Faculty of Information Technology, Monash University

FIT2004: Algorithms and Data Structures

Week 12: Topological Sort and Design Principles

These slides are prepared by M. A. Cheema and are based on the material developed by Arun Konagurthu and Lloyd Allison.

Overview

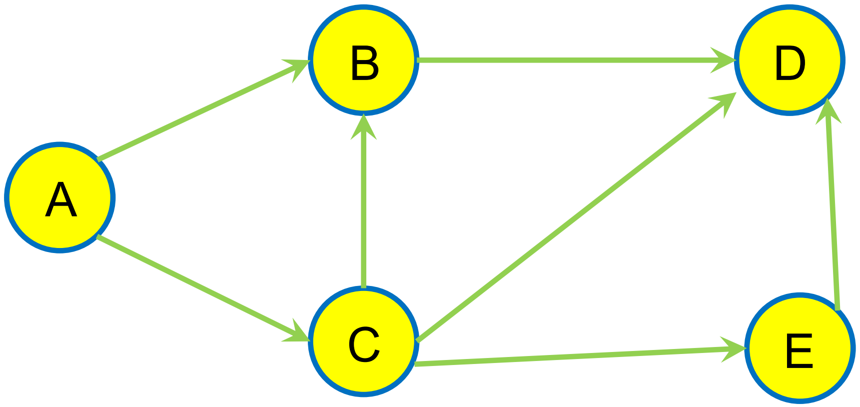
- Topological Sort
  - Kahn's Algorithm
  - Depth First Search
- Design Principles (FIT2004 Summary)
- Final Exam etc.

Directed Acyclic Graph (DAG)

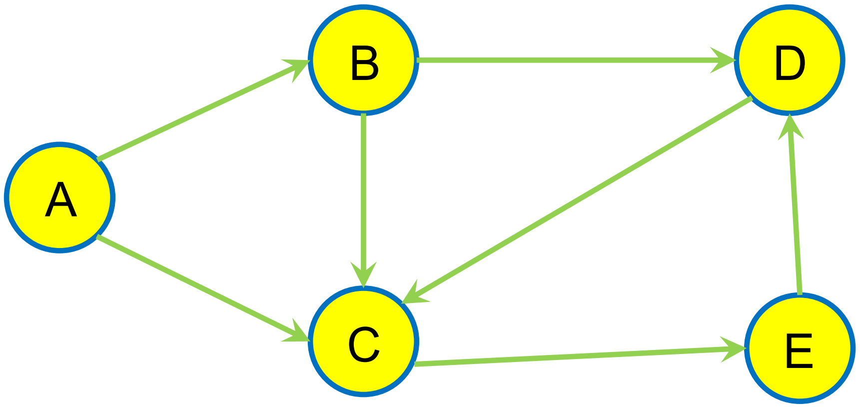
A Directed Acyclic Graph (DAG) is

- Directed
- Acylcic – has no cycles
- Graph

Which of the two graphs is a DAG?



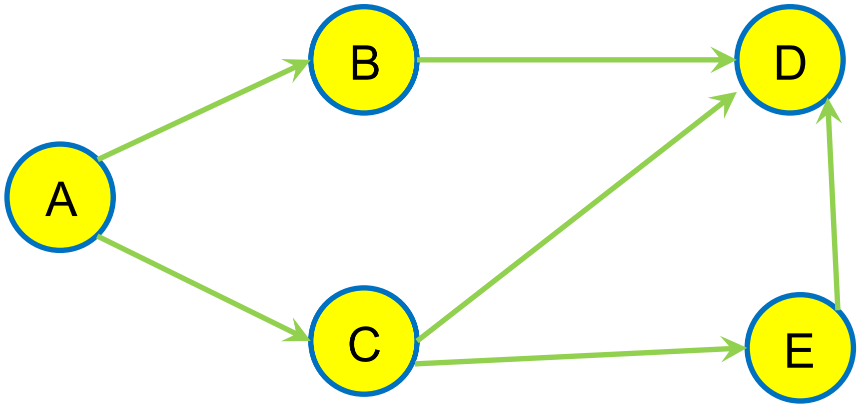
Graph 1

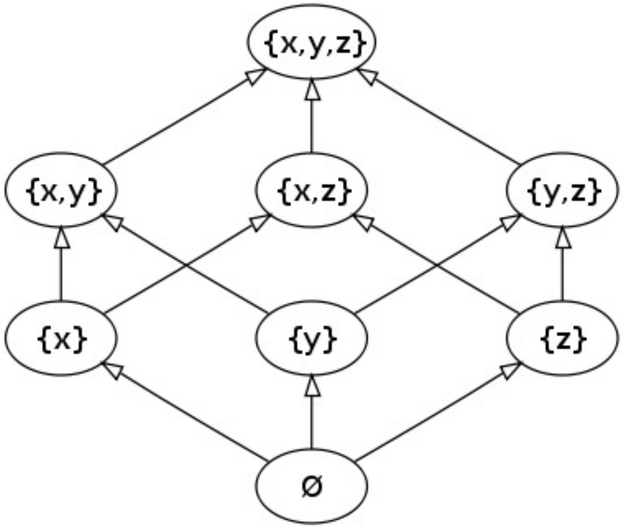


Graph 2

DAG: Examples

- sub-tasks of a project and which "must finish before"
  - A → B means task A must finish before task B
  - so, DAGs useful in project management
- relationships between subjects for your degree -- "is prerequisite for"
  - A→B means subject A must be completed before enrolling in subject B
- people genealogy – "is an ancestor of"
  - A → B means A is an ancestor of B
- power sets and "is a subset of"
  - A → B means A is a subset of B





Source: wikipedia

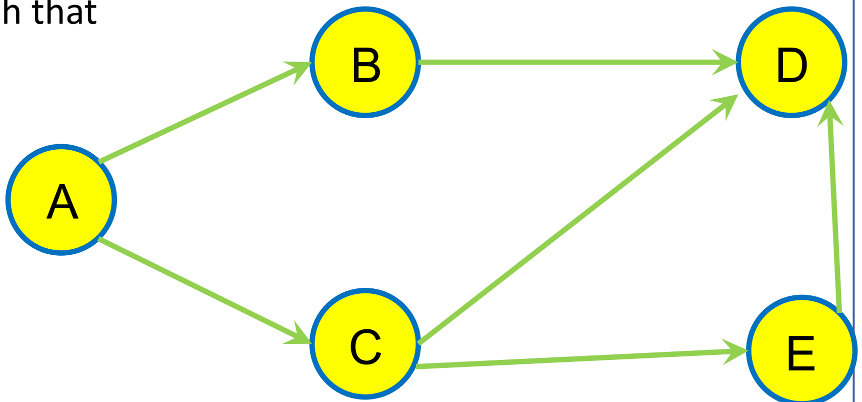
Topological Sort of a DAG

Order of vertices in a DAG

- A < B if A→B.
  - Note that if A → B and B→D, we have A < B and B < D which implies that A < D (i.e., transitivity).
- Some vertices may be incomparable (e.g., B and C are incomparable), i.e. A< B and A < C but we do not know whether C < B or B < C.

A topological Sort

- is a permutation of the vertices in the original DAG such that
- for **every** directed edge u→v of the DAG
  - u appears before v in the permutation



Example: A, B, C, E, D

- Topological sort of a DAG of “is prerequisite of” example gives an ordering of the subjects for studying your degree, one at a time, while obeying prerequisite rules.

Topological Sort of a DAG

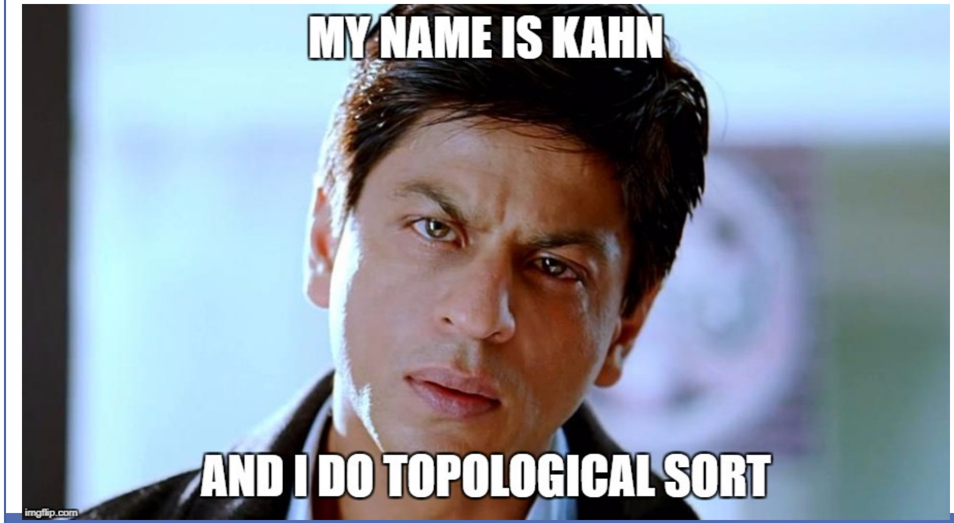
- A DAG can have many valid topological sorts, e.g., let u and v be two incomparable vertices, u may appear before or after v.

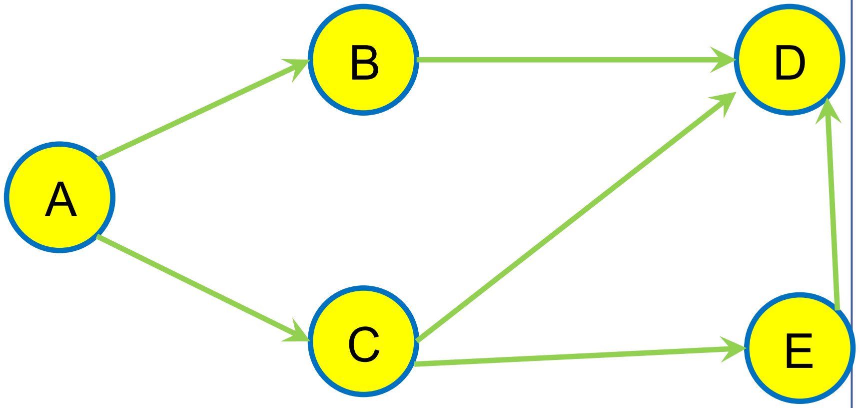
Which of these is NOT a valid topological sort of the DAG

1. A, B, C, E, D
2. A, C, B, E, D
3. A, C, E, B, D
4. A, B, E, C, D

How to do topological sort?

- Kahn's Algorithm



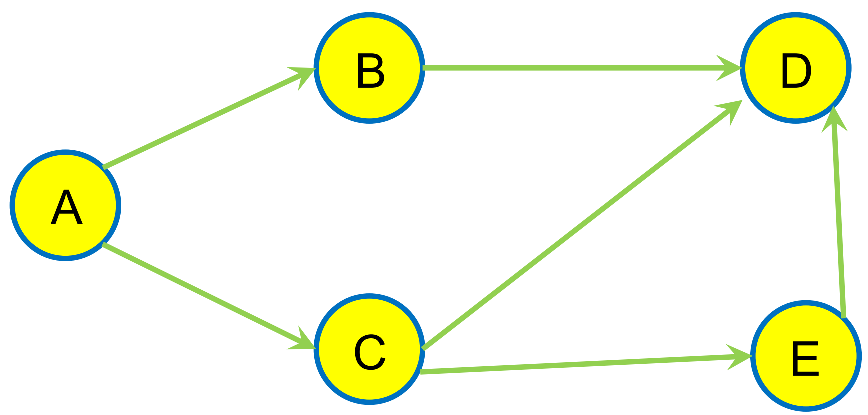


Overview

- Topological Sort
  - Kahn's Algorithm
  - Depth First Search
- Design Principles (FIT2004 Summary)
- Final Exam etc.

Kahn's Algorithm: High level idea

```
For each vertex v that does not have ANY incoming edge
   Add v to sorted
   Remove the outgoing edges of v
```



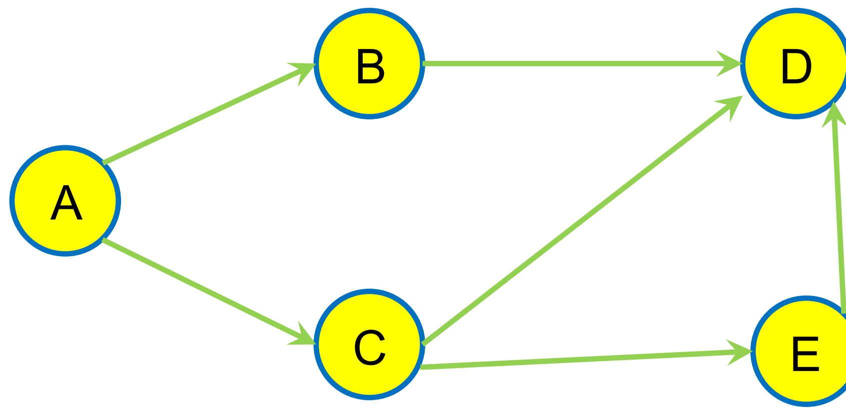
Sorted:

A	B	C	E	D

Kahn's Algorithm: Detailed pseudocode

```
initialize Sorted to be empty # Sorted will contain the topological sort
initialize a list L with vertices that do not have any incoming edge ?
while L is not empty:
    remove any vertex v from L
    Sorted = Sorted + {v}
    for each outgoing edge v → u of v:
        remove edge v → u from the graph
        if u has no other incoming edge: ?
            insert u in L # all the vertices that must appear before u have already been added to Sorted
if graph still has some edges:
    return error # graph has a cycle
else:
    return Sorted
```

Time Complexity?



L: | [illegible] | C |
|---|---|

Sorted: | A | B | C | E | D |
|---|---|---|---|---|

Kahn’s Algorithm: Complexity

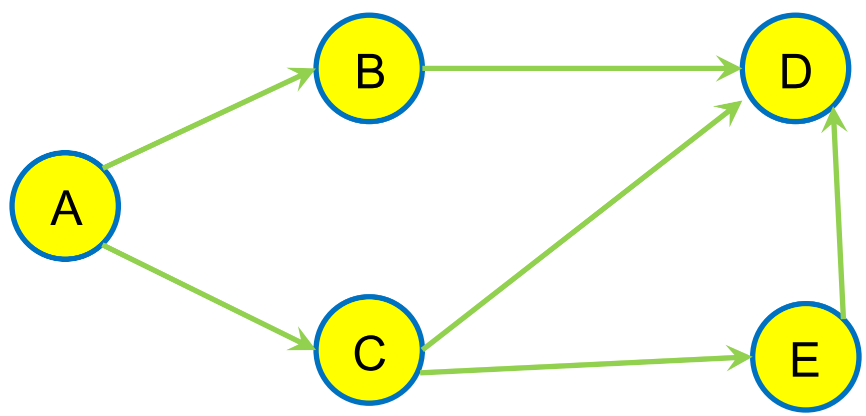
```
initialize Sorted to be empty # Sorted will contain the topological sort
initialize an array IncomingEdges[] of size V with all values set to 0
for each edge u → v in E:
    IncomingEdge[v] += 1
initialize a list L containing vertices for which IncomingEdges[v] = 0
while L is not empty:
    remove any vertex v from L
    Sorted = Sorted + {v}
    for each outgoing edge v → u of v:
        remove edge v → u from the graph
        IncomingEdges[u] = IncomingEdges[u] - 1
        if IncomingEdges[u] == 0: # u has no incoming edge
            insert u in L
if graph still has some edges:
    return error # graph has a cycle
else:
    return Sorted
```

O(E+V)

O(E+V)

Time Complexity: O(V+E)

Space Complexity: O(V+E)



Overview

- Topological Sort
  - Kahn's Algorithm
  - Depth First Search
- Design Principles (FIT2004 Summary)
- Final Exam etc.

Depth First Search (DFS)

Below is the DFS algorithm we saw in week 8

- function DFS(v):
  - Mark u as Visited
  - For each adjacent edge (v,u)
    - If u is not visited
      - DFS(u)

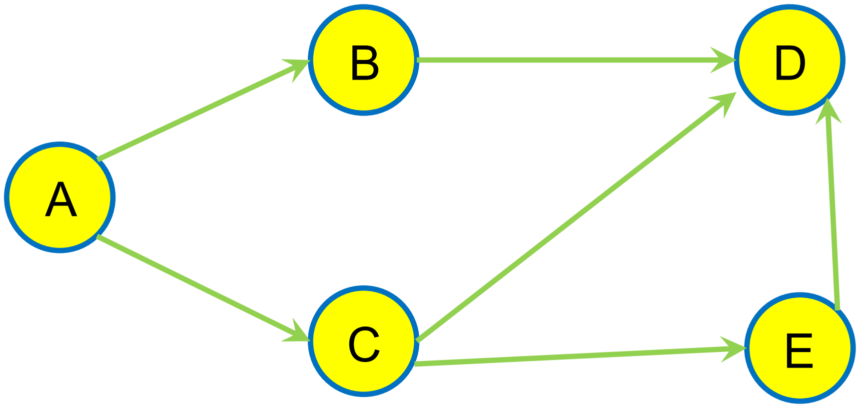
Assume we call DFS(A), which of the following is NOT a possible order in which vertices are marked visited.

A, B, D, C, E

A, C, E, D, B

A, C, D, E, B

A, C, E, B, D

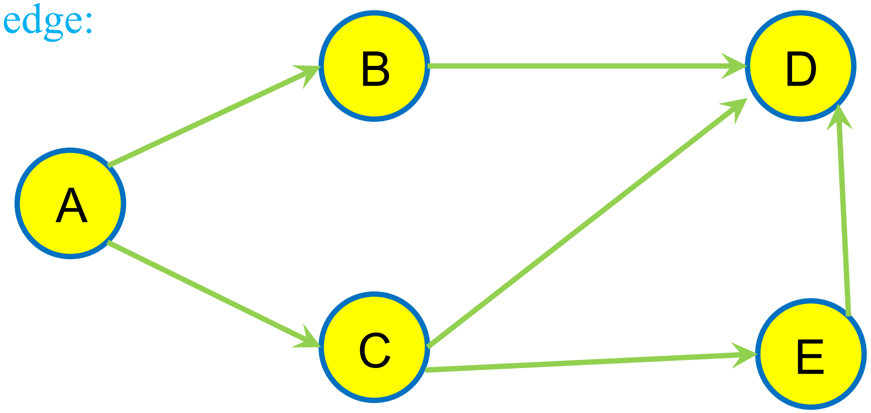


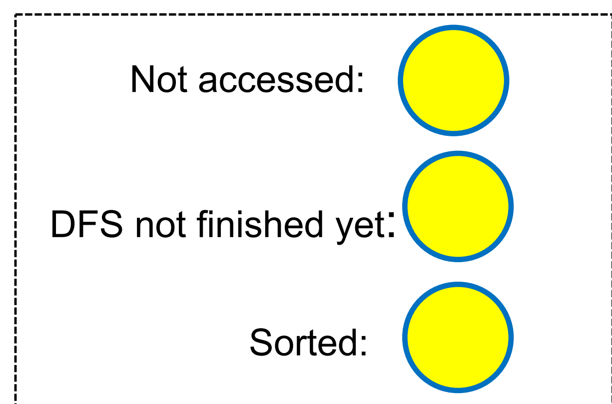
DFS for Topological Sort

Using DFS for topological sort assuming graph is a DAG!

- For each vertex v that does not have any incoming edge:
  - Call DFS(v)

- function DFS(v):
  - For each adjacent edge (v,u)
    - If u is not visited
      - DFS(u)
  - Mark v visited and append in Sorted (at front)





Time and space complexity:

O(V+E)

Sorted:

A	C	E	B	D

Overview

- Topological Sort
  - Kahn's Algorithm
  - Depth First Search
- Design Principles (FIT2004 Summary)
- Final Exam etc.

Design Principles (Summing up FIT2004)

Here are some broad strategies to (try to) solve algorithmic problems:

- Look out for good invariants to exploit
- Attempt to balance your work as much as possible
- Do not repeat work (so, store and re-use!)
- Use appropriate data structures
- Try well-known problem solving strategies
- Sometimes greed is good!
- These are general guidelines. As always, there are many exceptions

Look out for good invariants to exploit

- Here are some algorithms we considered in the unit that do precisely this!
- Binary Search (Refer Week 2 lecture)
- Sorting (Refer Lectures from Weeks 2 and 3)
- Shortest Paths and Connectivity
  - Dijkstra’s algorithm (Refer Week 8 Lectures)
  - Floyd-Warshall algorithm (Refer Week 9 lectures)
- Minimum Spanning Tree Algorithms (Refer Week 10 lectures)

Balance your work as much as possible

- For problems that allow division of labour (eg. Divide and Conquer)
- Try to divide work equally as much as possible
- Merge sort achieves this
  - O(N log N)-time always!
- Quick sort does not necessarily achieve this – depends on the choice of the pivot (Refer week 3)
  - Good pivots give O(N log N)-time
  - Bad pivots give $O(N^2)$-time

Choose Data Structures with care

- Certain data representations are more efficient than others for a given problem
- Priority Queue in Dijkstra’s algorithm (Refer Week 8)
- Union-Find data structure in Kruskal’s algorithm (refer Week 9)
- Efficient Search and retrieval data structures of various kinds (Refer Weeks 5,6,7 lectures)

Don't repeat work

- Do not compute anything more than once (if there is room to store it for reuse)
- Underpins Dynamic Programming strategy
  - Edit Distance (Refer Week 4 Lecture)
  - Knapsack Problem (Refer Week 4 Lecture)

Try well known problem solving strategies

- Divide and Conquer (Refer Weeks 3, 4 lectures)
- Dynamic Programming (Refer Weeks 4, 8, 9 lectures)

Sometimes greed is good

- A greedy strategy is to make a “local” choice based on current information
- Sometimes gives optimal solution, e.g.
  - Dijkstra’s single source shortest paths algorithm (Refer Week 8 Lectures)
  - Minimim Spanning Tree Algorithms – Prim’s and Kruskal’s (Refer Week 10 lectures) minimum spanning tree algorithm.
- Greedy is sometimes a good heuristic!
  - Sometimes gives a “good” solution to a (combinatorial) problem even if not guaranteed optimal

Overview

- Topological Sort
  - Kahn’s Algorithm
  - Depth First Search
- Design Principles (FIT2004 Summary)
- Final Exam etc.

Final Exam

- Time allowed: 2 hours + 10 minutes reading time
- Total Marks: 60
- Exam is **NOT** open book
- Question style similar to mid-semester tests – but do not try to guess what will be on the exam
  - If a question asks you to **describe an algorithm**, you can write your idea in plain English
  - If a question asks you to **write pseudocode**, you **must** write your idea in a more structured way (like the ones in lecture slides or even Python code)
- Hurdles:
  - At least 16 out of 40 marks in in-semester assessments (assignment + mid-semester test + lecture/tutorial participation)
  - At least 24 out of 60 marks in the final exam
  - At least 50 marks overall
- Do not miss final exam even if you fail in-semester hurdle.
  - It affects your WAM

Non-Examinable Content

- Additional material in lecture notes is NOT examinable
  - In other words, anything NOT covered in lectures, tutorials, labs is NOT EXAMINABLE!
- Advanced questions in tutorials are NOT examinable

Consultations for Final Exam

- Consultation sessions have been organized
- At least one each day
- Please check Moodle before attending the consultation

Date	Day	Time	Name of the tutor	Location
3 June 2019	Monday	11am-12pm	Chaluka Salgado	G20/ 14 Rainforest walk
4 June 2019	Tuesday	12pm-1pm	Chaluka Salgado	G22/ 14 Rainforest walk
5 June 2019	Wednesday	11am-12pm	Tharindu Warnakula	G21/ 14 Rainforest walk
6 June 2019	Thursday	1pm-2pm	Vishwajeet Kumar	G20/ 14 Rainforest walk
7 June 2019	Friday	11am-12pm	Tharindu Warnakula	G20/ 14 Rainforest walk
10 June 2019	Monday	11am-1pm	Shams Rahman	G15/ 14 Rainforest Walk
11 June 2019	Tuesday	11am-1pm	Nathan Companez	G14A/ 14 Rainforest Walk

Consultations for Final Exam

- Please come to the consultations prepared
  - Do not ask questions like “Can you please explain Dynamic Programming from scratch?”.
- Don’t try getting hints about the questions on final exam!
  - E.g., Is Kruskal’s algorithm going to be on the exam?
  - Important: Last night I got an accident and have lost my memory – so do not waste your energy
  - Only I, Ammar (head tutor), Ian (lecturer at Malaysia campus) and the Chief Examiner have seen the exam paper. Some of the options you have are the following
    - Selfish Option
      - In many Bollywood movies, a second accident brings the memory back.
      - Pray that I get another accident
      - but the second accident may bring false memories and you may get all the wrong hints :P
    - Risky Option
      - Ask Ammar. He has told me that he will lie to you :P
    - Expensive Option
      - Buy a ticket to Malaysia and try your luck by talking to Ian :P
    - Useless Option
      - Ask the Chief Examiner
      - I am the Chief Examiner :P
    - Easiest Option
      - Study!!!

Suggestions for preparation (written after I lost my memory)

- Understand how each algorithm works
- Practice writing pseudocode for each algorithm
- Understand its complexity analysis
  - Don't confuse algorithms: Bellman-Ford vs Floyd-Warshall vs Ford-Fulkerson
    - Despite warning, every semester, students mix up algorithms losing all marks for the question
- Solve different recurrence relations
- Prove by **induction** for recurrence solutions
- Practice writing proofs of correctness of algorithms

Algorithm for exam preparation

```
1.  for i in range(1,13):
2.     carefully go through lecture slides and lecture notes week i and ensure you understand the content
3.     practice writing pseudocode for each algorithm covered in week i
4.     make sure your understand space/time complexities
5.     if something is unclear:
6.         listen lecture recording # slides are optimized for lectures and should only be treated as summaries.
7.         seek help
8.  for i in range(2,13):
9.       attempt tutorial questions week i
10.      for each question attempted:
11.          compare your solution with the sample solution
12.          seek help if something is unclear
13. for each question on the sample exam and mid-semester tests:
14.       attempt the question
15.       post your solution on Moodle forum
16. If current_date <= exam_date
17.            goto line 1
18. Print("HURRAYYYYYY!!!  I did it!")
```

```
# WARNING: This is a brute force algorithm. Using greedy algorithm (studying selected topics) may be disastrous
```

Summary

Take Home Message

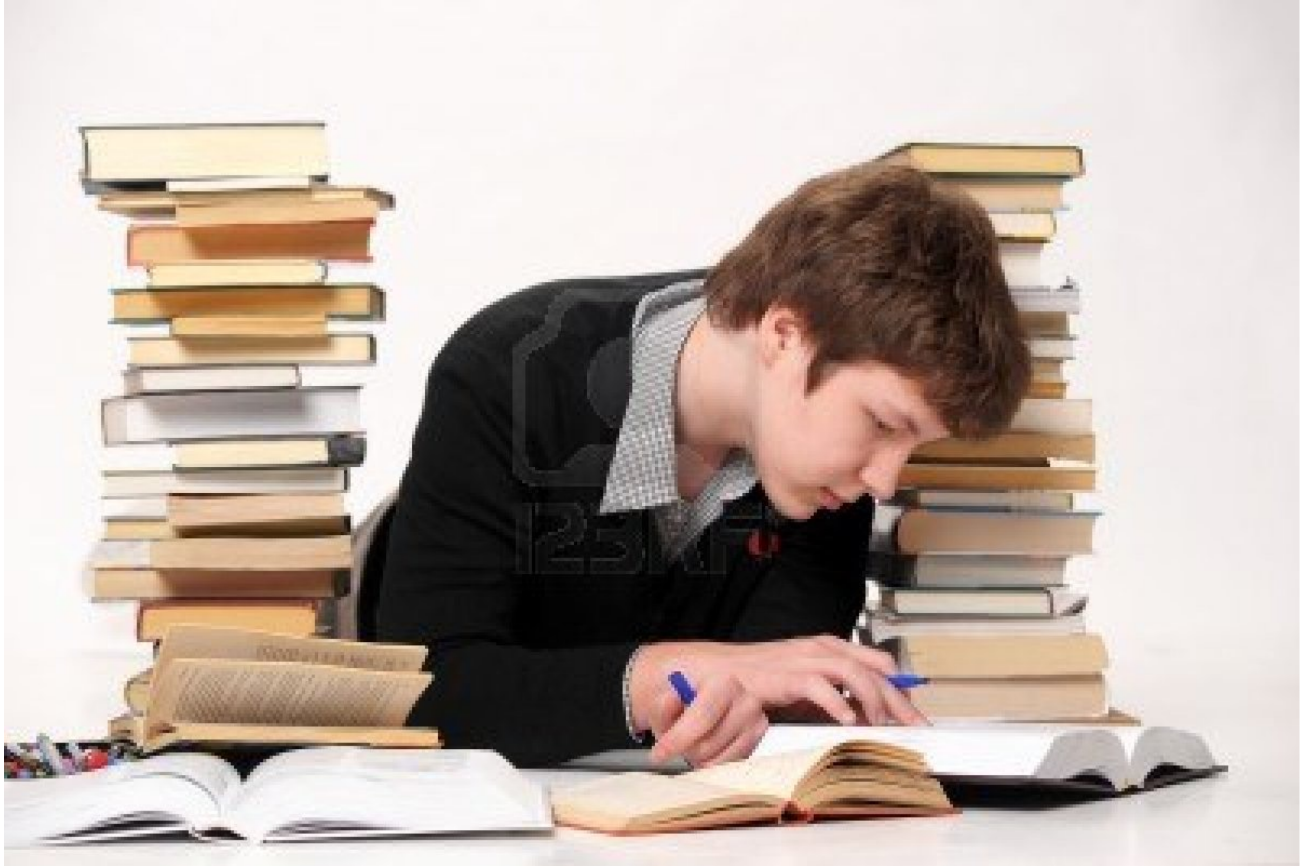
- **See Design Principles in this lecture**

Things to do (this list is not exhaustive)

- **See the algorithm to prepare for final exam**

Coming Up Next

Coming Up Next



SWOT VAC

Coming Up Next

Final Exam

Coming Up Next

Results Day

And you live happily ever after

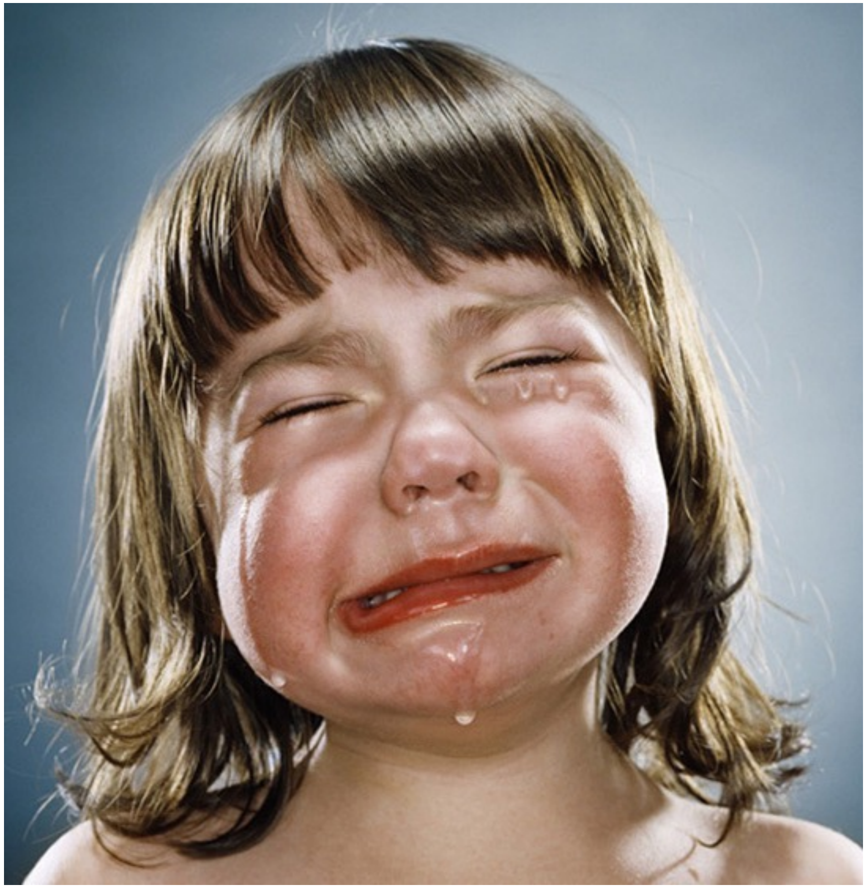
Coming Up Next (Second Scenario)

SWOT VAC

Coming Up Next (Second Scenario)

Final Exam

Coming Up Next (Second Scenario)



Results Day

Moral: Work hard or be brave!

Just before I sign off …

- Most of you have done really well given the **challenging** nature of the unit!
- Good luck for the final exam!
- Do not hesitate to contact me if I can be of any help even after this semester
  - E.g., if you need a reference letter

That's all folks!