Kolkata Short Course **Course Assignment** **Due Wed. 31 Aug.**

This assignment has eight problems totaling 200 points. There is a mixture of types of problems. I have chosen the mix to correspond to your grading system, which I understand allows a somewhat lower percentage than my standard 90% to be counted as top marks. This is an individual-work assignment that will constitute the major portion of your marking in the course (there is also a 5-pt. bonus for those I recorded as demonstrating their work on the day-3 challenge question and some consideration of attendance).

I have created `http://www.cse.buffalo.edu/~regan/KolkataAlgorithms/` as a separate web folder for the course. I have uploaded relevant sources there, including updating some of my slides, and you are welcome to use those *freely*. Any other reasonable source (such as a previous textbook), however, must be identified and credited in your assignment. And of course you are barred from looking for answers on the Internet. I have tried to make this assignment "non-cheatable" in certain respects, but especially with introductory material it cannot be done perfectly, so you are on your honour in accordance with the policies of your University.

If you have questions, you are welcome to e-mail them to me. I will have good e-mail contact in August 18–22 and Aug. 27–29, less other times. Most of the problems are from the days 2–4 topics which actually extended past lunch on day 5; there is only a small representation of the last day's material which I had to cover in "allegro" fashion. All the best on this assignment.

(1) (15 pts.)

Use the method of graph cycles to show that the following Boolean formula in 2CNF is not satisfiable:

$$(\bar{u} \vee v) \wedge (u \vee y) \wedge (\bar{x} \vee v) \wedge (y \vee w) \wedge (w \vee x) \wedge (\bar{v} \vee \bar{y})$$
$$\wedge (\bar{x} \vee y) \wedge (u \vee \bar{x}) \wedge (\bar{w} \vee \bar{v}) \wedge (u \vee v) \wedge (x \vee u).$$

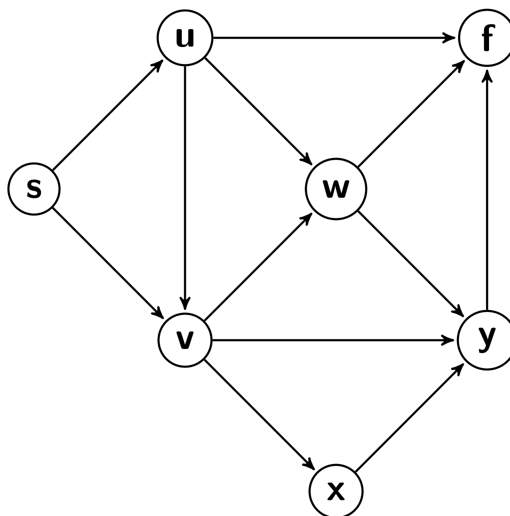
(2) (10 + 15 = 25 pts.)

Here is a more-formal statement of the rules for a mini-army of k riders to be able to "conquer" a directed graph G in which every node u has a "hit-strength" $h_u \geq 1$:

1. All k riders begin on the start node s.
2. If j in-neighbours of a node u are occupied and $j \geq h_u$ then node u is "conquered" and any of the k riders (not necessarily one of the j who attacked) can occupy node u.
3. If a rider vacates a node $v \neq s$ that was previously conquered, then node v goes into "passive resistance" mode. This means that a rider cannot re-occupy the node unless it is re-conquered all over again. (The start node s can always be re-occupied.)
4. Multiple riders can occupy any node, and a rider can move to a (newly-)conquered node directly from anywhere (figuratively, "riding by night through forests"), not necessarily through nodes controlled by the riders.

The relevance of these rules is that the k riders are k registers of a random-access machine. Moving from node u to a newly-occupied node v over-writes u by v in the register—this is why the army "forgets" that node u was previously conquered. A new value can be loaded into any register in one step—that is the reasons for the last rule's liberality. Re-occupying s is like resetting a register.

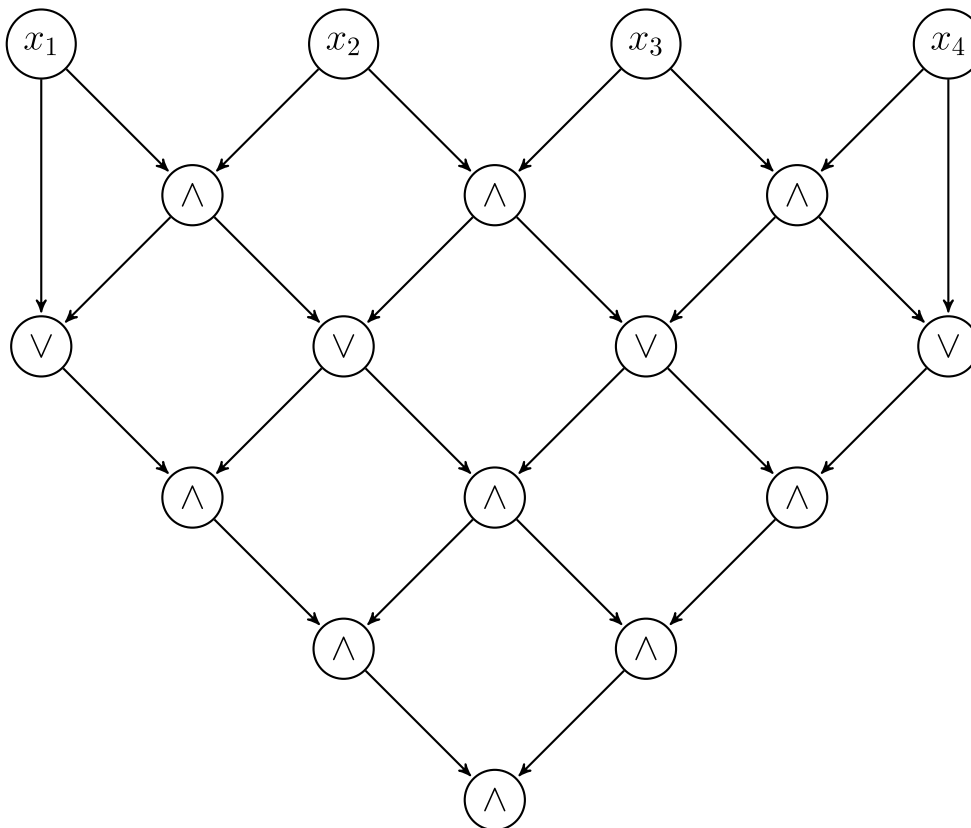
(a) Show that an army of $k = 3$ riders beginning at s can conquer node f in the following graph, where the hit strength of each node equals its in-degree (so y and f have 3, others 1 or 2). The rule change allowing the free re-occupation of s (compared to lecture where this was forgotten) makes a difference.



(b) If the edge from u to v is reversed—so that it now goes from v to u—then can an army of three riders still conquer node f? If you say no, prove it as best you can; if you find a way, say which nodes (other than s) are conquered more than once.

(3) (10 + 15 + 15 = 40 pts.)

More conquering, this time of Boolean circuits. Recall that an AND gate has hit-strength 2 but OR has only 1. Consider the following circuit:



You may also imagine a start node s at the top with out-edges to x_1, x_2, x_3, x_4 as I drew in lecture. With or without it, the idea is that any of the four input nodes may be freely re-occupied/re-conquered by any rider. In these examples we are skipping the negated inputs $\bar{x}_1, \bar{x}_2, \bar{x}_3, \bar{x}_4$ which you would in general get from applying DeMorgan's Laws. Put another way, this third problem is only dealing with *monotone* circuits.

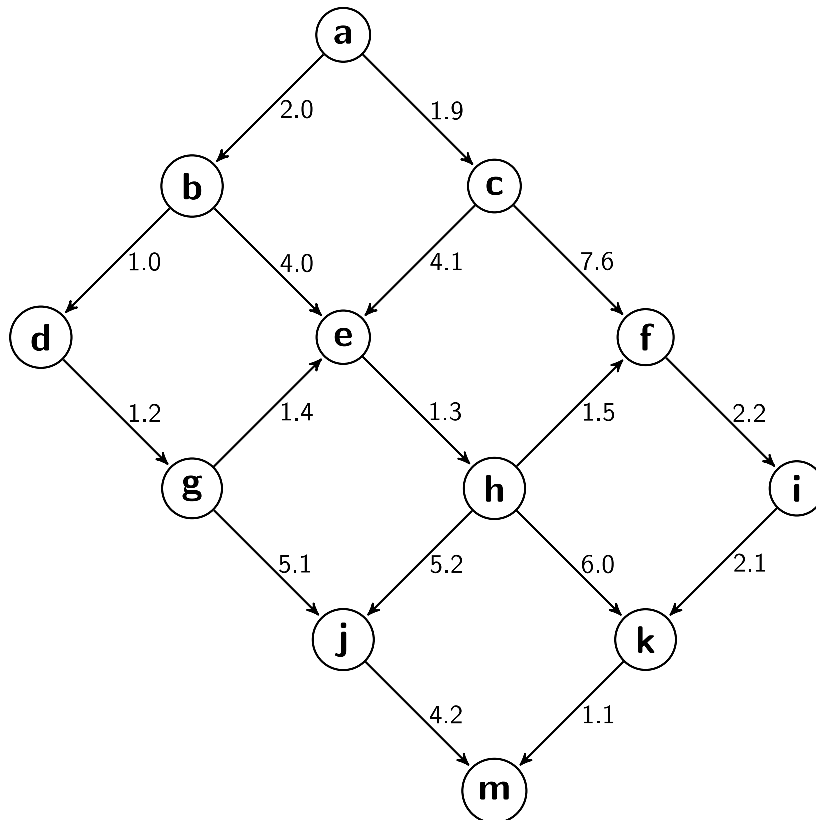
(a) Argue that three "riders" *cannot* conquer the output node at the bottom.

(b) Show, however, that if any one of the bottom six gates is changed from AND to OR—that is, from $\wedge$ to $\vee$—then the circuit *can* be conquered by an army of 3 riders.

(c) For a more-general question, consider the computational problem of deciding, given the graph G of a circuit (or *any* directed graph G), whether G can be conquered with 3 riders. Prove that this problem belongs to the complexity class NL—by showing how it can be "solved" by a *nondeterministic* algorithm that maintains finitely many pointers into G as a read-only data structure.

(4) (15 pts.)

Show how Dijkstra's algorithm works to find a shortest path from node a to node m in the following graph. Note that the arrows from g to e and from and h to f go up not down. Your trace of the run of the algorithm should include the use of a heap data structure and any changes in the optimal path computed to any node.



(5) (35 pts. total)

Here is a formal description of the "liberal" algorithm for computing a minimum-weight spanning tree:

1. Begin with a forest of n subtrees, each being an isolated vertex.
2. At any step, choose any one of the subtrees T in your forest "at will.'
3. Consider every node in T, and choose a minimum-weight edge that touches a node in T bit does not make a cycle.
4. Continue until a spanning tree S is built.

If you always choose a T that a/the minimum-weight edge overall touches (that is, minimum among edges that do not cause cycles), then you get *Kruskal's algorithm.* If you choose one node t and then always choose T to be the connected subtree that includes t, then you get *Prim's algorithm.* The question is whether the above liberal allowance to blend the two ideas can ever make a mistake.

The consensus of the class in the Monday 8/8 lecture is that this liberal version is still correct. Your task now is to *prove* it. You may suppose that all edge-weights are distinct, so that there is a unique minimum spanning tree T_*. Your proof builds a spanning tree S. Consider the first step at which a run of the "liberal" algorithm might possibly deviate from T_*; since edge weights are distinct this also means that no possible continuation of inspired choices could possibly correct the mistake. Your proof should involve this idea.

Then finally try to extend your proof to cover cases of equal edge-weights. In case two or more edges touching T are tied for the minimum, again you may choose one "at will." For your proof, focus on the clause "no possible continuation. . . " above.

For a confidence-building warmup, and also fun, treat the graph in problem (4) as *undirected* and try this way of determining the "at-will" choices: For each letter in your full name in the range a–m (maybe counting 'p' as 'f' and 'l' as follow-Kruskal), choose the tree T that includes the vertex with that letter. For example, "Mamata Banerjee" chooses first the edge weighted 1.1 between m and k (not the Kruskal edge (b, d) weighted 1.0), and next the 1.9 edge (a, c). For her second m she chooses *not* the edge (j, m) but rather the edge (i, k) worth 2.1 since it is part of the same component as m. She does choose (a, b) next but only because node c has worse options. The letter 't' she just skips, then finally the Kruskal edge (b, d) is taken with the last letter in her first name. The 'B' then selects edge (d, g) since it is in the same component as node b and has the least available weight. If and when you reach the end of your last name, start again from the beginning. Please do this exercise as a help in case of error in the proof, so to clarify that you do understand the concept. Since all the edge weights are distinct, the answer you get should be the same as if you followed Kruskal and/or Prim strictly.

(6) (6 + 9 = 15 pts.)

First some short problems on edit distance. Without filling in the dynamic-programming matrix, give a short clear statement on why the edit distance between "HOOGLY" and "GANGES" is 5. Then *with* filling in the matrix, demonstrate the edit distance between "HUGLEE" and "GANGES."

(7) (40 pts. total)

Now for a more theoretical question. Define a *2-ary rewriting system* to be a finite set R of rules of the form A→BC where A,B,C are any letters, possibly equal. The question is now about a single word w rather than a pair of words and a starting letter c. The question is, starting from c, can we apply rules to generate the word w? Note that if w has length n, then we will apply rules exactly $n - 1$ times, because each application increases the length of the word we are building by 1. Here is an example:

$$R = \{\mathrm{S} \to \mathrm{SS}, \mathrm{S} \to \mathrm{GA}, \mathrm{A} \to \mathrm{AN}\}, \quad c = \mathrm{S}, \quad w = \mathrm{GANGA}.$$

The answer is yes, we can generate GANGA from S by S→SS→GAS→GANS→GANGA. Note that the rule S→GA was applied twice in this sequence.

Design a dynamic-programming algorithm to solve this problem in the abstract. Here is a generous hint: Given the particular word w of length n and a character c'—where c' is not necessarily the same as c—you will want to make an $n \times n$ table $D_{c'}$. Actually you only need the upper-left triangular half of the table for entries $[i, j]$ with $i \leq j$, since the goal is to tell whether $w[i \dots j]$ (that is, the part of w from the ith character up to and including the jth character) can be generated starting with c'. As-described you will have one table for each c', but as a convenience you can make just one table D and fill in each box with any and all characters c' that work—or *emptyset* for the empty set if none do. To get you started, the cells D along the bottom edge of the triangle where $j = i$ have nothing but the respective characters of w, since the only way to get a single-char string $w[i, i]$ is to start with the right character and apply zero rules.

(8) (15 pts.)

Consider the problem of whether, given a directed graph G and 3 "riders" as before, there is a way the riders can place themselves so that they either occupy or threaten every node, Call this "surveying"—we are ignoring any need to "conquer" the occupied nodes. For example, in the graph of problem (2), the riders can occupy s, u, v and then the other nodes w, x, y, f are all menaced at least once. If, however, you make a graph G' by adding a new node z with an edge to y, then you need observers on both s and z, and no other node can handle both f and x, so the answer is no.

Show in the case of this graph G how to create a logical formula ϕ such that G can be surveyed from 3 nodes if and only if ϕ is satisfiable, and show this also with G' which will produce a corresponding (but unsatisfiable) ϕ.