CS 4102 Final

Name _____

You MUST write your e-mail ID on **EACH** page and bubble in your userid at the bottom of this first page. And put your name on the top of this page, too.

If you are still writing when "pens down" is called, your exam will be ripped up and not graded – even if you are still writing to fill in the bubble form. So please do that first. Sorry to have to be strict on this!

Other than bubbling in your userid at the bottom of this page, please do not write in the footer section of this page.

There are 9 pages to this exam. Once the exam starts, please make sure you have all the pages. Questions are worth different amounts of points.

If you do not bubble in this first page properly, you will not receive credit for the exam!

This exam is CLOSED text book, closed-notes, closed-calculator, closed-cell phone, closed-computer, closed-neighbor, etc. Questions are worth different amounts, so be sure to look over all the questions and plan your time accordingly. Please sign the honor pledge below.

_____

_____

_____

_____

_____

It is exceptional that one should be able to acquire the understanding of a process without having previously acquired a deep familiarity with running it, with using it, before one has assimilated it in an instinctive and empirical way.
–John von Neumann

(the bubble footer is automatically inserted into this space)

Page 2: Dynamic Programming 1

1. [5 points] What is ”optimal substructure”?

   Solutions to large problem is composed of solution to smaller subproblems

2. [5 points] Describe how we used optimal substructure for the Seam Carving algorithm discussed in class.

   Least energy path ending at pixel (i,j) is calculated using the energy of the path to pixels (i-1, j-1), (i-1, j), (i-1, j+1)

3. [5 points] How does optimal substructure differ with regards to a greedy algorithm versus a dynamic programming algorithm?

   Greedy algorithms depend only on a single subproblem, DP may depend on several.

Page 3: Dynamic Programming 2

4. [10 points] In class we discussed that a greedy algorithm is optimal for the change-making problem in some instances (e.g. for U.S. currency), but does not work in general (e.g. if we included "The Kim").

   Devise a dynamic programming algorithm to solve change-making in the general case. I.e., if you receive an endless supply of coins of n different positive integer denominations $C = \{c_1, \ldots c_n\}$ and a positive integer v, write an algorithm which gives the fewest total number of coins from C needed to sum exactly to v. (To be clear, your algorithm only needs to return the *number* of coins)

   To get you started, I recommend you compare a solution in which the largest coin is of value c_1 with a solution in which the largest coin is of value c_2, etc..

   Idea: $Best(v, c_i)$ = best solution for target value v s.t. c_i is the highest-denomination coin.
   assume w.l.o.g. that $c_i < c_{i+1}$ (if this is not the case then sort them)
   optimal substructure: $Best(v, c_i) = \min_{i>j}\{Best(v, c_j + 1)\}$
   This gives a $n \times |v|$ array, which can be computed using a double for loop. The return value should be $\min_i\{Best(v, c_i)\}$

5. [5 points] What is the run time of your algorithm? Is it polynomial time or is it exponential time (relative to the size of the input)?

   $O(n \times |v|)$ which is exponential as it depends on the value of v rather than its size.

Page 4: Divide and Conquer

6. [points] A list L of length n has a majority element m if more than half of the elements in the list are equal to m. We can find the majority element of a list using the following divide and conquer algorithm.

   Arbitrarily pair up all elements in the list (making $\lfloor \frac{n}{2} \rfloor$ pairs. For each pair, if its two values are equal, add the first value to a new list L'. If n is odd, add the odd-item-out to L'

Page 5: Greedy

7. [10 points] There are n jars on a table filled with very valuable magic dust. Each jar i weighs w_i grams, and its total contents are worth c_i dollars. You have a bag that can hold up to W grams of dust. Give a greedy algorithm which gives how much of the dust from each jar you should put into your bag in order to maximize the total value of the bag's contents, without exceeding its capacity W. (Your solution may include any arbitrary fraction of a jar's dust)

   select all of the dust in the jars in order of highest "value density" first (i.e. select in order of $\frac{c_i}{w_i}$)

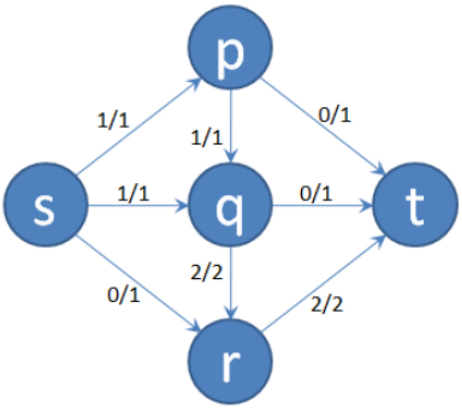
8. [5 points] Explain why your algorithm is greedy. As part of your answer, state clearly what the *greedy rule* (or *greedy selection function*) is for your algorithm. (If this is explained in the previous part, you can reference back to that answer.)

   This is greedy because only one possible subproblem is considered.

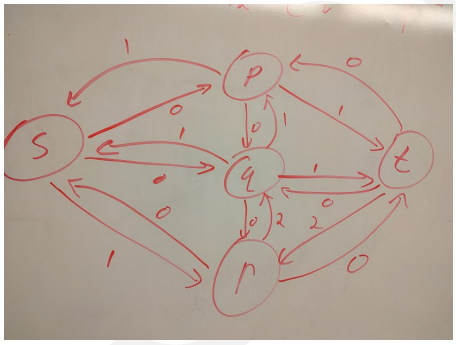
Page 6: Graphs 1

9. [6 points] Describe two situations in which one might choose to use the Bellman-Ford algorithm instead of Dijkstra's.

   1) there are negative edge weights, 2) the graph's edge weights change over time (it's dynamic)



10. [6 points] Given the graph G above, draw its residual graph G'.



11. [3 points] Find an augmenting path in the graph G'. List the nodes in the path you found in order (e.g., $s \to a \to b \to c \to d \to t$)

    s, r, q, t

Page 7: Graphs 2

12. [10 points] Prove the following statement is true. You may write your argument in natural language, but your argument should still be logically formal:

    If a flow network has no augmenting path with respect to the current flow f, then there exists a cut S whose capacity is exactly f.

    If there is no augmenting path, then f is the max flow, therefore is also the min cut, so a cut exists with capacity f.

13. [5 points] What does the Cut Theorem (a.k.a. the Cut Property) state regarding Minimum Spanning Trees?

    There is always a MST which includes the least-weight edge which crosses any cut.

Page 8: NP-Completeness

14. [12 points] Given a problem A, state how one would show that A belongs to each of these classes:

    P:

    Solve A in polynomial time

    NP:

    Verify A in polynomial time

    NP-Hard:

    reduce a known NP-Hard problem to A in polynomial time

    NP-Complete:

    show it is both in NP and NP-Hard

15. [10 points] Prove the following: If someone proves that a given problem A in NP has an exponential lower bound, then no problem in NP-complete can be solved in polynomial time.

    Assume toward a contradiction that there was a polynomial time algorithm for some NP-complete problem B . By definition, every NP problem reduces to every NP-Hard problem (which includes NP-complete) in polynomial time. This means that A could be solved in polynomial time by reducing A to B in polynomial time, then solving B in polynomial time. Thus B cannot be solved in polynomial time.

Page 9: Misc

16. [3 points] Differential Privacy: As you turn in your exam, flip a coin at the front of the room. If the coin lands heads, write "yes" here, if the coin lands tails, please truthfully answer:

    As far as you're aware, did you, at any point this semester, violate the collaboration policy in CS4102?

    Yes/No

17. [3 points] Extra Credit: My grandma is 88 years old, and has never used a computer before in her life. Explain what you've learned in this course in a way that my grandma could appreciate.