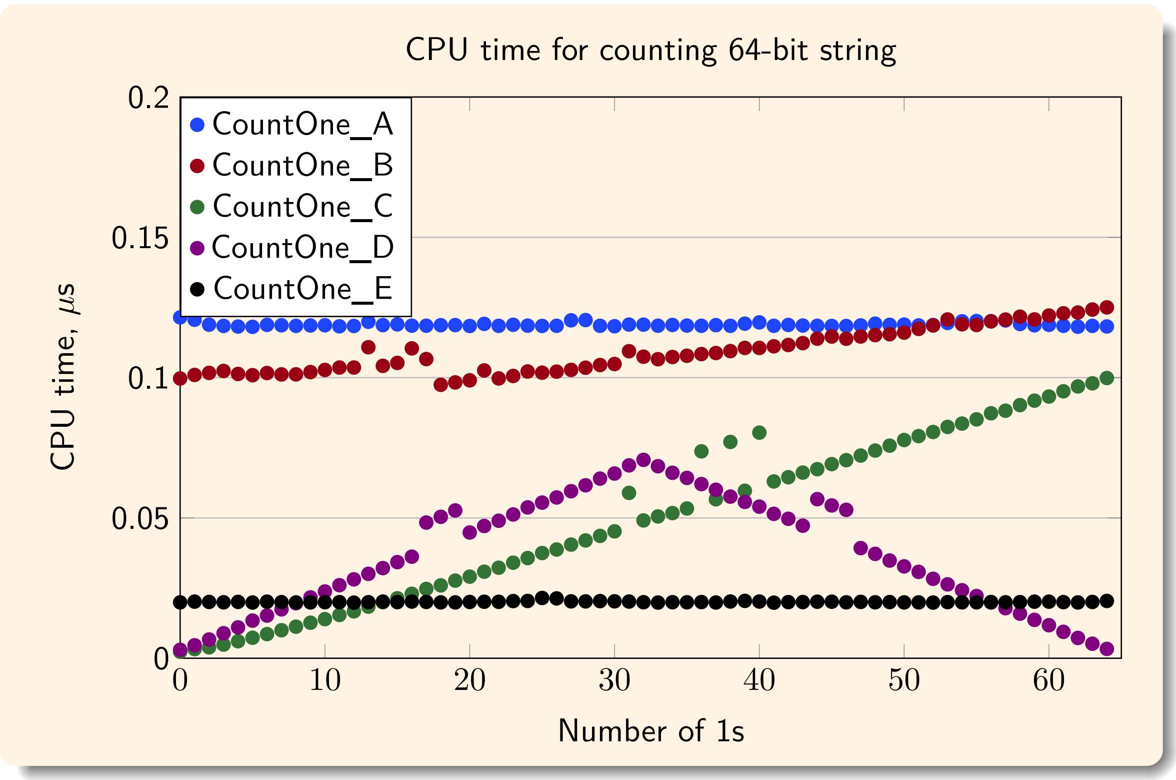
$B = 1101,0001$

Iteration 1	1101,0001
B & D_1:	0101,0001
$B >> 1$ & D_1:	0100,0000
B:	1001,0001

Iteration 2	1001,0001
B & D_2:	0001,0001
$B >> 2$ & D_2:	0010,0000
B:	0011,0001

Iteration 3	0011,0001
B & D_3:	0000,0001
$B >> 4$ & D_3:	0000,0011
B:	0000,0100

Comparisons of 5 Approaches



- `CountOne_E` is the most efficient and it's performance is independent to the number of 1s in the bit string.

Counting Ones in a Bit String – Summary

- Five different ways to count 1s in a bit string

Algorithm	Number of iterations	Operations per iteration	Worst-case #operations	Local memory
CountOne_A	n	1	n	c, i
CountOne_B	n	2	$n+c$	c, i
CountOne_C	c	3	$3n$	c
CountOne_D	$\min\{c, n-c\}$	5	$5n/2$	c, BB
CountOne_E	$\lg n$	4	$4\lg n$	$i, D_1, \cdots, D_k$

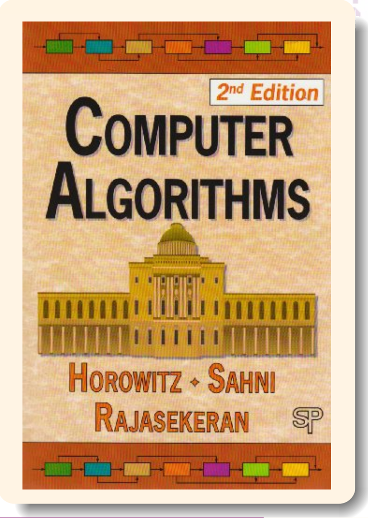
- CountOne_D and CountOne_E need more local memory
  - Shifted and AND results are assumed to store in registers.
- Choose the algorithm best fit for the applications.

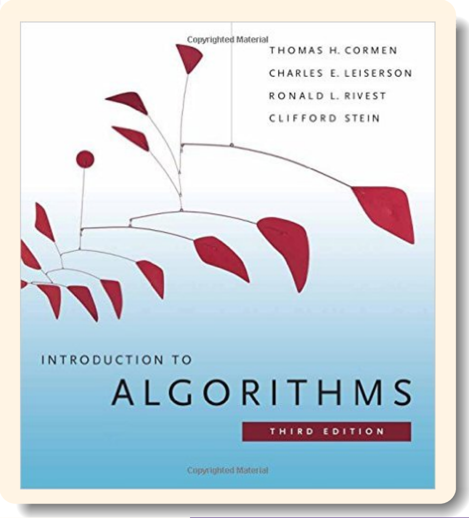
Study Algorithms

- Given a problem, there might be more than one way to solve it.
- Which algorithm is more efficient?
  - Time and memory space.
- Are there general methods to develop algorithms?
- Some problems have been solved, one should adopt the best approach for one's application.
- More aggressive goals
- What is the best algorithm for a particular problem?
- Can we find one, or is it possible?
- What if there is no algorithm that can solve the problem in reasonable time?

Algorithms – Course Information

- Class time: T3,T4,R3: lectures and discussions.
- Class room: Dalta 208.
- Text books
  - *Computer Algorithms*, by E. Horowitz, S. Sahni, and S. Rajasekeran, 2nd edition, Silicon Press, 2008.
  - *Introduction to Algorithms*, T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, 3rd edition, MIT Press, 2009.
- Office hours: Wednesday 10 - 11:30 AM.
  - Or by appointment (michang@ee.nthu.edu.tw).





Algorithms – Syllabus

Course Info

Unit 1. Analysis
- 1.1 Foundations
- 1.2 Analysis
- 1.3 Analysis, II
- 1.4 Mathematical backgrounds

Unit 2. Data structures
- 2.1 Stack and queue
- 2.2 Trees
- 2.3 Sets and graphs

Unit 3. Divide and conquer
- 3.1 Divide and conquer
- 3.2 Sorts
- 3.3 More on divide and conquer

Unit 4. Tree and graph traversal
- 4.1 Breadth first Search
- 4.2 Depth first Search

Unit 5. The greedy method
- 5.1 The greedy method
- 5.2 The greedy method, II
- 5.3 The greedy method, III

Unit 6. Dynamic programming
- 6.1 Dynamic programming
- 6.2 Dynamic programming, II
- 6.3 Dynamic programming, III

Unit 7. All-space searching methods
- 7.1 Backtracking
- 7.2 Branch and bound

Unit 8. Lower bound theory

Unit 9. $\mathcal{NP}$-hard and $\mathcal{NP}$-complete

Unit 10. Approximation algorithms

Unit 11. Randomized algorithms

Unit 12. Algebraic problems

Evaluation

- Evaluation

Category	% each	Number	Total
Homework	4	12	48
Midterm	16	2	32
Final	20	1	20
Absence	-1	-	-

- Homework:
  - Could be a significant loading,
  - C programming and report writing.
- Mid-term exams:
  - Apr. 28,
  - May 26,
  - Machine tests at EECS 406
- Final exam:
  - Jun. 30,
  - Machine test at EECS 406

Homework

- Homework is designed for you to practice what you have learned in class.
- Grading criteria:
  - Ontime submission (20%),
    - Due on 11:59 PM of the day specified on the announcement.
  - Solution correctness (50%),
  - Program and report writing (30%),
    - Legibility and efficiency,
    - Clearness and logic,
    - Solution approach and comments.
- Download and submit on EE workstations.
- Discussions with classmates encouraged but no plagiarism.
  - Write your own programs.
- Algorithms are solving specific problems
  - They should be language independent.
  - When implemented they become functions, procedures, or subroutines.
  - Applicable in structure programming and object oriented programming.
- We will practice implementing algorithms in more basic C programming language.
  - Programming guidelines are also the same as before.

Handouts and Homework

- Class handouts can be found on EE workstation.
  - Download (ftp) through daisy (140.114.24.31).
  - Directory: ~ee3980/notes
    - `lec10.pdf`,
    - `lec11.pdf`,
    - `lec21.pdf`, ...
- Homework can be found in each homework directory.
  - ~ee3980/hw01,
  - ~ee3980/hw02, ...
- Homework should be turned in on EE workstations.
- Submission command:

```
$ ~ee3980/bin/submit hw01 hw01.c hw01a.pdf
```

- To check homework or exam grades:

```
$ ~ee3980/bin/score
```