CS 170 (Clancy) **Solutions and grading standards for exam 2**
Fall 2000

Exam information

142 students took the exam. Scores ranged from 5 to 28, with a median of 17 and an average of 16.7. There were 17 scores between 23 and 30 (inclusive), 72 between 16 and 23, 50 between 8 and 15, and 3 between 1 and 7. (Three-quarters of the points on exams plus good grades on homework and projects would earn you an A–; similarly, a test grade of 16 may be projected to a B–.)

There was only a single version of the exam.

If you think we made a mistake in grading your exam, describe the mistake in writing and hand the description with the exam to your discussion t.a. or to Mike Clancy. We will regrade the entire exam.

Solutions and grading standards

Problem 0 (1 point)

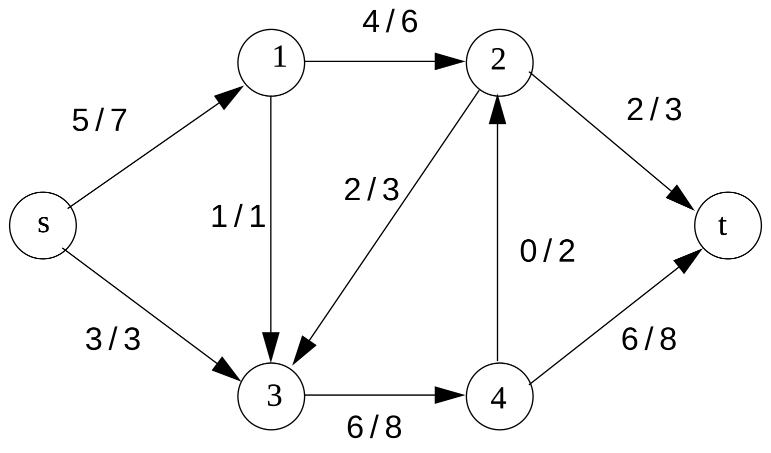
You could have lost the point for this problem for several reasons:

you earned some credit on a problem and did not put your name on the page;
you did not indicate your discussion section and t.a.;
you did not indicate your login on the EECS instructional systems;
you did not identify where you were sitting.

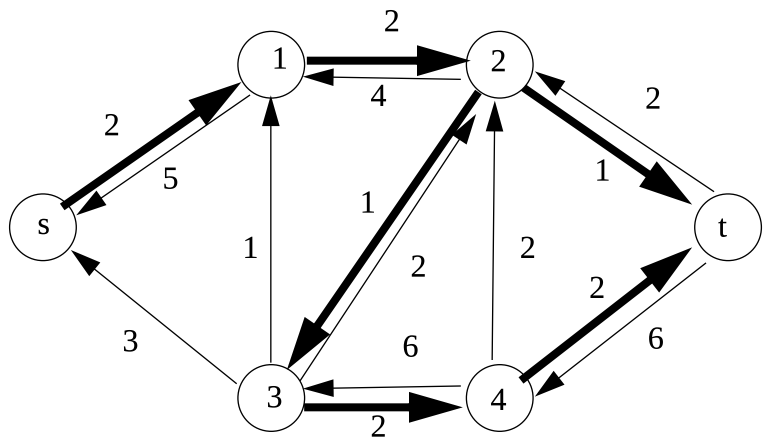
The reason for this apparent harshness is that we need ways to recover misplaced or unstapled. We also need to know where you will expect to get your exam returned, and how to submit your grade to the grading system.

Problem 1 (6 points)

In this problem, you considered the following network.



For part a, you were to display the corresponding residual graph, shown below. Also shown in bold are paths from s to t in the graph; you didn't have to specify these, but they're useful for the remaining parts of this problem.



For part b, you were to find an edge whose capacity could be reduced to make the given flow optimal. This edge would be found by examining the residual graph, and choosing an edge whose removal would block all paths from s to t. (Reducing the edge's capacity to the level of its flow would saturate the edge; it thus would not appear in the residual graph.) There are two edges that qualify: the edge from s to 1 and the edge from 1 to 2. Either capacity can be reduced by 2.

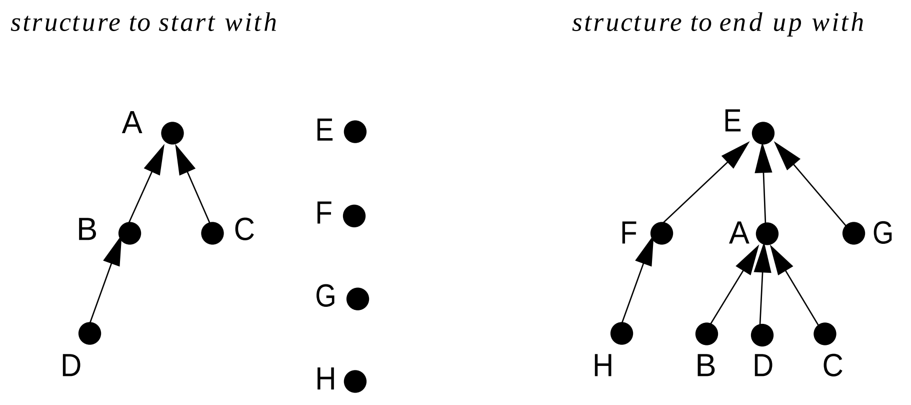
Part c was to produce a corresponding cut. Again, this could be determined from the (modified) residual graph; all vertices reachable from s constitute a minimum cut. Your answer here would depend on your answer to part b. If you reduced the capacity of edge (s, 1), the cut would be {s}; if you reduced edge (1, 2), the cut would be {s, 1}. You were allowed merely to draw the cut on the diagram.

Points for this problem were allocated 3 for part a, 2 for part b, and 1 for part c. You received 2 points for an almost correct solution to part a; typically, such a solution included a small error—e.g., a single missing edge—but showed clear knowledge of the basic components of the residual graph, that is, the unsaturated edges and the back edges. A solution that included unsaturated edges but not back edges earned 1 out of 3 on part a.

An incorrect solution to part b probably earned 0 points. Reducing an edge capacity below its current flow and reducing the capacity of a nonessential edge were common errors. The point for part c was based on your solution to part b; thus it was possible to earn no credit on part b but still find a cut of capaicty 8 to earn the point for part c.

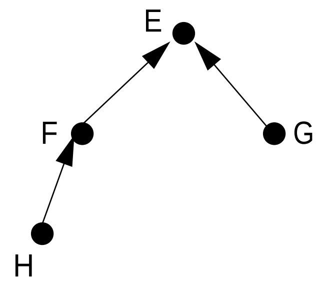
Problem 2 (5 points)

This problem was to give a sequence of calls to union (using path compression and union-by-rank) that would produce the structure on the right in the diagram below from the structure on the left..



Inspection of the goal structure indicates that elements E, F, G, and H form a structure identical to the structure initially containing A, B, C, and D, and path compression has been done on element D. We thus start by forming the set {E, F, G, H}:

```
union (H, F);
union (G, E);
union (H or F, G or E);
```



The “or” indicates a choice of arguments; thus the last call could have been one of

```
union (H, G);
union (H, E);
union (F, G);
union (F, E);
```

Each of the calls to union involve arguments with equal rank (0 in the first two calls, 1 in the next); the tie is broken by putting the characteristic element of the *second* argument at the root of the structure. The following call combines the two trees and also incorporates the desired path compression for D:

```
union (D, E or F or G);
```

Essentially there were five things that had to be done in this problem: four links to be defined, and one link to be changed via path compression. In general, 1 point was awarded for each of these. Illegal operations, e.g. explicit calls to find, received no credit. We attempted to determine the steps you got correct and award credit appropriately, so that an error would not lose you all the points for remaining operations.

You were allowed to separate the path compression from the combination of {A, B, C, D} with {E, F, G, H} by including an extra call to union, for example:

```
union (D, A);
```

Other errors included operations performed out of sequence (which produced incorrect answers because of union-by-rank) and mishandling arguments of equal rank. Not explaining the operations with words or diagrams lost you a point.

Problem 3 (6 points)

For this problem, you considered a linear programming problem whose solution, if variables were restricted to 0-1 values, would represent a vertex cover. You were first to use the possibly fractional solution values to design an approximation algorithm for finding a vertex cover, then to determine a ratio bound of the algorithm.

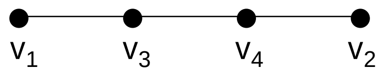
Here's an algorithm. First, make a pass through the x_k, changing all those with value $\frac{1}{2}$ to have value 1. Then choose for the vertex cover all vertices v_k for which $x_k = 1$.

- Can there be an edge (v_j, v_k) that's not covered? No, since in order to satisfy the constraint $x_j + x_k \geq 1$, either one of x_j and x_k has value 1 or both have value $\frac{1}{2}$ in the LP solution; in either case, at least one of v_j and v_k is chosen for the cover.
- How good an approximate cover results? The value of the LP objective function $x_1 + \ldots + x_n$ is a lower bound on the size of the optimal cover. By changing x values from $\frac{1}{2}$ to 1, we at most double the value of the objective function. With all x_k taking on 0-1 values, the objective function represents the size of the corresponding cover. Thus the cover produced by the algorithm is at most twice the size of the optimal cover. The ratio bound is $\max(\frac{1}{2}, \frac{2}{1}) = 2$.

Part a was worth 4 points, 2 for the algorithm and 2 for the argument that it produces a vertex cover. The most common error was to offer no argument at all that the algorithm produces a vertex cover. Next most common was a flawed attempt at optimization: instead of just taking all v_k for which $x_k > 0$, some students tried to be more selective, changing some of the x_k values to 0 and some to 1. An example of this approach is the following algorithm:

> While there are x_k's with value $\frac{1}{2}$, pick one, change it to 1, and change all the neighboring x_j with value $\frac{1}{2}$ to 0.

This doesn't work for the path of length 4 shown below.



The optimal solution for the LP is $x_1 = x_2 = x_3 = x_4 = 1/2$; choose x_1, set it to 1 and set x_3 to 0, then choose x_2, set it to 1 and set x_4 to 0. The vertices thus selected are v_1 and v_2, which are not a vertex cover. With an algorithm like this, you generally earned 1 point in part a and 0 points in part b.

Part b was worth 2 points. A correct bound alone, without some justification, earned 0 points. Many solutions provided an incorrect justification similar to the following:

> Any vertex cover includes at least one endpoint from each edge. My algorithm might include both endpoints from some edges. Therefore my algorithm produces a vertex cover with at most twice as many vertices as the optimal vertex cover.

This overlooks the fact that a vertex may cover more than one edge. Consider a "star" graph with a center vertex connected to four other vertices. The center vertex alone covers all four edges, but including the other endpoint for each of the four edges results in a cover of five vertices, and of course $5/1 > 2$.

Problem 4 (6 points)

This problem involved proving the Maximum Common Induced Subgraph problem NP-complete. (We'll call it MCIS in the discussion below.)

You first had to provide the corresponding decision problem, which was

> Given k, G_1, and G_2, is there a graph H with at least k vertices such that H is an induced subgraph of both G_1 and G_2?

You were allowed to say "exactly k vertices" instead of "at least k vertices", since if there's a common induced subgraph of size $\geq k$, there is also one of size exactly k.

Next came the actual proof of NP-completeness, which has two parts: showing the problem is in NP, and reducing some known NP-complete problem to it.

Showing membership in NP involves producing, along with a "yes" answer to the decision problem, a *certificate* with which the answer can be checked in polynomial time. The certificate must include information that enables efficient checking for the following:

- The graph H has k vertices.
- H is an induced subgraph of G_1.
- H is an induced subgraph of G_2.

An important component of the certificate is thus the *mapping* from vertex numbers in H to vertex numbers in G_1 and G_2. Without this mapping, one is reduced in the worst case to a brute-force search for corresponding vertices as you did in project 1.

The most challenging part of the actual reduction was choosing the problem to reduce from. Many of you chose the Subgraph Isomorphism problem, attempting the following reduction.

> Consider an arbitrary instance of the Subgraph Isomorphism problem; that is, determine if a graph H is a subgraph of a graph G.
>
> Supply G and H unmodified to the maximum common induced subgraph decider along with $|V(H)|$, asking if G and H have a common induced subgraph of size $\geq |V(H)|$. Its answer is the answer to return for subgraph isomorphism.

Unfortunately, a “no” answer from the maximum common induced subgraph decider merely says that H is not an *induced subgraph* of G, not that H isn’t a *subgraph* of G. (We had hoped that the example at the start of the problem would make the distinction clear.) Greg Jones put together a correct reduction from Subgraph Isomorphism; it will soon be included for your inspection.

A better choice is to reduce from Clique, as you did for the short exercise on homework assignment 6:

> Consider an arbitrary instance of the Clique problem, namely, given a graph G and an integer k, determine if G contains a clique of size k.
>
> Supply G, a clique of size k, and k itself to the maximum common induced subgraph decider, asking if G and the clique have a common induced subgraph of size $\geq k$. If the answer is “yes”, the subgraph is the clique itself requested for the Clique problem. If the answer is “no”, G does not contain a clique of size k.

Part a, the decision problem, was worth 1 point. Errors that lost that point included asking for a common induced subgraph of size *at most* k (there is always a common induced subgraph of size 0 or 1) and asking whether the maximum common induced subgraph of G_1 and G_2 has size k. The latter question will cause serious problems when one tries to show the decision problem is in NP: even if you demonstrate a common induced subgraph of size k, how do you know you can’t do any better? (Saying “at least k” would have worked fine.)

Part b was worth 5 points, 2 for showing the problem is in NP, 3 for showing it’s NP-hard.

Several people talked about it being easy to check the certificate without ever mentioning what the certificate was. Other people said the certificate was the two subgraphs, and then said it’s easy to check that these are isomorphic. But this is the Graph Isomorphism Problem, which isn’t known to be in P. Thus the correspondence between the vertices in the subgraphs should be included as part of the certificate. People who were fairly explicit about the certificate and checking that the subgraphs were induced, but who didn’t address the problem of their isomorphism, got 1 point out of 2.

As noted above, many of you tried to reduce Subgraph Isomorphism to MCIS. You earned 1 point for this if you did it in the correct direction with the correct arguments but ignored the distinction between subgraph and induced subgraph. Other people tried to reduce Graph Isomorphism to MCIS. Even with a valid reduction, however, this wouldn't show that MCIS is NP-hard since Graph Isomorphism isn't known to be NP-complete. No points were awarded for this.

You earned 1 point for trying to reduce Clique to MCIS, 1 more point for showing that "yes" from MCIS maps to "yes" from Clique and 1 more point for showing that "no" maps to "no".

Problem 5 (6 points)

For this problem, you were to prove that the given greedy algorithm for covering points on the real number line with unit intervals (which have length 1) produces the smallest possible set of covering intervals. (This problem, incidentally, is CLR exercise 17.2-5.)

One approach to proving greedy algorithms correct is to verify the greedy choice and optimal substructure properties. Here, the greedy choice property says that the interval chosen by the algorithm—the interval $[x_1, x_1+1]$—is part of an optimal cover. The optimal substructure property says that optimal solutions are built from optimal subsolutions. Here, it says that if $[x_1, x_1+1] \cup C$ is an optimal cover for $\{x_1, x_2, \ldots, x_n\}$, then C is an optimal cover for points in $\{x_1, x_2, \ldots, x_n\} - [x_1, x_1+1]$.

A proof of the greedy choice property can construct a suitable cover as follows. First, we assume that some optimal cover C does not contain $[x_1, x_1+1]$. One of its intervals must contain x_1, so assume that interval is $[y, y+1]$ where $y < x_1 \leq y+1$. Since $y < x_1$, $y+1 < x_1+1$, so the interval $[x_1, x_1+1]$ contains at least as many of the x_k's as the interval $[x_1, y+1]$. Moreover, since x_1 is the leftmost point of those to be covered, there are no points in the interval $[y, x_1)$. Thus $[x_1, x_1+1]$ contains at least as many of the x_k's as $[y, y+1]$, and substituting $[x_1, x_1+1]$ for $[y, y+1]$ in C results in a cover of all the x_k's with exactly as many intervals.

A variation on this proof would assume that no optimal cover contains $[x_1, x_1+1]$ and then go on to derive a contradiction using the above approach.

We prove optimal substructure—if $[x_1, x_1+1] \cup C$ is an optimal cover for $\{x_1, x_2, \ldots, x_n\}$, then C is an optimal cover for points in $\{x_1, x_2, \ldots, x_n\} - [x_1, x_1+1]$—by induction on the number of greedy choices. First, the base case: If one interval covers all the points, C and the resulting set of points are both empty, so we have a trivial "optimal substructure". Now suppose that two or more greedy choices are made, that $[x_1, x_1+1] \cup C$ is an optimal cover for $\{x_1, x_2, \ldots, x_n\}$, and that C is not an optimal cover for points in $\{x_1, x_2, \ldots, x_n\} - [x_1, x_1+1]$. Thus there is another cover C' with one fewer interval. But C', together with the interval $[x_1, x_1+1]$, covers all the points in $\{x_1, x_2, \ldots, x_n\}$ and has one fewer interval than C. This contradicts the assumption that C is an optimal cover of $\{x_1, x_2, \ldots, x_n\}$.

A more concise proof that the algorithm works, provided by Francis Hsu, is the following. Consider the left endpoints of the intervals chosen by the greedy algorithm.

All these x_k's must be covered; moreover, no unit interval can cover more than one of them since by construction the difference between one of them and the next is more than 1. Thus any legal cover must contain at least as many intervals as the cover produced by the greedy algorithm.

For solutions that split the proof into greedy choice and optimal substructure sections, we awarded up to 3 points per section. Of the 3 greedy choice points, you earned 1 by noting that $[x_1, x_1+1]$ included the most points among all intervals containing x_1. You earned the other 2 by showing the existence of an optimal cover containing $[x_1, x_1+1]$; your proof had to include some mention of substituting $[x_1, x_1+1]$ for some other interval in an optimal cover to get either of these points. Of the 3 optimal substructure points, you earned 1 for noting some kind of equivalent substructure. The other 2 points were for removing $[x_1, x_1+1]$, invoking the induction hypothesis, and then putting $[x_1, x_1+1]$ back into the cover thus derived.

Many solutions attempted a proof by induction on the number of points, claiming that if the greedy algorithm works on all sets of n points, it will also work on all sets of $n+1$ points. This approach won't work; it essentially goes *backward* from the induction of the optimal substructure property. Still, it could earn as many as 2 points: 1 for the base case and 1 for analyzing cases on the last interval. Your case analysis had to be along the following lines to get this point: either the last interval—not necessarily an interval beginning at x_n!—contains x_{n+1}, or it doesn't, and in the latter case, either $x_{n+1} - x_n > 1$ or $x_{n+1} - x_n \le 1$. A few other solutions tried a proof by contradiction, and may have earned 1 point for limiting consideration to the *smallest* set of n points that the algorithm doesn't work for.

A few other common errors were the following. In the greedy choice proof, some students wanted to show that $[x_1, x_1+1]$ *must* be in an optimal cover. Some students tried to vary the *sequence* in which intervals are chosen, noting, say, that the choice of the interval $[x_2, x_2+1]$ before $[x_1, x_1+1]$ would not lead to a better cover. This, however, doesn't say anything about covers produced in other ways. Finally, some students concluded incorrectly in a proof by contradiction that if an interval $[x_k, x_k+1]$ was not in an optimal cover, then x_k was not included in any of the intervals of the cover.

Reduction of Subgraph Isomorphism to Maximum Common Induced Subgraph

We're still working out the details of this.