CS630 Graduate Algorithms

September 3, 2024

by Dora Erdos and Jeffrey Considine

Today:

- course info, logistics
- semester topics with examples

No labs tomorrow, Wed Sept 4!

Course Staff

Prof. Dora Erdos

edori@bu.edu

Prof. Jeffrey Considine

jconsidi@bu.edu

TF Spyros Dragazis

dragazis@bu.edu

TF Garry Kuwanto

gkuwanto@bu.edu

Prerequisites

Undergraduate prereq: CS330

Graduate prereq: an undergraduate algorithms course, should be familiar with the following topics:

- Proof techniques (e.g. direct proof, proof by contradiction, induction)

- Data structures (e.g. lists, queues, heaps, hash tables, trees, graph adjacency list)

- Asymptotic analysis of running time (i.e. big-Oh)

- Algorithm design paradigms, such as greedy, divide and conquer, dynamic programming, various graph algorithms

- pseudocode

Course Infrastructure

Syllabus:

https://cs-people.bu.edu/edori/CS630_Fall_2024_Syllabus.pdf

Piazza: https://piazza.com/bu/fall2024/cs630

- all course communication — questions, announcements
- link to Google Drive
- time and location/link for office hours

Gradescope: https://www.gradescope.com/courses/847056 Entry Code: G378R5

- homework submission

TopHat: https://app.tophat.com/ course code 019862

- in class real-time questions

Google Drive:

- slides, lab problems, homework assignments, other material

Textbook

No required text, readings will be posted prior to each lecture in the Google Drive.

Some useful textbooks:

- Kleinberg, Tardos, *Algorithm Design.*
- Cormen, Leiserson, Rivest, and Stein. *Introduction to Algorithms.*
- Dasgupta, Papadimitriou, Vazirani. *Algorithms*.
- V. Vazirani. *Approximation Algorithms*.
- Williamson and Shmoy. *The Design of Approximation Algorithms*.
- Motwani, Raghavan. *Randomized Algorithms*.
- Harvey. *A First Course in Randomized Algorithms*.

For prerequisite review:

- Tim Roughgarden. *Algorithms Illuminated*. (the 4th part of the book also contains material on NP)
- J. Erickson. *Algorithms,* 2019. Available from http://algorithms.wtf/
  See also the extensive exercises on the website.

Mathematics for Computer Science by Eric Lehman, Tom Leighton, and Albert Meyer.
(Useful background on discrete mathematics.)

Exams and Grading

Midterm exam: **Thursday 10/10** in class

Final exam: during finals week

- material is cumulative
- make travel plans after you know your final exam schedule, makeup exams will not be given for travel reasons.

Grading:

5% attendance - based on TopHat

30% weekly homework assignments (first assignment due Wed 9/18)

30% in-class midterm (Thur 10/10)

35% cumulative final exam (during finals week)

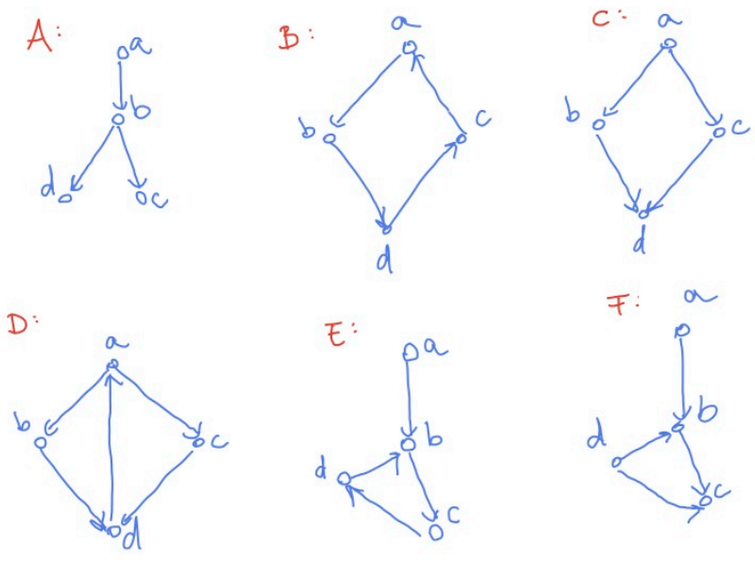
] have to pass both exams → at least 40% of points

TopHat

- 2-4 multiple choice questions per class
  - questions are designed to help you with understanding the material.
  - counts towards 5% of your final grade
    - each problem is worth 1 pt = 0.8 for participation + 0.2 for correctness
    - you need to earn 80% of the total available points to get full credit.

A directed acyclic graph (DAG) is a directed graph that contains no directed cycles.

- a *directed* cycle is a sequence of vertices $v_0, v_1, .., v_k$ so that there are *directed* edges between subsequent nodes (v_i, v_{i+1}) and an edge (v_k, v_0).



TopHat:
Select all the graphs that are DAGs.

TopHat

- 2-4 multiple choice questions per class
  - questions are designed to help you with understanding the material.
  - counts towards 5% of your final grade
    - each problem is worth 1 pt = 0.8 for participation + 0.2 for correctness
    - you need to earn 80% of the total available points to get full credit.

Algorithm 1: DFSWrapper(G, s)

```
   /* G adjacency list as nested hash table, s source node     */
1  parents ← empty hash table/* parents of nodes in the DFS tree
       (forest)                                                 */
2  times ← empty hash table /* tuples of <discovery time, finish
       time>                                                    */
3  time ← 0/* time counter                                     */
4  parents[s] ← None;
5  DFS(G, s);
6  return parents, times
```

Algorithm 2: DFS(G, u)

```
1  time ← time + 1;
2  times[u][0] = time;
3  for v in G[u] do
       /* recursively explore u's neighbors
4      if v not in parents then
5          parents[v] = u;
6          DFS(G,v)
7  time ← time + 1;
8  times[u][1] = time;
```

Question: How many times total is line 6 called across the recursive calls to DFS?

A. O(n)

B. O(m)

C. O(n+m)

D. O(nm)

TopHat

- 2-4 multiple choice questions per class
  - questions are designed to help you with understanding the material.
  - counts towards 5% of your final grade
    - each problem is worth 1 pt = 0.8 for participation + 0.2 for correctness
    - you need to earn 80% of the total available points to get full credit.

- subscription fee $35

How should we implement `find_compatible_room(c,A)` using PQs?

Question. PQs contain <key, value> pairs. What are the tuples to use for `find_compatible_room`?

A. *key* = room id , *value* = last lecture id

B. *key* = room id, *value* = room finish time

C. *key* = room finish time, *value* = last lecture id

D. *key* = room finish time, *value* = room id

Homework Assignments

- workload in this course is **heavy**
- assignments to be submitted via **Gradescope** every week
  - solutions should be **typeset** (LaTeX or any text editor)
- posted Thursdays, due **Wednesdays 11:59 pm**
  - first assignment due Wed 9/18
- 2 lowest grades dropped
- no late assignments
- regrade requests through Gradescope

Problems are mostly theory. Often involves clever application or modification of some algorithm that we study in class. Solving a problem that requires creativity takes time, make sure that you start early.

Collaboration Policy and Academic Conduct

Exam:

- absolutely **NO** collaboration is allowed.
- closed book, some limited notes (cheat sheet)

Homework:

- ask questions and start discussions on Piazza!
- you can collaborate with up to **3 people** on each assignment
- you have to write down the solutions by yourself using your own words
- list your collaborators on your assignment (or explicitly write collaborators: none)

Academic Conduct:

- participants must follow the CAS Academic Code of Conduct `https://www.bu.edu/academics/policies/academic-conduct-code/`
- the work you submit has to be your own (except for your collaborators or input from course staff)
- if you use some resource from outside of class (e.g. web, book) use proper citation
- you are forbidden to explicitly search for the problem solutions online or get them from a person not currently enrolled in this course

Course Support

- Discussion sections/ labs
  - problem sets related to the algorithms we study, hints for the hw
- hw assignments
  - are a tool to help you study
  - we grade hw with the intent to give you feedback
- Piazza
  - ask questions, clarification about course material, assignments, logistics, etc.
- Office hours

We want you to succeed in this course, but you need to help us to do that

- make use of the above resources
- there is no such thing as a silly or too simple question, please ask
- if you are having difficulties, either academic or personal, **reach out** so that we can help.

Algorithm design and analysis

Algorithm: a finite set of *unambiguous instructions* for solving computational problems.

What is a good algorithm?

- efficient runtime → scalable with growing input size
- less resources → space requirements
  - quality of hardware
- abstract / general
  → so that we can apply the same idea to multiple problems
- simple, easy to understand
- correct

Set Cover

Finding the minimum set of courses to fulfill your HUB requirements (for this example, assume you need just one unit of each.)

do we need this course?

total : 5 courses

	CS111	CS112	CS132	AN101	AN103	EE150	BI306	AR307
Quant Reas	X	X	X					X
Crit. think	X	X	X		X			X
Creat/Inno	X	X						
Digi/multi			X					
res and inf				X		X	X	X
Social Inq				X	X			
Indiv in Com					X			
Sci Inq						X	X	X
History						X		

CS132 is the only course covering Digi/multi

AN101 Intro to Cult Anthropology AN103 Anthropology through Ethnography AS105 EE150 Sustainable Energy BI306 Bio of global change AR307 Archeological Science

Set Cover

Finding the minimum set of courses to fulfill your HUB requirements

Algorithm: always pick the course with the largest number of additional HUB units (this is a greedy algorithm!)

	CS111	CS112	CS132	AN101	AN103	EE150	BI306	AR307
Quant Reas	X	X	X					X
Crit. think	X	X	X		X			X
Creat/Inno	X	X						
Digi/multi			X					
res and inf				X		X	X	X
Social Inq				X	X			
Indiv in Com					X			
Sci Inq						X	X	X
History						X		

AN101 Intro to Cult Anthropology AN103 Anthropology through Ethnography AS105 EE150 Sustainable Energy BI306 Bio of global change AR307 Archeological Science

Set Cover

Finding the minimum set of courses to fulfill your HUB requirements

Algorithm: always pick the course with the largest number of additional HUB units

	CS111	CS112	CS132	AN101	AN103	EE150	BI306	AR307
Quant Reas	X	X	X					X
Crit. think	X	X	X		X			X
Creat/Inno	X	X						
Digi/multi			X					
res and inf				X		X	X	X
Social Inq				X	X			
Indiv in Com					X			
Sci Inq						X	X	X
History						X		

AN101 Intro to Cult Anthropology AN103 Anthropology through Ethnography AS105 EE150 Sustainable Energy BI306 Bio of global change AR307 Archeological Science

Set Cover

Finding the minimum set of courses to fulfill your HUB requirements

Algorithm: always pick the course with the largest number of additional HUB units

	CS111	CS112	CS132	AN101	AN103	EE150	BI306	AR307
Quant Reas	X	X	X					X
Crit. think	X	X	X		X			X
Creat/Inno	X	X						
Digi/multi			X					
res and inf				X		X	X	X
Social Inq				X	X			
Indiv in Com					X			
Sci Inq						X	X	X
History						X		

AN101 Intro to Cult Anthropology AN103 Anthropology through Ethnography AS105 EE150 Sustainable Energy BI306 Bio of global change AR307 Archeological Science

Set Cover

Finding the minimum set of courses to fulfill your HUB requirements

Algorithm: always pick the course with the largest number of additional HUB units

	CS111	CS112	CS132	AN101	AN103	EE150	BI306	AR307
Quant Reas	X	X	X					X
Crit. think	X	X	X		X			X
Creat/Inno	X	X						
Digi/multi			X					
res and inf				X		X	X	X
Social Inq				X	X			
Indiv in Com					X			
Sci Inq						X	X	X
History						X		

AN101 Intro to Cult Anthropology AN103 Anthropology through Ethnography AS105 EE150 Sustainable Energy BI306 Bio of global change AR307 Archeological Science

Set Cover

Finding the minimum set of courses to fulfill your HUB requirements

Algorithm: always pick the course with the largest number of additional HUB units

set cover : "cover" all elements in the set of HUB units

	CS111	CS112	CS132	AN101	AN103	EE150	BI306	AR307
Quant Reas	X	X	X					X
Crit. think	X	X	X		X			X
Creat/Inno	X	X						
Digi/multi			X					
res and inf				X		X	X	X
Social Inq				X	X			
Indiv in Com					X			
Sci Inq						X	X	X
History						X		

AN101 Intro to Cult Anthropology AN103 Anthropology through Ethnography AS105 EE150 Sustainable Energy BI306 Bio of global change AR307 Archeological Science

Correctness

Set Cover: Given a set of n items $U = \{u_1, u_2, \ldots, u_n\}$ and m subsets of these items $S_1, S_2, \ldots S_m$ find a minimum number of subsets, such that their union contains every element in U.

Is this algorithm "correct"? How can you prove this?

no: we ended up with one extra class

yes: it wasn't the opt answer, but it was the quickest.

yes: the algo yields a subset of courses that covers every HUB unit

Correctness

Is this algorithm “correct”? How can you prove this?

- We can write a formal proof that an algorithm always returns the correct answer.
  - formal proof that a shortest paths algorithm, e.g.Dijkstra’s, always finds the shortest paths
  - the Greedy Set Cover algorithm doesn’t terminate while there are uncovered items left - unless there are no more sets available - hence, the output must cover all items. ← this argument doesn't prove opt, just that it's a set cover
- Find a certificate
  - for each specific input some kind of evidence that the output indeed solves the problem
    - path in the graph that has the length that is returned by the shortest paths algorithm
    - for each item in the input we can map it to a set in the output that contains it.
- ?? ← we haven't proven optimality

Set Cover

Finding the minimum set of courses to fulfill your HUB requirements

Algorithm: always pick the course with the largest number of additional HUB units

	CS111	CS112	CS132	AN101	AN103	EE150	BI306	AR307
Quant Reas	X	X	X					X
Crit. think	X	X	X		X			X
Creat/Inno	X	X						
Digi/multi			X					
res and inf				X		X	X	X
Social Inq				X	X			
Indiv in Com					X			
Sci Inq						X	X	X
History						X		

AN101 Intro to Cult Anthropology AN103 Anthropology through Ethnography AS105 EE150 Sustainable Energy BI306 Bio of global change AR307 Archeological Science

Set Cover

Finding the minimum set of courses to fulfill your HUB requirements

	CS111	CS112	CS132	AN101	AN103	EE150	BI306	AR307
Quant Reas	X	X	X					X
Crit. think	X	X	X		X			X
Creat/Inno	X	X						
Digi/multi			X					
res and inf				X		X	X	X
Social Inq				X	X			
Indiv in Com					X			
Sci Inq						X	X	X
History						X		

AN101 Intro to Cult Anthropology AN103 Anthropology through Ethnography AS105 EE150 Sustainable Energy BI306 Bio of global change AR307 Archeological Science

Efficient running time

Running time: number of *computational steps* that an algorithm takes on an input of size n

- usually asymptotic running time - $O(n^2)$ or $O(n+ m \log n)$, etc.
- what type of running time?
  - worst case - number of comp steps on the worst (slowest) input
  - average case - number of comp steps averaged over all possible inputs
  - amortized - number of comp steps averaged over a sequence of inputs
    - e.g. resizing a dynamic array ← in most iterations the algo is very fast but occasionally we have to do some time consuming operation

amortized time : is it worth the tradeoff ?

Efficient running time

Running time: number of *computational steps* that an algorithm takes on an input of size n

Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record the algorithm as taking a very long time.

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

image source: Kleinberg, Tardos, ch. 2.1 page 34.

Complexity classes

How can we prove that there is no efficient - in this context: polynomial - algorithm?

if there is an efficient algo → we can give you the algo and you can see that it's fast

Often we can't.

Complexity classes - problems that have provable similar asymptotic (worst case) running times

- polynomial-time reductions → *
- example: given a large set of vectors and an input vector v, find the vector with closes cosine similarity to v. We can show how this problem relates to Nearest Neighbor Search.

* the technique to prove that two problems are equally complex

Course topics part I.

- Network flow
    - graph problem representing material throughput in a network
    - many real life problems can be modeled by it
  - notion of polynomial time reductions, certificates
- complexity classes, NP and NP-Complete

After this:

algorithms that find “good enough” solutions and are efficient.

- these are the algorithms that are used in practice

Approximation algorithms

Set Cover: Given a set of n items $U = \{u_1, u_2, \ldots, u_n\}$ and m subsets of these items $S_1, S_2, \ldots S_m$ find a minimum number of subsets, such that their union contains every element in U.

Algorithm 1: GreedySC$(U, S_1, \ldots S_m)$

```
1 X ← U /* uncovered elements in U                    */
2 C ← empty set of subsets;
3 while X is not empty do
4 |   Select S_i that covers the most items in X;
5 |   C ← C ∪ S_i;
6 |   X ← X \ S_i;
7 return C;
```

← implementation?

→ not optimal

what is nice about this algo?

→ simple

→ fast O(n) ← worst case

→ it finds a correct set cover

Course topics part II - approximation algorithms

Approximation algorithm: an algorithm to an optimization problem that finds a solution that is guaranteed to be at most x% off from the optimal min/max value.

GreedySC always finds a solution where the number of subsets is at most ~63% more than in the minimal solution.

Topics:

- many examples of problems and their approx algorithms
  - set cover, vertex cover, load balancing, makespan, bin packing, etc.

- methods how to compute their approximation ratios
  - we need to find ways to prove that our algorithms yield “good enough” outputs

Randomized algorithms

Randomized algorithm: an algorithm that there is randomness involved in the process

- randomized quicksort: in each iteration pick a random element as pivot
  - good expected runtime

standard trick:
- run this algo for a predetermined number of iteration
- if sorted → done!
- else: start over

- contention resolution: n processes are competing for access to the same database.

In each round only one process can gain access, how can we make sure (with high probability) that all processes get access in a reasonable number of rounds, if they can't communicate with each other?

- eliminates communication

→ in each iteration
→ every process flips a coin with prob p
→ if heads, make an attempt at access
→ else: don't

Randomized algorithms

Randomized algorithm: an algorithm that there is randomness involved in the process

- load balancing: jobs are arriving in a streaming fashion. Assign them to one of m machines, such that the worst case wait time for jobs to finish is as low as possible.
  - randomized algorithm:

for job pick a machine at random
→ in expectation this divides the jobs evenly

- power of two choices:

pick two rnd machines and assign the job to the less full

Efficient implementations using randomness

Greedy Set Cover:

- in each iteration select the set that covers the *most additional* items.
    - how can we find this set efficiently?

data structures:

- hash tables
    - hashing methods
    - applications, e.g. Bloom filters

data summary: sketching, counting, sampling

topics part III. - randomness

- algorithms that make random choices
  - expected running time
  - prove expected quality of solution
- tools for efficiency
- random walks on graphs
  - e.g. PageRank and co.