CS161 Final Exam

March 23, 2023

Instructions:

- This exam's duration is **180 minutes**.
- You may use two two-sided sheet of notes that you have prepared yourself. **You may not use any other notes, books, or online resources. You may not collaborate with others.**
- This exam has **5** problems over **20** pages, worth a total of **65** points. Please check that you have all of them both before beginning and before submission.
- **If you have a clarification question about the exam:** There are TAs outside the exam room to answer questions.
- There are extra pages at the end of the exam for scratch work. You may use it for additional space on problems if you need, but please indicate in the original space for the relevant problem if you have work on the back pages that you want to be graded, and label your work clearly.
- Please do not discuss exam questions or answers with other students until solutions are posted.
- You may cite any result we have seen in lecture without proof unless otherwise stated.
- **Please write your name at the top of all pages.**

Advice: If you get stuck on a problem, move on to the next one. Pay attention to how many points each problem is worth. Read the problems carefully.

The following is a statement of the Stanford University Honor Code:

1. *The Honor Code is an undertaking of the students, individually and collectively:*
   (1) *that they will not give or receive aid in examinations; that they will not give or receive unpermitted aid in class work, in the preparation of reports, or in any other work that is to be used by the instructor as the basis of grading;*
   (2) *that they will do their share and take an active part in seeing to it that others as well as themselves uphold the spirit and letter of the Honor Code.*
2. *The faculty on its part manifests its confidence in the honor of its students by refraining from proctoring examinations and from taking unusual and unreasonable precautions to prevent the forms of dishonesty mentioned above. The faculty will also avoid, as far as practicable, academic procedures that create temptations to violate the Honor Code.*
3. *While the faculty alone has the right and obligation to set academic requirements, the students and faculty will work together to establish optimal conditions for honorable academic work.*

By signing your name below, you acknowledge that you have abided by the Stanford Honor Code while taking this exam.

Signature: _____

Name: _____

SUNetID: _____

1 True or False (6 pt.)

For each of the following questions, select whether the given statement is True or False. **No explanation is required for True/False questions.** Please clearly mark your answers; if you must change an answer, either erase it thoroughly or very clearly indicate which answer you have chosen. **Ambiguous answers will be marked incorrect**.

1.1. **(1 pt.)** Floyd-Warshall is always asymptotically slower than Bellman-Ford (for finding shortest paths between all pairs of nodes).

True False

Solution

False
Bellman-Ford can take $O(n^4)$ time when $m = O(n^2)$, because the runtime becomes $O(n^2 m)$.

1.2. **(1 pt.)** Kruskal's MST algorithm is an example of a greedy algorithm that grows a forest.

True False

Solution

True
Kruskal can grow separate trees (aka a forest) that eventually join into one tree.

1.3. **(1 pt.)** You can detect whether a graph $G = (V, E)$ is bipartite in $O(|V| + |E|)$ time.

True False

Solution

True
Use BFS.

1.4. **(1 pt.)** Any algorithm that can be solved optimally using Dynamic Programming can also be solved using a Greedy algorithm

True False

Solution

False
Greedy algorithms only work when we have especially nice sub-structures, whereas DP is more robust.

1.5. **(1 pt.)** The SCCs returned by Kosaraju's algorithm can be different based on which node the algorithm starts at.

True False

Solution

False
SCCs of a graph can be determined based on graph structure alone, and do not depend on a starting point.

1.6. **(1 pt.)** Amortized runtime of insertion is $O(\log(n))$ for all binary search trees.

True False

Solution

False
Consider the case where the tree is a linked list.

2 Multiple Choice (10 pt.)

For each of the following questions, select **all correct answer choices**. **No explanation is required.** Please clearly mark your answers; if you must change an answer, either erase it thoroughly or very clearly indicate which answer you have chosen. **Ambiguous answers will be marked incorrect**.

2.1. **(2 pt.)** Suppose that H is a hash family of size s, and we choose h randomly from H. Which of the following options imply by themselves that an item can be found using h in constant time? Let n be the number of buckets h hashes to, and let u be the size of the universe of keys.

A) $\mathbb{P}(h(u_i) = h(u_j)) \leq \frac{1}{n^2}$ when $u_i \neq u_j$

B) $\mathbb{P}(h(u_i) = h(u_j)) \leq 0.0001$ when $u_i = u_j$

C) For every bucket b, $\mathbb{E}$[number of items in b] ≤ 2

D) For any item u_i, $\mathbb{E}$[number of items in u_i's bucket] ≤ 2

Solution

A, D
(A) implies universal hash family, (D) implies that for any item there can be at most one pair-wise collision.

2.2. **(2 pt.)** Which of the following algorithms can be used to find shortest paths on all graphs with negative edges?

A) Dijkstra

B) Floyd-Warshall

C) BFS

D) DFS

E) Bellman-Ford

Solution

B, E
See lecture slides for proof.

2.3. **(2 pt.)** Recall the 0/1 Knapsack problem from lecture, where the objective is to maximise the value of the knapsack given that we have a single copy of each item. Select the options that are *false*.

A) The optimal solution cannot always be obtained using a greedy algorithm.

B) A smaller capacity Knapsack is sufficient optimal sub-structure to solve this problem if using DP.

C) There is a DP solution for this algorithm that can be easily modified to return the contents of the knapsack along with the value of the knapsack.

D) Bottom-up algorithms are usually recursive whereas top-down algorithms are usually iterative.

Solution

B, D
The optimal substructure should have both a smaller or same size knapsack, and a list of items that have been considered so far.

2.4. **(2 pt.)** The shortest path from a source node to all other nodes in a graph will remain unchanged if:

A) A positive constant value is added to all edge weights.

B) A positive constant value is multiplied by all edge weights.

C) A positive constant value is subtracted from all edge weights.

D) All edge weights are divided by a positive constant value.

Solution

B, D
In B, D path costs difference do not depend on the number of edges the path. Thus, the relative ordering of path lengths will stay the same.

2.5. **(2 pt.)** Which of the following are not characteristics of wicked problems? Select all that apply.

A) Their solutions are either correct or incorrect.

B) They have no definitive and exhaustive formulation.

C) They are symptomatic of, and interconnected with, other problems.

D) Solutions may be tested as many times as is necessary.

E) None of the above.

Solution

A, D.
The 10 features of a wicked problem are:

- No definitive formulation
- Unique
- Symptoms of other problems
- No stopping rule
- Solutions are not true-or-false, but good-or-bad
- No enumerable (or an exhaustively describable) set of potential solutions, or well-described set of permissible operations
- No immediate and no ultimate test of a solution
- No rule or procedure to deal with discrepancy
- Every solution is a "one-shot operation
- The planner has no right to be wrong

3 Short Answers (16 pt.)

3.1 Central Location (3 pt.)

Suppose you are given a connected directed weighted graph with vertices V. Devise an algorithm to find the node u in V which minimizes the quantity $\Sigma_{v \in V, u \neq v} \text{dist}(u, v)$, where $\text{dist}(u, v)$ returns the length of the shortest path between u and v. Your algorithm should run in time $O(n^3)$.

[We are expecting: Pseudocode OR an English description of your algorithm, along with a proof of runtime.**]**

Solution

Use Floyd Warshall to find the distance between every pair of nodes. This runs in time $O(n^3)$. Then loop through each node and find the average distance between it and the other nodes. This takes time $O(n^2)$. Thus our total runtime is $O(n^3)$.

3.2 Longest Path (3 pt.)

Suppose you are given a weighted, directed, acyclic graph. Devise an algorithm to find the longest path between two vertices a and b.

[We are expecting: Pseudocode OR an English description of your algorithm. No proof of runtime is necessary, but you should pick an algorithm that is relatively efficient.**]**

Solution

Flip the sign on all of the edge weights, and then run Bellman Ford. We are guaranteed that no negative cycles exist on our new graph because no cycles at all exist on it. Thus we will find the shortest path when using negative weights, which corresponds exactly to the longest path on the original graph.
(Note that the concept of longest path is well-defined - we cannot create an arbitrarily large path because there are no cycles).

3.3 Species Extinction (3 pt.)

Suppose you are given a list of n species and a list of m statements of the form "if species x goes extinct, then species y will also go extinct." Construct an algorithm to find the maximum set S of species such that if any one species in S goes extinct, then all the rest of the species in S will eventually go extinct. Your algorithm should run in time $O(m+n)$.

For example, if your species are [Rhinos, Hippos, Hawks], and your extinction relationships are [(Hawks extinct $\rightarrow$ Rhinos extinct), (Rhinos extinct $\rightarrow$ Hippos extinct), (Hippos extinct $\rightarrow$ Hawks extinct)], then you should return the set S = [Rhinos, Hippos, Hawks].

[We are expecting: Pseudocode OR an English description of your algorithm, along with a proof of runtime.**]**

Solution

First, construct a directed graph G with the vertices being each species, and an edge from x to y if we were told the x going extinct will cause y to go extinct. See that our maximum set S from the problem statement is equivalent to a strongly connected component of G. Thus we can find all strongly connected components using the $O(m+n)$ strongly connected components algorithm from lecture, and then from these pick the largest SCC.

3.4 Baby Hashing (3 pt.)

Suppose that you are given access to an unlimited number of independent random hash functions that each output 5 digit numbers such that for any two distinct elements x, y in your domain and each randomly sampled hash function h, $p(h(x) = h(y)) \leq 1/10$. One call of $h(x)$ runs in time $O(1)$. Construct a new random hash function h that outputs upto $O(n)$-digit numbers such that $p(h(x) = h(y)) \leq 2^{-n}$, and such that $h(x)$ can be computed in time $O(n)$.

[We are expecting: A definition of your hash function, along with a brief explanation for why it works.**]**

Solution

Randomly select n hash functions $h_1, ...h_n$ which satisfy the property stated in the problem statement. Let $H(x) = h_1|h_2|...h_n$ (— is the concatenate symbol). See that if $H(x) = H(y)$, then $h_i(x) = h_i(y)$ for $1 \leq i \leq n$. The probability of this happening is $(1/10)^n$. Thus the probability that $H(x) = H(y)$ is $(1/10)^n \leq 2^{-n}$, as desired.

3.5 DP runtime (4 pt.)

Consider the DP algorithm below:

```
global array A
# initialize A as an array of length n+1, all values None
def fooDP(x):
  if x < 1:
    return 0
  if A[x] != None:
    return A[x]
  y = x
  for i in range(log(x)):
    y = hash(y)
  y = y + fooDP(x/2) + fooDP(x/4)
  A[x] = y
  return y
```

Assume that n is a power of 2. What is the runtime of fooDP(n)? Assume **hash** runs in constant time.

[We are expecting: The runtime, along with a thorough proof.**]**

Solution

Because n is guaranteed to be a power of 2, we end up filling in A for $x = n, n/2, n/4, n/8, n/16, ...1$. Even if we just had the one fooDP($x/2$) recursive call in the final line we would still need to fill in all of these array entries. The additional call to fooDP($x/4$) doesn't require us to fill in any additional array entries because we were already would have gotten to fooDP($x/4$) in our call to fooDP($x/2$).
The runtime for filling in the array entry $A(x)$ is dominated by the $\log(x)$ calls to hash(y). The total number of times we call $y = (\text{hash})(y)$ is:
$\Sigma_{i=0}^{\log n} \log(n/2^i) = \Sigma_{i=0}^{\log n} \log n + \log(2^{-i}) = (\log n)^2 + \Sigma_{i=0}^{\log n} - i = (\log n)^2 - \frac{(\log n)^2}{2} = \frac{(\log n)^2}{2}$.
Thus our total runtime is $O((\log n)^2)$.

Note that if you wanted to write a recurrence relation for this DP problem, it would have to be $T(n) = T(n/2) + \log(n)$. This recurrence relation captures the fact that we have to do $O(\log n)$ work for each power of two less than or equal to n exactly one time. For the call to $T(n/4)$ we just return a memoized value from two recursive calls to $T(n/2)$, so adding a $+T(n/4)$ to our recurrence relation adds work to the relation that the program never actually has to do.
To think about this in another way, considering what a tree diagram would look like for this problem. It would just be a straight line down (one child per node) with subproblem size dividing by 2 at each node, and with $O(\log n)$ work at every node. This is all another way of saying that our runtime would have been the same if the code had said $y = y + \text{fooDP}(x/2)$, instead of $y = y + \text{fooDP}(x/2) + \text{fooDP}(x/4)$.

4 Graph Algorithm Analysis (23 pt.)

Let $G = (V, E)$ be a directed, weighted graph with the following properties:

- V is a set of n vertices.
- E is a set of m edges.
- For edge $(x, y) \in E$, let $w(x, y)$ be its *possibly negative* edge weight.
- Let $p = [x_1, \ldots, x_i]$ be a path of vertices in V. We define the path length $l(p) = \sum_{j=1}^{i-1} w(x_j, x_{j+1})$. In other words, the path length is the sum of all edge weights along the path.
- For $x \in V$ and $z \in V$, let distance $d(x, z)$ be the shortest path length from x to z.

4.1 (4 pt.)

For $x \in V$, let $h(x) = \min_{z \in V}(d(z, x))$. In other words, $h(x)$ is the shortest distance from *any* vertex to x (including from x to itself).

Describe an algorithm that computes $h(x)$ for all $x \in V$ in $O(mn)$ time if G has no negative cycles, and returns False otherwise.

[**We are expecting:** *Clear English Description OR Pseudocode, along with a proof of run-time and correctness*]

[**Hint:** *Try adding an extra node to the graph*]

Solution

We create a copy of G, which we call H. We add a new vertex q to H and create new edges (q, x) for each existing vertex x. We set the weights of these new edges to 0.
To compute $h(x)$ for each vertex x, we find the shortest path from q to x in H using Bellman-Ford in $O(mn)$ time. If Bellman-Ford detects a negative cycle in H, there must be a negative cycle in G also since there are no inbound edges to q, so we return False.

Proof of correctness Assume there exists some vertex x in G such that the value of $h(x)$ returned by the algorithm is not $\min_{z \in V}(d(z, x))$.
Case 1: $h(x) < \min_{z \in V}(d(z, x))$. Let p be a path from q to x in H with length $h(x)$. Let p' be p but with the first vertex q removed, so that is a path from some vertex r to x. The length of p' must also be $h(x)$ since the outbound edges from q have zero weight. Additionally, p' is also a valid path in H since it does not traverse q. Thus, $h(x) \geq d(r, x)$. This is a contradiction.
Case 2: $h(x) > \min_{z \in V}(d(z, x))$. Let p be a path from some vertex z to x in G with length $\min_{z \in V}(d(z, x))$. If we prepend p with q, then this would be a valid path from q to x shorter than the shortest path identified by Bellman-Ford. This is a contradiction.
Therefore, we conclude that the algorithm correctly returns $h(x) = \min_{z \in V}(d(z, x))$.

4.2 (3 pt.)

Let p be a path in G with starting vertex a and ending vertex b. Prove that $l(p) + h(a) \geq h(b)$, assuming G has no negative cycles.

[**We are expecting:** *A thorough proof of the statement*]

Solution

We can prove this by contradiction. Assume there exists a path p from a to b where $l(p)+h(a) < h(b)$. By definition of h, there must be a path from some vertex to a with length $h(a)$. Concatenating this path with p, we have a path from some vertex to b with length $l(p)+h(a)$. Additionally, by definition of h, the shortest path from any vertex to b is length $h(b)$. This is a contradiction since we have constructed a path to b with a shorter length than $h(b)$. Thus, we conclude $l(p) + h(a) \geq h(b)$.

4.3 (4 pt.)

Let G' be a directed, weighted graph with the following construction:

- $G' = (V, E)$. In other words, G' has the same set of vertices and edges as G.
- Let $w'(x, y)$ be the weight of edge (x, y) in G'. We define $w'(x, y) = w(x, y) + h(x) - h(y)$.

For a path $p = [x_1, \ldots, x_i]$ with nonzero length, let $l'(p)$ be the the length of p in G'. Prove that $l'(p) = l(p) + h(x_1) - h(x_i)$

[**We are expecting:** *A thorough proof of the statement*]

Solution

Algebraic proof From the definition of path length:

$$\begin{aligned} l'(p) &= \sum_{j=1}^{i-1} w'(x_j, x_{j+1}) \\ &= \sum_{j=1}^{i-1} \big(w(x_j, x_{j+1}) + h(x_j) - h(x_{j+1})\big) \qquad (1) \\ &= h(x_1) - h(x_i) + \sum_{1}^{i-1} w(x_i, x_{i+1}) \end{aligned}$$

For the final step, we note that all the $h(x_j)$ terms cancel out to zero except for $h(x_1)$ and $h(x_i)$. The final equation is equivalent to $l(p) + h(x_1) - h(x_i)$.

Proof by induction Let P be the set of all nonzero-length paths in G and G'. Let P_t specifically be the set of all paths traversing t edges. We note that $P = \bigcup_{t=1}^{\infty} P_t$.
We show by induction that for all $p \in P$, $l'(p) = l(p) + h(x_1) - h(x_i)$.
Base case: $t = 1$
Here, $p = [x_1, x_2]$. The distance through the single edge (x_1, x_2) in G is $w(x_1, x_2)$, and the distance in G' is $w'(x_1, x_2) = w(x_1, x_2) + h(x_1) - h(x_2)$. Thus $l'(p) = l(p) + h(x_1) - h(x_i)$ for $p \in P_1$.
Inductive step: Assume $l'(p) = l(p) + h(x_1) - h(x_i)$ for $p \in P_t$.
Let $q = [x_1, \ldots, x_{i+1}]$ be a path with length $p + 1$. The sub-path q excluding the final vertex must be a path of length p. Let this sub-path be s Thus, $l'(s) = l(s) + h(x_1) - h(x_i)$. Then:

$$\begin{aligned} l'(q) &= l'(s) + w'(x_i, x_{i+1}) \\ &= (l(s) + h(x_1) - h(x_i)) + (w(x_i, x_{i+1}) + h(x_i) - h(x_{i+1}) \\ &= (l(s) + w(x_i, x_{i+1})) + (h(x_1) - h(x_{i+1}) \\ &= l(q) + (h(x_1) - h(x_{i+1})) \end{aligned} \tag{2}$$

Thus, $l'(p) = l(p) + h(x_1) - h(x_i)$ for $p \in P_{t+1}$
We thus conclude for $p = [x_1, \ldots, x_i]$, $l'(p) = l(p) + h(x_1) - h(x_i)$.
Note: It is possible to use a zero-edge path as the base case in the proof by induction. However, this is not required since we can assume that paths are of nonzero lengths.

4.4 (3 pt.)

Conclude that all edge weights in G' are non-negative.

[**We are expecting:** *A thorough proof of the statement*]

Solution

We note that each edge $(x, y) \in E$ constitutes a path from x to y with length $w'(x, y)$ in G' and length $w(x, y)$ in G.
From the definition of G', $w'(x, y) = w(x, y) + h(x) - h(y)$.
Applying the property shown in subproblem 2, it must be that $w(x, y) + h(x) \geq h(y)$, so $w(x, y) + h(x) - h(y) \geq 0$. Thus, $w'(x, y) \geq 0$.

4.5 (4 pt.)

Let there be a path from a to b in G and G'. Show that if p is a shortest path from a to b in G, then it is also a shortest path from a to b in G'.

[**We are expecting:** *A thorough proof of the statement*]

Solution

For a given path $p \in P$, $l'(p) = l(p) + h(a) - h(b)$. Here, $h(a) - h(b)$ is a constant value for all $p \in P$. Thus, the path in P with the lowest l' must also have the lowest l.

4.6 (5 pt.)

Describe an algorithm that computes the all-pairs-shortest-paths problem for G in $O(mn + n^2 \log n)$ time if G has no negative cycles, and returns False otherwise. Feel free to use results from the previous parts.

[**We are expecting:** *Clear English description OR pseudocode, along with a proof of run-time and correctness*]

Solution

First, we compute $h(x)$ for $x \in V$ using the procedure in part 1. This takes $O(mn)$ time. If there is a negative cycle detected in G, we return False.
Next, we construct G' from G and $h(x)$. Since this can be done in a single pass over vertices and edges, this takes $O(m + n)$ time.
Lastly, we run Dijkstra's algorithm from every vertex in V on G'. Each call to Dijkstra's computes the shortest paths from a vertex to every other vertex. We can use Dijkstra's algorithm on G' since edge weights are all non-negative. This takes $O(m + n \log n)$ per call, which results in $O(mn + n^2 \log n)$ total runtime.
Together, the algorithm runs in $O(mn + n^2 \log n)$ time.

5 Table Making (10 pt.)

You are the manager of a small factory that produces handcrafted furniture. Your factory has been growing in popularity, and you've decided to hire a team of n skilled workers to help you meet the increasing demand for your products.

One day, you receive a big order for a set of m custom-designed dining tables, each of which requires a specific set of skills to complete. You need to decide which workers to assign to each task while taking into account their availability and skill sets.

Suppose each worker i has his/her own flexible schedule, and can only work on certain days $D_i \subseteq [1, N]$. Also, since different tables require different skills to make, not all workers can make all the tables. For every table j, we use $W_j \subseteq [1, n]$ to denote the set of workers that can make it. It always takes exactly one day for a worker to make a customized table.

Assume that $m = \Theta(n), N = \Theta(n), \sum_{j=1}^{m} |W_j| = \Theta(n), \sum_{i=1}^{N} |D_i| = \Theta(n^2)$ for the purposes of runtime calculation.

5.1 (5 pt.)

Design a max-flow algorithm that can help you decide what is the max number of custom tables that can be made by these workers. The runtime of your algorithm should be $O(n^2)$.

[**We are expecting:** *Clear English description OR pseudocode, along with a proof of runtime and correctness*]

Solution

To solve this problem, we can represent it as a network flow problem where each worker and each table correspond to one node in the network. We also create a source node and a sink node. We connect the source node to each worker node with an edge whose capacity is the number of days that the worker is available to work, connect each worker node to each table node that they can work on with an edge whose capacity is 1, and connect each table node to the sink node with an edge whose capacity is 1.
Use the max-flow algorithm to find the maximum flow from the source node to the sink node, taking into account the capacity constraints on the edges connecting the table nodes to the worker nodes. The value of the maximum flow represents the maximum number of tables that can be completed by the workers.
Since $\sum_{j=1}^{m} |W_j| = \Theta(n)$ and the maximal flow is $m = \Theta(n)$, according to the Ford-Fulkerson algorithm the runtime is $O(n^2)$ (recall that Ford-Fulkerson has a runtime of $O(mf)$).

5.2 (5 pt.)

It turns out that your factory has limited working space, and it can sometimes become too crowded if too many workers show up. As a result, you've recently decided that at most k workers can come to work on any given day. Modify your algorithm to compute the max number of tables that can be made after you enforce this policy. The runtime of your algorithm should be $O(n^3)$.

[**We are expecting:** *Clear English description OR pseudocode, along with a proof of run-time and correctness*]

Solution

To satisfy the additional constraint, we modify the graph and make it a three layer graph as follows. In addition to the source node, sink node, worker nodes and table nodes, we create an additional node for each day.

Instead of connecting the sink node directly to worker nodes, we first connect the sink node to each day node with capacity k, then we connect each day node to a worker node where the worker is available on this specific day, and set capacity to 1.

Similar to previous case, we connect each worker node to each table node that they can work on with an edge whose capacity is 1, and connect each table node to the sink node with an edge whose capacity is 1.

The total number of edges in this new graph is $O(n^2)$ and the maximal possible flow is $m = O(n)$, so according to the Ford-Fulkerson algorithm the runtime is $O(n^3)$.

This page is intentionally blank for extra space for any question.

Please indicate in the relevant problem if you have work here that you want to be graded, and label your work clearly.

This page is intentionally blank for extra space for any question.

Please indicate in the relevant problem if you have work here that you want to be graded, and label your work clearly.

This page is intentionally blank for extra space for any question.

Please indicate in the relevant problem if you have work here that you want to be graded, and label your work clearly.

This page is intentionally blank for extra space for any question.

Please indicate in the relevant problem if you have work here that you want to be graded, and label your work clearly.