CS231 Algorithms | **Handout # 16**
Prof. Lyn Turbak | **April 26, 2022**
Wellesley College | ***Revised May 5, 2022***

Problem Set 5
Dueish: Wednesday May 4

Corrections & clarifications:

- [2022/05/05] (1) The original submission template ps5-submission.tex had a header with PS4 that should be PS5; (2) For Problem 1 there have been many questions about parts (c) and (d). To understand what these questions are asking, be sure to study the **revised** version of the Stable Matching slides (which I further tweaked today), especially slides 3, 4, 7, and 8. Slide 8 (which is new) explains why all executions of the Gale-Shapley algorithm return the same stable matching (KT 1.7). Also note that there can be multiple stable matchings for a given set of preferences (see Example 3 on slide 4). (3) In Problem 2, (a) you need to represent a single person by more than one node in the directed graph and (b) the topological sort algorithm has a helpful property (see slide 13 of the Directed Graphs slides).

Important notes about PS5 and PS6

- Because there is already a **lot** of CS231 work on your plate, and PS5 was posted later than expected, PS5 will be the last **required** pset.
- There will be a short PS6, but it will be **completely optional**. PS6 will be posted by Mon May 2 and dueish on Mon May 9. It will have only a few problems, mostly related to dynamic programming. You will not be required to submit any problems from PS6, but any points you earn for problems you **do** submit will count in both the numerator and denominator of your course pset score and can only raise your grade. **However, the material covered by PS6 will be covered in Solo C, so at the very least you will want to study the PS6 solutions.**
- **This PS5 pset uses features for drawing graphs that require a different LaTeX compiler to be used in Overleaf.** To compile this file in Overleaf, follow the instructions at https://www.overleaf.com/learn/how-to/Changing_compiler for changing the Overleaf LaTeX compiler to XeLaTeX

Starting this pset

The LaTeX source code for this pset can be found in the zip file `ps5.zip`. The unzipped folder includes a file `ps5-submission.tex` that you should use for your submission. It also includes an updated CS231 packages/macros file `231-v1.5.tex` and the source code for this handout (`ps5.tex`).

This assignment involves analyzing algorithms and specifying algorithms via pseudocode.

For your submission, you should edit `ps5-submission.tex` to include your solutions to each problem between the `\begin{problemSoln}` and `\end{problemSoln}` commands, This will color your solution in brown to distinguish it from the problem specification. You can keep the problem specification, but you don't have to; you can delete it or comment it out (using `%` or `\begin{comment}` and `\end{comment}`).

Contact me if you have trouble understanding aspects of the LaTeX code that you need to use in your submission.

Submitting this pset

Use whatever implementation of LaTeX you're using (Overleaf is recommended) to generate a PDF file from `ps5-submission.tex`, and submit this to Gradescope. Students working in pairs need only submit one PDF, under either member's Gradescope account; be sure to include the names of both partners in the boxed area at the top of the file. As part of submission, you will need to indicate which pages have the answers to which problems and subproblems.

Problem 1 [12]: Stable Matching Questions

Each of the following two parts is a statement. Decide whether you think the statement is true or false. If it is true, give a brief explanation why. If it is false, give a counterexample. As part of answering these questions, you should review the Stable Matching lecture slides (which have been updated since the lecture was presented).

a [4] True or false? In every instance of the Stable Matching Problem, there is a stable matching containing a pair (e, a) such that e is ranked first on the preference list of a and a is ranked first on the preference list of e.

Put your answer here

b [4] True or false? Consider an instance of the Stable Matching Problem in which there exists an employer e and an applicant a such that e is ranked first on the preference list of a and a is ranked first on the preference list of e. Then in every stable matching S for this instance, the pair (e, a) belongs to S.

Put your answer here

c [4] True or false? Because the Gale-Shapley algorithm nondeterministically chooses to process the next free employer with a nonempty queue, different executions of the algorithm can return different stable matchings. No proof is required here. You should refer to a result from the textbook/slides.

Put your answer here

d [4] True or false? Consider an instance of the Stable Matching Problem in which S is a stable matching. The Gale-Shapley algorithm can return S as a possible result. Recall from the textbook/slides that there can be multiple stable matchings for a given Stable Matching Problem. E.g., see Example 3 in slide 4.

Put your answer here

Problem 2 [15]: Processing Ethnography Data

You're helping a group of ethnographers analyze some oral history data they have collected by interviewing members of a village to learn about the lives of people who have lived there over the past two hundred years. From these interviews, they have learned about a set of n people (all of them now deceased), whom we'll denote as $P_1, P_2, ..., P_n$. They have also collected facts about when these people lived relative to one another. Each fact has one of the following two forms:

- for some i and j, person P_i died before person P_j was born; or
- for some i and j, the life spans of P_i and P_j overlapped at least partially.

Naturally, they are not sure that all these facts are correct; memories are not so good, and a lot of this was passed down by word of mouth. So what they'd like you to determine is whether the data they've collected is at least internally consistent, in the sense that there could have existed a set of people for which all the facts they've learned simultaneously hold.

Give an efficient algorithm that can verify the information collected by these ethnographers. In other words, it should report whether the events recorded by them are consistent or not.

Your answer should clearly describe the algorithm (you can write it in plain English sentences), the input given to the algorithm, and how the output is determined from the input. You can use as "black boxes" algorithm(s) presented in the Directed graph slides (which have been updated since the lecture).

Hints:

```
function PackTrucks($W$, $boxes$)
        ▷ $W$ is the weight limit and $boxes$ is a 1-indexed array of positive box weights in the
        ▷ order they should be shipped. Assume no single box has weight $> W$. Returns a an
        ▷ array of arrays, where each array is a list of box weights for the boxes on one truck.
    $trucksShipped \leftarrow []$                    ▷ Initialize array of shipped trucks to empty array
    $currentTruck \leftarrow []$                     ▷ Empty array of boxes for current truck being packed
    $currentWeight \leftarrow 0$                     ▷ Weight of boxes on $currentTruck$
    for all $b \in boxes$ do
        if $currentWeight + b > W$ then
            ▷ Current truck has reached weight capacity. Send it off and start to packing new one
            $trucksShipped@currentTruck$              ▷ Ship the now full currentTruck
            $currentTruck \leftarrow [b]$             ▷ Re-initialize to be new truck with its first box.
            $currentWeight \leftarrow b$
        else                                          ▷ Add box to current truck
            $currentTruck@b$
            $currentWeight \leftarrow currentWeight + b$
    return $trucksShipped$
```

Figure 1: An algorithm for packing trucks.

- This is a directed graph problem. How can you represent the facts as a directed graph? In particular, can you represent the birth and death of each person and the relationships between them? You'll need to represent a single person by more than one node in the directed graph.
- What property of the resulting graph determines whether or not the facts are consistent? How do you efficiently determine this property? Study slide 13 of the Directed Graphs slides.

Put your answer here

Problem 3 [12]: "Stay Ahead" Greediness In Shipping

You are consulting for a trucking company that does a large amount of business shipping packages between New York and Boston. The volume is high enough that they have to send a number of trucks each day between the two locations. Trucks have a fixed limit W on the maximum amount of weight they are allowed to carry. Boxes arrive at the New York station one by one, and each package i has a weight w_i. The trucking station is quite small, so at most one truck can be at the station at any time.

Company policy requires that boxes are shipped in the order they arrive; otherwise, a customer might get upset if a box that arrived in New York after their box ends up arriving in Boston earlier than their box.

At the moment, the company is using a simple greedy algorithm for their truck-packing algorithm: they pack boxes into trucks in the order they arrive, and whenever the next box does not fit (i.e., it would put the truck over the weight limit W), they send the truck on its way. Fig 1 presents pseudocode for their current algorithm.

But they wonder if they might be using too many trucks, and they want your opinion on whether the situation can be improved. Here is how they are thinking. Maybe they could decrease the number of trucks needed by sometimes sending off a truck that was less full, and in this way allow the next few trucks to be better packed.

Prove that the greedy algorithm PackTrucks uses the fewest possible trucks by showing that it "stays ahead" of any other solution. I.e., if the greedy algorithm fits boxes $b_1, ..., b_j$ onto the first t trucks and an alternative algorithm fits boxes $b_1, ..., b_k$ onto the first t trucks, then $k \leq j$.

The proof is by an induction expressed via the following two parts:

a [3]: Base Case

Prove that the "stay ahead" claim is true when the number of trucks $t = 1$

Put your answer here

b [9]: Inductive Case

Prove that if the "stay ahead" claim is true when the for a number of trucks $t - 1$, it is true for t trucks.

Put your answer here

Problem 4 [28]: Triathlon Scheduling

You are working as a camp counselor, and you're in charge of organizing activities for a set of campers. You've decided to organize a mini-triathlon exercise in which each contestant must first swim 20 laps of a pool, then bike 10 miles, and finally run 3 miles.

The plan is to send the contestants out in a staggered fashion, via the following rule: the contestants must use the pool one at a time. In other words, first one contestant swims the 20 laps, gets out, and starts biking. As soon as this first person is out of the pool, a second contestant begins swimming the 20 laps; as soon as he or she is out and starts biking, a third contestant begins swimming ... and so on.

Each contestant has a projected swimming time (the expected time they will take to complete the 20 laps), a projected biking time (the expected time it will take them to complete the 10 miles of bicycling), and a projected running time (the time it will take them to complete the 3 miles of running).

You want to figure out a good schedule for the triathlon — an order in which to sequence the starting times of the contestants. The **competition duration** of a schedule is the total time it takes all contestants to finish all three legs of the triathlon, assuming they each spend exactly their projected swimming, biking, and running times on the three parts. If the competition is assumed to start at time 0, the competition duration is the time at which the last contestant finishes the competition. Note that participants can bike and run simultaneously, but at most one person can be in the pool at any time.

You decide that you want the competition to be over as early as possible, so you want a schedule of contestant starting times that minimizes the competition duration. Your goal is to determine a greedy strategy for ordering the contestants so that the competition duration is as short as possible.

For thinking about this problem, assume that

- For a contestant c, their estimated swimming time is $s(c)$, their estimated biking time is $b(c)$, and their estimated running time is $r(c)$ (in minutes).
- When you order the contestants in a 1-indexed starting order, the contestant at index i has a swimming time s_i, biking time b_i, and running time r_i (in minutes).

a [12]: Incorrect strategies

You consider the following six greedy strategies for scheduling the contestants., only one of which is actually correct:

Strategy 1: Shortest swim time first: x can precede y in the schedule if $s(x) \leq s(y)$.

Strategy 2: Longest swim time first: x can precede y in the schedule if $s(x) \geq s(y)$.

Strategy 3: Shortest biking time + running time first: x can precede y in the schedule if $b(x) + r(x) \leq b(y) + r(y)$.

Strategy 4: Longest biking time + running time first: x can precede y in the schedule if $b(x) + r(x) \geq b(y) + r(y)$.

Strategy 5: Shortest total time first: x can precede y in the schedule if $s(x)+b(x)+r(x) \leq s(y) + b(y) + r(y)$.

Strategy 6: Longest total time first: x can precded y in the schedule if $s(x)+b(x)+r(x) \geq s(y) + b(y) + r(y)$.

Five of the above strategies are incorrect for minimizing the competition duration. For each these five strategies, develop a simple counterexample for that strategy involving particular swimming/biking/running times for **exactly two** contestants a and b such that ordering the contestants as specified by that strategy leads to a **longer** competition duration than the opposite ordering. For instance, below is one particular counterexample worth considering, but you will have to develop one or more additional counterexamples:

Contestant c	$s(c)$	$b(c)$	$r(c)$
Alex	5	55	35
Bailey	10	60	40

Note: Some counterexamples may work for more than one strategy.

Put your answer here

b [3]: Candidate strategy Let's give the name **candidate strategy** to the single strategy that remains after filtering out the five nonworking strategies from part (a). Show that the candidate strategy works correctly for all the examples you developed for part (a). (This doesn't prove that it is correct and will always work, but does provide some confidence that it may work.)

Put your answer here

c [13]: Prove that the candidate strategy is optimal

Do this by using an **exchange argument**, which is a proof technique whose key property in this context is:

> (Removing An Inversion Maintains Optimality) Assume that S_O is an optimal schedule of triathlon contestants (i.e., it minimizes competition duration), but the contestant ordering is not consistent with the ordering specified by the candidate strategy. Then there is at least one 1-based index in the schedule such that consecutive contestants c_i and c_{i+1} are not correctly ordered with respect to the candidate strategy. This consecutive pair of contestants c_i and c_{i+1} is said to be an **inversion**. Then the alternative schedule S'_O that results by removing the inversion by swapping c_i and c_{i+1} is still an optimal schedule.

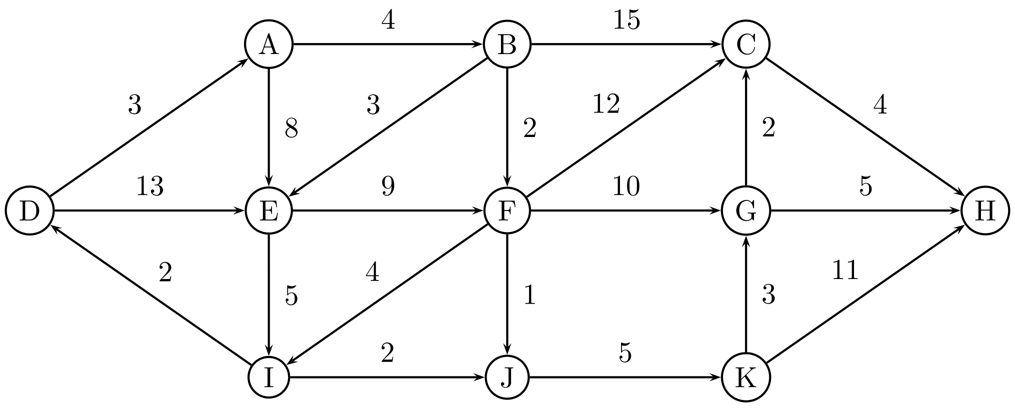
i. [10] Prove that the candidate strategy satisfies the property (Removing An Inversion Maintains Optimality)

Put your answer here

ii. [3] Argue that repeated application of the property (Removing An Inversion Maintains Optimality) to an optimal solution S_O that does not satisfy the candidate strategy leads to an optimal solution S that satisfies the candidate strategy.

Put your answer here

Problem 5 [40]: Shortest Paths and Minimum Spanning Trees
Suppose you are given the following weighted, directed graph G:



a [20]: Diijkstra's Single-Source Shortest Path Algorithm

Use Dijkstra's single-source shortest path algorithm to determine the shortest paths from D to each of the other vertices. Use the algorithm and table-based notation described in the Shortest Path lecture slides to show the progress of the algorithm and the final shortest path tree (with directed edges) and annotate each vertex by its shortest path distance from D.

Flesh out the skeletons of the table and tree provided in the problemSoln environment, and pay attention to the LaTeX formatting notes there.

Below is a skeleton of a table that summarizes the steps of Dijkstra's algorithm for the graph G and source vertex D. Your goal is to flesh out this table.

Each row shows the following at 1-indexed step i:

- The pair $m, d_D[m]$, where m is the value v with minimal distance-from-D estimate $d_D[v]$ that is the returned by the PQDeleteMin operation in Dijkstra's algorithm on slide 7 of the Shortest Path slides. This entry is N/A (not applicable) at step 1, which shows the state variables after executing INITIALIZESINGLESOURCE (slide 5).
- For each vertex v in G a pair $d_D[v]/parent[v]$, where ∞ represents infinite distance (no path yet from D to v) and ? means "no parent" (written **none** in the algorithm).
  - Use `\IN{}` ("infinity/none") as a short way to write the pair $\infty/?$
  - In the step where a vertex v is dequeued and added to *shortest* with distance $d_D[v]$, the pair should be shown in **green bold** by wrapping it in `\Sh{}` (which stands for "shortest"). Moreover, this pair should **not** be shown in any later rows to reduce clutter.
  - After a RELAX step has reduced the distance estimate $d_D[v]$ for a vertex v, the pair should be shown in *blue italics* by wrapping it in `\Rx{}` (which stands for "relax") to highlight when this estimate decreases.

The first 3 rows have been fleshed out for you; you should flesh out the cells of the remaining rows, which have placeholders that help you distinguish the cells when editing the LaTeX. Note that cells below the **green bold** entries should remain empty.

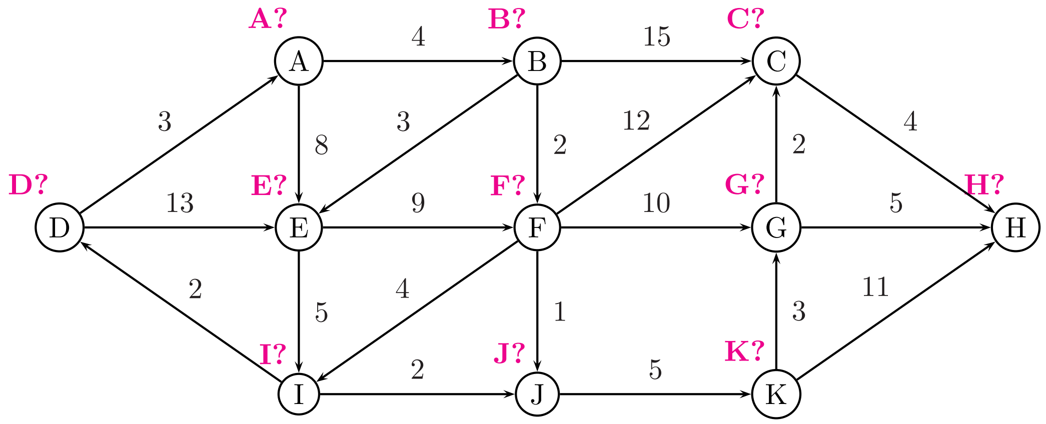
Table for Dijkstra's algorithm

Step	$m, d_D[m]$	A	B	C	D	E	F	G	H	I	J	K
1	N/A	∞/?	∞/?	∞/?	0/?	∞/?	∞/?	∞/?	∞/?	∞/?	∞/?	∞/?
2	D,0	*3/D*	∞/?	∞/?	**0/?**	*13/D*	∞/?	∞/?	∞/?	∞/?	∞/?	∞/?
3	A,3	**3/D**	*7/A*	∞/?		*11/A*	∞/?	∞/?	∞/?	∞/?	∞/?	∞/?
4	m4		B4	C4		E4	F4	G4	H4	I4	J4	K4
5	m5		B5	C5		E5	F5	G5	H5	I5	J5	K5
6	m6		B6	C6		E6	F6	G6	H6	I6	J6	K6
7	m7		B7	C7		E7	F7	G7	H7	I7	J7	K7
8	m8		B8	C8		E8	F8	G8	H8	I8	J8	K8
9	m9		B9	C9		E9	F9	G9	H9	I9	J9	K9
10	m10		B10	C10		E10	F10	G10	H10	I10	J10	K10
11	m11		B11	C11		E11	F11	G11	H11	I11	J11	K11
12	m12		B12	C12		E12	F12	G12	H12	I12	J12	K12

Tree for Dijkstra's algorithm

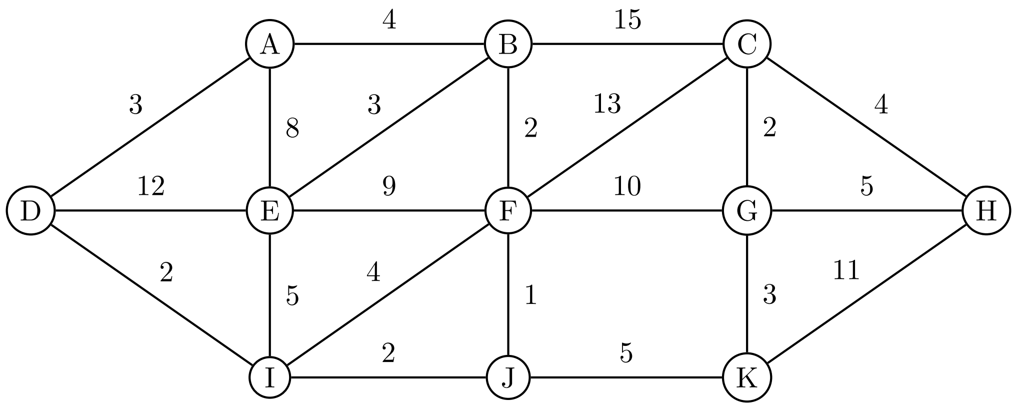
Below is a copy of the graph G from above. You should modify it in two ways:

1. Highlight the directed edges that appear in the shortest-path tree from D by changing each edge in this tree from `\dirEdge` to `\dirTreeEdge`
2. Annotate each vertex v with the distance from D using the shortest path tree. To do this, replace the **v?** in the call `\dist{v}{v?}` for each vertex v.



b [9]: Kruskal's Minimum Spanning Tree Algorithm

Now we consider an undirected graph H that's a version of G in which all the edge directions have been erased:

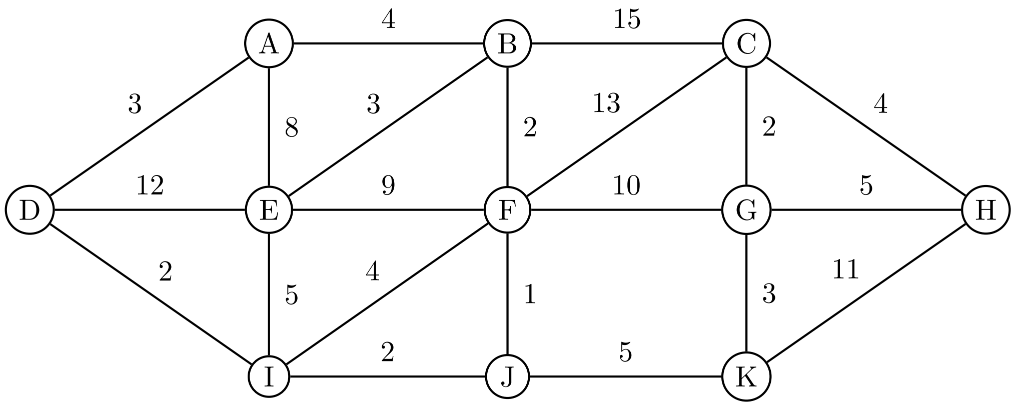


Use Kruskal's algorithm from the Minimum Spanning Tree slides to construct a minimum spanning tree for H. When there's a choice between adding edges with the same weight to the current forest, choose the one that has the least vertex in alphabetical order.

Below is a table showing the order in which edges are added to the MST forest using Kruskal's algorithm. Your goal is to flesh out the missing table entries. Each entry should use the notation *edgeWeight:*$\{vertex1, vertex2\}$, where the vertices in a set are in alphabetical order, e.g., 3:$\{B, E\}$.

index	1	2	3	4	5	6	7	8	9	10
w:(u,v)	1?	2?	3?	4?	5?	6?	7?	8?	9?	10?

Below is a copy of the undirected graph H. Modify it to show the MST from Kruskal's algorithm by changing each `\undEdge` to `\undTreeEdge` for each MST tree edge to display it as a thicker edge.



c [5]: Prims's MST Algorithm

Use Prims's algorithm fom the Minimum Spanning Tree slides to construct a minimum spanning tree for H starting at node **G**. When there's a choice between adding edges with the same weight to the current tree, choose the one that has the least vertex in alphabetical order. You should get the same MST as in part (b).

Below is a table showing the order in which edges are added to the MST tree starting at vertex **G** using Prims's algorithm. Your goal is to flesh out the table entries. Each entry should use the notation *edgeWeight:*$\{vertex1, vertex2\}$, where the vertices in a set are in alphabetical order, e.g., 3:$\{B, E\}$.

index	1	2	3	4	5	6	7	8	9	10
w:(u,v)	1?	2?	3?	4?	5?	6?	7?	8?	9?	10?

d [6]: Reverse Delete MST Algorithm

Use the Reverse Delete algorithm from the Minimum Spanning Tree slides to construct a minimum spanning tree for H/ When there's a choice between deleting edges with the same weight from the current connected graph (without disconnecting it!), choose the one that has the least vertex in alphabetical order. You should get the same MST as in part (b).

Below is a table showing the order in which edges are deleted from the MST tree using the Reverse Delete algorithm. Your goal is to flesh out the table entries. Each entry should use the notation *edgeWeight:*$\{vertex1, vertex2\}$, where the vertices in a set are in alphabetical order, e.g., 3:$\{B, E\}$.

index	1	2	3	4	5	6	7	8	9	10	11
w:(u,v)	1?	2?	3?	4?	5?	6?	7?	8?	9?	10?	11?