Examinable material

Course pages 2023 – 24

Algorithms 1

Syllabus | Course materials | Ticks | Recordings | For supervisors

Sorting. Review of complexity and O-notation. Triv
quadratic complexity. Review of merge sort and qu
their memory behaviour on statically allocated arr
Other sorting methods including sorting in linear t
statistics.

Strategies for algorithm design. Dynamic program
conquer, greedy algorithms and other useful parad

Data structures. Elementary data structures: point
queues, lists, trees. Binary search trees. Red-black
tables. Priority queues and heaps.

Course pages 2023 – 24

Algorithms 2

Syllabus | Course materials | Ticks | Recordings | For supervisors

Graphs and path-finding algorithms. Graph representations. Breadth-first and depth-first search. Single-source shortest paths: Bellman-Ford and Dijkstra algorithms. All-pairs shortest paths: dynamic programming and Johnson's algorithms.

Graphs and subgraphs. Maximum flow: Ford-Fulkerson method, Max-flow min-cut theorem. Matchings in bipartite graphs. Minimum spanning tree: Kruskal and Prim algorithms. Topological sort.

Advanced data structures. Binomial heap. Amortized analysis: aggregate analysis, potential method. Fibonacci heaps. Disjoint sets.

Be an algorithms chef, not an algorithms cook.
The exam questions will ask you to mix and match approaches and tricks you've learnt.

Inhabit

proof by induction

O/Ω analysis

$O/\Omega/\Omega$ gap

Records, pointers, etc.

dynamic programming

clever-greedy
divide-and-conquer

translation

lazy / fastidious / amortized

Φ defn + thm + use

Memorize

heapsort

binary heap

Dijkstra

BFS

DFS

MaxFlow

dynamic array

Know what terms mean

stable sorting; stack, queue, priority queue, dictionary;
linked list, doubly linked list; heap, search tree; hash table with chaining or open addressing

Grasp essence

selectsort, mergesort

bucket-based sorting

quicksort & median

topo sort

Prim, Kruskal

Bellman-Ford, dynamic programming

matchings

hash table

binomial heap

DisjointSet

B-tree, red-black tree

Appreciate

Johnson

FibHeap

Recall the O/Ω language we use:

"My alg is $O(n^2)$" or "My alg's worst-case is $O(n^2)$" means:

$\exists K>0,\ n_0 \quad \forall n \geq n_0 \quad \forall$ inputs of size n : cost $\leq Kn^2$

"My alg's worst-case is $\Omega(n^2)$" means:

$\exists \delta>0,\ n_0 \quad \forall n \geq n_0 \quad \exists$ input of size n : cost $\geq \delta n^2$

"The problem's worst case is $\Omega(n \log n)$" means:

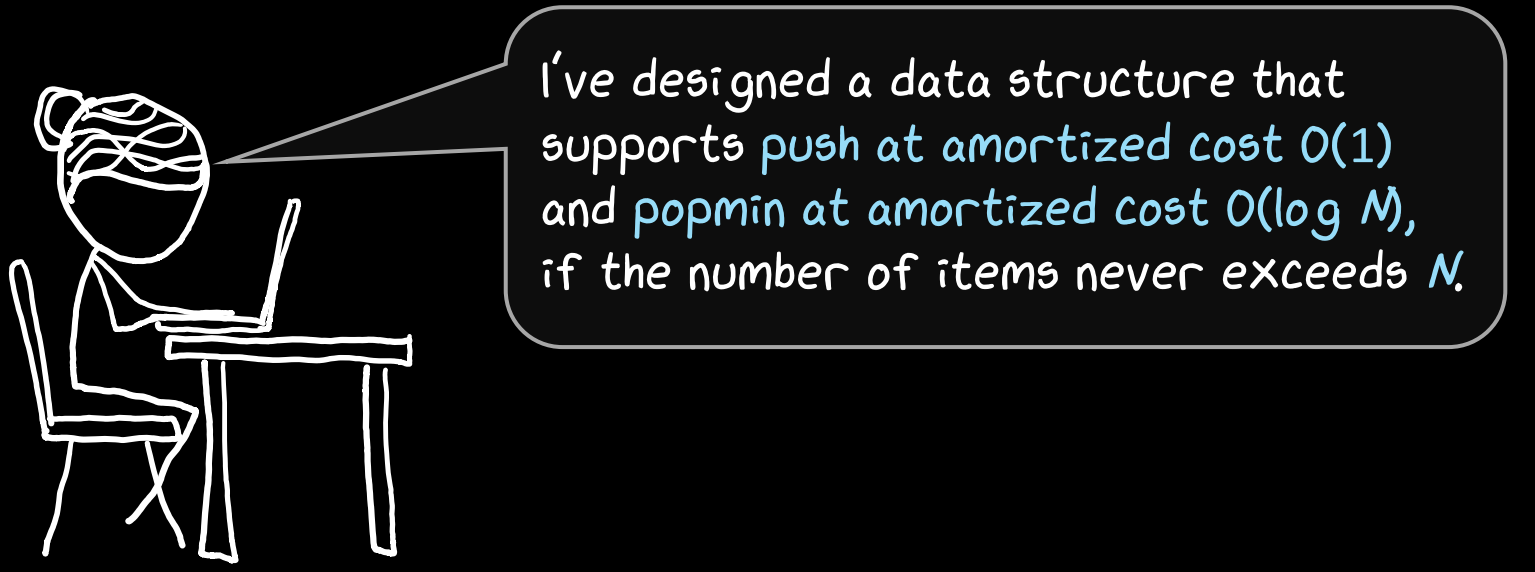
$\exists \delta>0,\ n_0 \quad \forall n \geq n_0 \quad \forall$ alg $\quad \exists$ input of size n: cost $\geq \delta n \log n$.

If the O and Ω for our algorithm don't match, we haven't understood it properly. Maybe our O bound is wasteful; maybe we haven't spotted which input is actually the hardest for our algorithm.

If our O for our algorithm doesn't match our Ω for the problem, either we have a wasteful algorithm, or we haven't spotted which input is actually the hardest for any algorithm.

Both of these are fruitful places to look for improvements in our algorithm / our understanding.

What does it mean to find matching O/Ω bounds in amortized analysis? [See ex.sheet 6 q.7]



This means:

$\exists\ K > 0,\ N_0 \quad \forall\ N \geq N_0 \quad \forall$ sequences of ops (s.t. # items always $\leq N$, and starting from empty) cost $\leq K\,(\#\text{push} + \#\text{popmin} \cdot \log N)$

There's no convenient Ω-style notation for talking about a $\exists$ bound for worst-case inputs.

Nonetheless, it's useful to actually try to find a sequence of operations that hits the proposed upper bound. If we can't find one, perhaps we've missed an opportunity for getting a better amortized design.

You may be asked to “explain” or “prove” new results.

We’ve seen several proofs in this course. Take them as a guide for the level of detail that’s expected in your proofs. You should memorize some of the proofs, and for the others you should grasp their essence.

Memorize
including the proof / analysis

heapsort
binary heap
Dijkstra
BFS
DFS
MaxFlow
dynamic array

Grasp essence

topo sort
Bellman-Ford
matchings

Exam questions will use keywords to indicate what level of detail is required:

explain why — show — show carefully — prove

could mean either.

"Give an intuitive explanation / do a calculation"

"Give a rigorous proof, same detail as in lecture notes"

Modify the standard BFS algorithm so that it returns *all* shortest paths from s to t.

```
01  from collections import deque

02  def shortest_paths(g, s, t):
03      all_shortest_paths = []  # all shortest paths from s to t
04      distance = {s: 0}        # shortest distance to each node
05      visited = set([s])
06      queue = deque()
07      queue.append((s, [s]))

08      while queue:
09          (v, path) = queue.popleft()

10          if v == t:
11              all_shortest_paths.append(path)
12              continue

13          for u in g.get(v, []):
14              if u not in visited or distance[u] == len(path):
15                  visited.add(u)
16                  distance[u] = len(path)
17                  queue.append((u, path + [u]))

18      return all_shortest_paths
```

AC

I have tested the algorithm and can confirm it works as expected.
Can you formally prove that this algorithm is correct

The correctness of the algorithm hinges on two fundamental claims:

Claim 1: The algorithm identifies paths that are shortest.

Claim 2: The algorithm exhaustively finds all such shortest paths, without exclusion.

correctness means:

{ paths returned by the algorithm } (COMPUTED) = { shortest paths from s to t } (PLATONIC)

alg. terminates

RTP
- { paths returned } ⊆ { shortest paths } CLAIM 1
- { shortest paths } ⊆ { paths returned }. CLAIM 2
- termination

Claim 1: The algorithm identifies paths that are shortest.

Invariant: For every vertex v reached during the BFS, the path discovered to v from s is the shortest path possible.

- *Base Case:* Initially, the path from s to itself is $[s]$, trivially the shortest.
- *Induction Step:* Assuming BFS has found the shortest paths to vertices within distance d from s, any vertex v reached at distance $d + 1$ is done by extending a shortest path of length d by one edge, thus maintaining the shortest path property.

QUESTION. What is involved in a proof by induction?

CLAIM: $P(n)$ is true for all $n \in \mathbb{N}$.

- Base case is prove $P(0)$
- Inductive step is prove $P(n) \Rightarrow P(n+1)$.

In any proof by induction, we have to make it clear what our induction is over. In other words, what is n? And we have to spell out our inductive hypothesis $P(n)$.

A common type of induction is "induction over program execution". Imagine there's an assertion at some line of code, and consider $P(n)$ to be "At the nth time we reach this line, my assertion is true." We call the assertion an *invariant*. In lectures, I described this as the *breakpoint proof strategy*.

There are two common types of induction in graph algorithms. Let's compare them, as applied to proving Dijkstra's algorithm correct.

BREAKPOINT PROOF

Invariant: When a vertex is popped from the priority queue, its computed distance is correct.

- *Base case:* trivially true for the start vertex
- *Inductive step:* proved by contradiction

In more detail:

Let the graph's vertices be $v_1, v_2, \ldots$ ordered by when the algorithm popped them. My inductive claim is

P(n): at the instant I popped v_n, v_n.distance is correct.

This is the start of a proof by induction of the invariant "assert: v.distance is correct" placed just after the line where v is popped.

ALONG-A-PATH PROOF

Consider any vertex v, and consider a shortest path from the start vertex to v.

Induction claim: For all vertices on this path, the computed distance is correct.

- *Base case:* trivially true for the start vertex
- *Inductive step:* assuming we correctly set u.distance, we'll correctly set the distance of all u's neighbours

In more detail:

Consider a shortest path from s to t, call it

$s = v_1 \to v_2 \to \cdots \to v_m = t$.

My inductive claim is

P(n): v_n.distance is correct, when the algorithm terminates. where n is distance along the path

This type of induction is used for the proof of Bellman–Ford, and also in ex4q15 to show that Dijkstra's algorithm can still be correct on graphs with negative weights.

NOT COVERED IN LECTURE

Note: for the along-a-path induction, when we actually try to prove the inductive step $P(n) \Rightarrow P(n+1)$, we find that our proof only works if we're running the algorithm all the way to completion. It doesn't work for the version of Dijkstra that has a target vertex, and stops as soon as it reaches that target, as this dodgy proof illustrates.

Exam question. Let `dijkstra_path`(g,s,t) be an implementation of Dijkstra's shortest path algorithm that returns the shortest path from vertex s to vertex t in a graph g. Prove that the implementation can safely terminate when it first encounters vertex t.

DODGY PROOF IN THE STYLE OF CHATGPT:

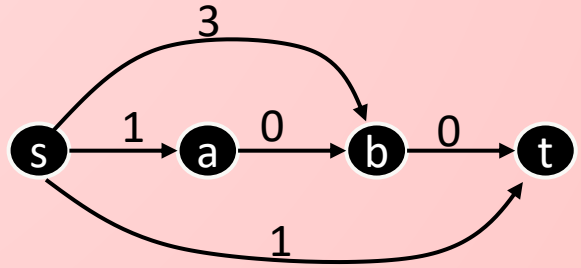
Consider a shortest path from s to t.

Invariant: Every vertex along this path has the correct computed distance.

- *Base case:* The algorithm sets s.distance=0, which is correct.
- *Induction step:* Assume that, for a vertex u on the path, its computed distance is correct. When visiting neighbours of u, including in particular the next vertex along the shortest path, it will set distance correctly.

Since the algorithm computes t.distance correctly, it can safely terminate when it reaches t.

This proof is incorrect. It fails on the example below, if we pop in order [s,t]. The induction step is written out with so little detail that it's easy to gloss over its bugginess.



Claim 1: The algorithm identifies paths that are shortest.

Invariant: For every vertex v reached during the BFS, the path discovered to v from s is the shortest path possible.

- *Base Case:* Initially, the path from s to itself is $[s]$, trivially the shortest.
- *Induction Step:* Assuming BFS has found the shortest paths to vertices within distance d from s, any vertex v reached at distance $d + 1$ is done by extending a shortest path of length d by one edge, thus maintaining the shortest path property.

This purports to be a proof by induction. But what actually *is* its inductive claim? Whenever we give a proof by induction, we **must** make clear what our induction claim is, and what the induction is over. Here, the word "invariant" suggests induction on program execution (i.e. a breakpoint proof), but the text of the inductive step suggests an induction over d ...

$P(d)$: for all vertices v at distance d from s, whenever (v, p) is pushed into the queue then p is a shortest path to v.

This is just my guess. In particular, I'm guessing what is meant by "the path discovered". The reader should not have to guess! And, if this is indeed the inductive claim, I can't actually see how to prove that $P(d) \Rightarrow P(d + 1)$!

Alternatively, perhaps ChatGPT had in mind a proof by induction on program execution, i.e. a "breakpoint" proof, establishing an invariant. What invariant might it have in mind? It takes creativity to come up with a good invariant! Perhaps an assertion just after line 9, with the corresponding inductive claim:

$P(n)$: at the nth time we reach this assertion, it is true

See CLRS lemma 22.3 for a proof along these lines of the claim that BFS is correct.

```
01  from collections import deque

02  def shortest_paths(g, s, t):

03      all_shortest_paths = []  # all shortest paths from s to t
04      distance = {s: 0}        # shortest distance to each node
05      visited = set([s])
06      queue = deque()
07      queue.append((s, [s]))

08      while queue:
09          (v, path) = queue.popleft()

10          if v == t:
11              all_shortest_paths.append(path)
12              continue

13          for u in g.get(v, []):
14              if u not in visited or distance[u] == len(path):
15                  visited.add(u)
16                  distance[u] = len(path)
17                  queue.append((u, path + [u]))

18      return all_shortest_paths
```

ASSERT that path is a shortest path to v, and that queue consists of vertices at distance dist(s to v) or dist(s to v) + 1, ordered by distance

To prove P(n) ⇒ P(n+1) we'll need to reason explicitly about all the pieces of logic (popleft, append right, 'if' condition on line 14).

Claim 2: The algorithm exhaustively finds all such shortest paths, without exclusion.

- *Revisitation Strategy:* Vertices can be revisited if reached by a path whose length equals the shortest distance previously recorded to that vertex. This is crucial for identifying multiple branching paths that occur within the graph's structure, ensuring no shortest path is overlooked.

- *Path Accumulation and Shortest Distance Tracking:* The algorithm maintains a distance dictionary, tracking the shortest distance to each vertex. When a path to a vertex equals this shortest distance, the path is considered valid and added to the collection of paths leading to that vertex. This approach ensures the algorithm accumulates all shortest paths to each vertex, particularly focusing on the target vertex t.

- *Ensuring Completeness:* By combining BFS's layer-by-layer exploration with the condition-based revisitation of vertices, the algorithm guarantees that all possible branching paths from s to t, which do not exceed the shortest path length, are explored. This exploration strategy ensures that no shortest path is missed, fulfilling the requirement for a complete search.

It's hard to even make sense of this purported "proof". Is it an induction? If so, what's the induction over? I'd prove it by induction over d, the distance from s, with an inductive claim

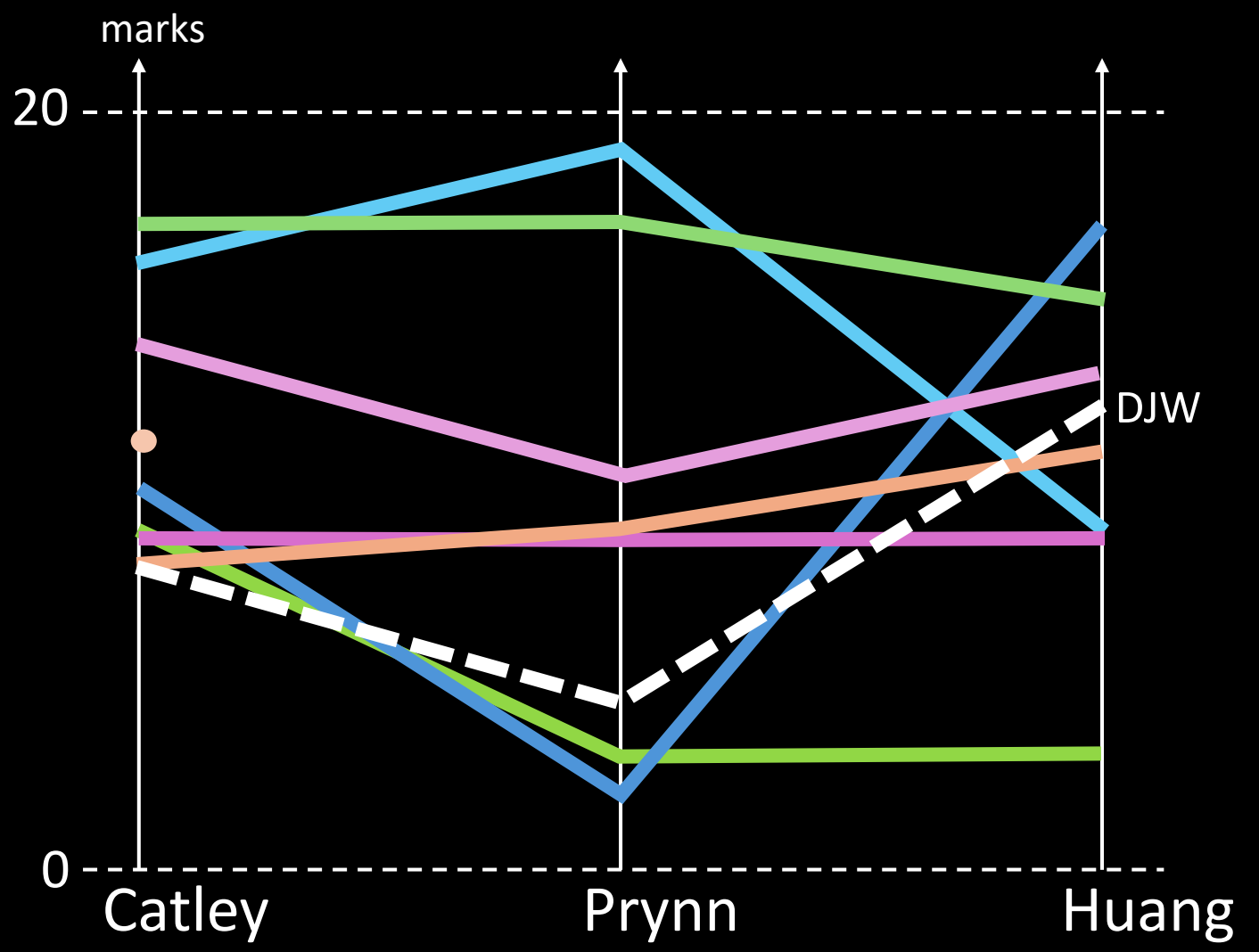
$P(d)$: for all vertices v at distance d from s, for all shortest paths p from s to v, we eventually push (v, p) into the queue.

This is faintly hinted at in ChatGPT's phrase "BFS's layer-by-layer exploration". Phrases like "is crucial for" and "this approach ensures" are waffle, and have no place in a rigorous proof.

Examiner's report for ChatGPT's proof

Recognize that this is a $A \subseteq B, B \subseteq A$ problem	3 / 4	Correctly identified the nature of the proof required, but omitted termination.
Proof that alg. finds *only* shortest paths	4 / 8	Identified that we need an induction, but missing detail on the exact inductive claim and the proof of the inductive step.
Proof that alg. finds *all* shortest paths (and terminates)	2 / 8	Mentioned some relevant features of what would be needed in a proof, but didn't even identify an induction.

Note: the task of proving the algorithm correct requires original thinking about how to set up the inductions, and is much harder than you'd face in an exam!



Each coloured line represents a different assessor's marks

How to learn a proof

PASSIVE LEARNING	ACTIVE LEARNING	REFLECTIVE LEARNING
▪ read it / watch it	▪ copy it out ▪ hide part of the proof, and try to fill it in ▪ identify the “beats” of the argument	▪ refactor it to be more elegant ▪ see if it still works when we tweak the problem statement