Notes for Recitation 4 (2/27/98)

Scheduling Courses

We describe an application of graphs to a scheduling problem. The example also highlights a special kind of digraph called a *directed acyclic graph* or *DAG*; a DAG is simply a digraph that does not contain a (directed) cycle.

We consider a simple kind of scheduling problem – for MIT courses with prerequisites. The problem is to determine how a student could complete all of the courses in the fewest terms, under the simplifying, if unlikely, assumption that there is no limit on the number of courses a student can take in a term as long as their prerequisites have all been met. We model this problem with a digraph. An example is shown in Figure 1.

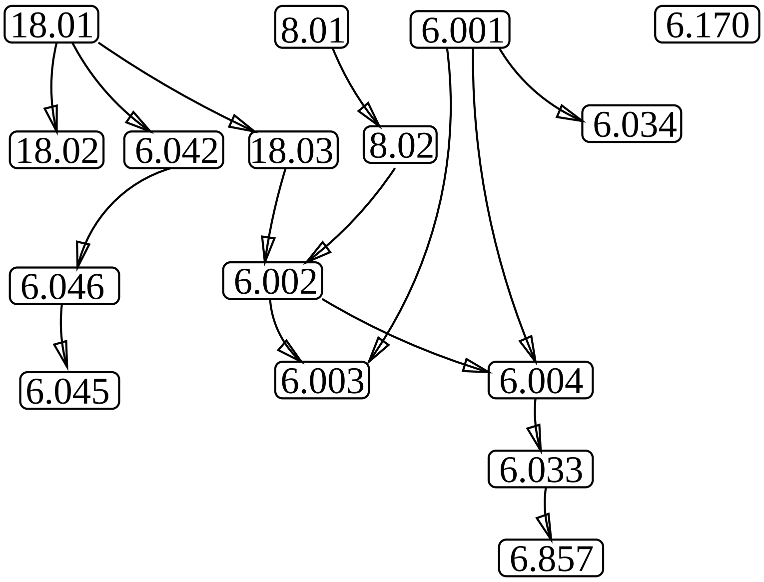

Figure 1: *This is a picture of a directed acyclic graph. Vertices are labeled with course names. An edge from u to v indicates that course u is a prerequisite for course v.*

This example is a DAG. Of course it is possible that the faculty in charge of courses created a cycle of prerequisites; this would mean that it was impossible to fulfill the prerequisites of any of the courses in the cycle before taking the course. Presumably this would not be the intention of the faculty, so we assume that the whatever the prerequisite requirements may be, they do not have cycles; so the digraph corresponding to the prerequisites will always be a DAG.

The connectedness relation of a DAG is a *partial order*. The example in Figure 1 suggests the reason for this name. The arcs in this graph restrict the order in which a student may take classes.

For example, 6.001 must be taken before 6.003. Of course, the student still has some choice about the order; for this reason, the set of classes is said to be "partially" ordered by this system of prerequisites. If we write $u \succeq v$ to mean that $u = v$ or there is a directed path from vertex u to v, then the relation $\succeq$ is obviously *transitive*:

$$[u \succeq v \wedge v \succeq w] \implies u \succeq w,$$

and *antisymetric*:

$$[u \succeq v \wedge v \succeq u] \implies u = v.$$

Formally, a partial order on a set is just a binary relation which is transitive and antisymetric[1].

Two elements u and v are said to be *comparable* under a partial order, $\succeq$, iff $u \succeq v \vee v \succeq u$. When the partial order is the one from connectedness in a DAG, vertices u, v are comparable precisely when $u = v$ or there is a directed path from u to v or vice-versa. Otherwise, u and v are incomparable. For example, 18.01 and 6.002 are comparable since there is a directed path from 18.01 to 6.002. However, 18.01 and 8.01 are incomparable because there is no directed path from one to the other.

Problem: How many terms are necessary to complete all of the courses in Figure 1, assuming prerequisites are observed, but course load is no issue?

The answer is suggested by the way Figure 1 is drawn. Each term the student could take all the courses in a horizontal row, starting with the first row and working down. All of the courses in each row are incomparable, and all the necessary prerequisites are in previous rows. Since there are six rows, the student can finish in six terms.

Furthermore, there is no way to finish in fewer than six terms, since each term the student can take at most one course in the six-course sequence 18.01, 18.03, 6.002, 6.004, 6.033, 6.857. Notice that this sequence constitutes the longest directed path in the graph.

Class exercise: Explain why every (finite) DAG has a longest path.

Solution: If path was as long as number of vertices, V, a vertex would repeat and this would be a cycle. So the length of a path is at most $|V| - 1$. So the set

$$\{|V| - (k+1) : k \text{ is the length of a path in the DAG}\}$$

is a nonempty set of natural numbers, and by the Well-ordering Principle, it has a least element k_{min}. Then $|V| - k_{min}$ is nonempty the length of the longest path. QUIBBLE: why is the set nonempty? Actually it might be empty if the graph has no edges; to cover this case we should regard a single vertex as a path of length 0.

A *chain* in a partial order, $\succeq$, is a single element or a nondecreasing sequence of elements. That is, $a_1, a_2, \ldots, a_n$ is a chain providing $a_{i+1} \succeq a_i$ for all $1 \leq i < n$. So every path in a DAG is a chain in the partial order determined by connectedness in the DAG. An *antichain* is a set of elements no two of which are comparable. In a DAG, an antichain is a set of vertices with no paths connecting any two of them. Any antichain in the course prerequisite DAG identifies a set of courses that can be taken in any order, or simultaneously, as long as their other prerequisites have been met.

[1] Rosen also requires that a partial order be *reflexive*: $v \succeq v$ for all v. In 6.042 we will not always insist on reflexivity, so the relation "there is a directed path of *positive* length from vertex u to v," will also count as a partial order. But above we have defined the connectedness relation of a DAG so it is in fact reflexive.

We let t be the number of vertices in the longest chain in the graph. This chain must be a path (Exercise: Why?). Now at least t terms are necessary to complete all the courses, because at most one course on the chain can be taken each term.

We would now like to show that there is a schedule for taking courses which achieves the minimum number of terms, t. In particular, the following theorem provides the required schedule.

Theorem 0.1 *If the longest path in a DAG G has length $t-1$, then the vertices can be partitioned into a sequence of t blocks such that any edge into a vertex must start at a vertex from an earlier block.*

This theorem implies that t terms are sufficient. In term i, the student should take the set of courses in the ith block. Since the edges into any block come from an earlier block, all the prerequisites of courses in a block have been taken by the time the block is scheduled to be taken. And since the blocks partition the set of all courses, the student will complete every course.

Class exercise: Explain why each block must be an antichain.

Proof. Define the *depth* of a vertex to be the length of the longest path in the DAG which ends at the vertex (vertices with in-degree zero have depth zero). (There always is such a longest path by the same argument used above to find a longest path in the whole DAG.) For $0 \leq i < t$, let the ith block, A_i, consist of all vertices of depth i. So every vertex is in a unique block. Moreover, each block is nonempty. In other words, the A_i's partition the set of vertices.

Class exercise: Why is each block nonempty?

The theorem will follow immediately if we show is that for all $a \in A_i$, if there is a path of positive length from b to a, then $b \in A_j$ for some $j < i$. We prove this by contradiction.

So suppose there is a path of positive length from b to a, that $a \in A_i$, and $b \notin A_j$ for all $j < i$. Since b is in some block, A_k, we conclude that $k \geq i$, so there is a path ending at b of length $\geq i$. If we append this path to the positive length path from b to a, we obtain a path ending at a of length $> i$, contradicting the fact that the longest path to a is of length i.

■

Invariants, correctness, and termination of state machines

Examples:

1. Fast Exponentiation: compute r^n for $r \in \mathbb{R}, n \in \mathbb{N}$.

   State machine with states in $(a, b, k) \in \mathbb{R} \times \mathbb{R} \times \mathbb{N}$.

   Initial state: $(1, r, n)$

   Next state:
   if $k > 0$ and k is even, then $(a, b, k) \rightarrow (a, b^2, k/2)$;
   if $k > 0$ and k is odd, then $(a, b, k) \rightarrow (ab, b, k-1)$.

   Invariant: $ab^k = r^n$

   Termination: k decreases.

2. 15 puzzle. Square tiles numbered $1, \ldots, 15$ in a 4×4 array with one blank square. Any tile adjacent to the blank square can slide into it.

   The state machine model of the puzzle has states consisting of a sequence (permutation), p, of $1, \ldots, 15$ (indicating the positions of the numbered tiles reading left-to-right and then top-to-bottom), with a nonnumerical token, b, inserted somewhere in the permutation to indicate where the blank square is. For example, the initial position

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	b

   is represented by $1, 2, \ldots, 15, b$, and the target position

15	14	13	12
11	10	9	8
7	6	5	4
3	2	1	b

   is represented by $15, 14, \ldots, 1, b$.

   Next state: Exchange the blank square and an adjacent tile. For example blank at square $(2, 2)$ exchanges position with tile above at $(1, 2)$:

   $$\begin{array}{|c|c|c|c|}\hline n_1 & n_2 & n_3 & n_4 \\ \hline n_5 & b & n_6 & n_7 \\ \hline n_8 & n_9 & n_10 & n_11 \\ \hline n_12 & n_13 & n_14 & n_15 \\ \hline \end{array} \longrightarrow \begin{array}{|c|c|c|c|}\hline n_1 & b & n_3 & n_4 \\ \hline n_5 & n_2 & n_6 & n_7 \\ \hline n_8 & n_9 & n_10 & n_11 \\ \hline n_12 & n_13 & n_14 & n_15 \\ \hline \end{array}$$

   corresponds to the state change

   $$(n_1 n_2 n_3 n_4 n_5 \ b \ n_6 n_7 \ldots n_{15}) \to (n_1 \ b \ n_3 n_4 n_5 n_2 n_6 n_7 \ldots n_{15})$$

   We prove that starting from the initial state, there is no way to reach the target state. For any permutation, p, of the integers $1, \ldots, n$, let inversions(p) be the number of out-of-order pairs there are in p. For example, with $n = 5$, the permutation 12453 has two out-of-order pairs: 43 and 53. The in-order sequence 12345 has no out-of-order pairs. More generally,

   $$\text{inversions}(1, 2, \ldots, n) = 0 \text{ and inversions}(n, n-1, n-2, \ldots, 1) = n(n+1)/2.$$

   Invariant: The parity of $k ::= \text{inversions}(p) + r$ where p is the permutation of $1, 2, \ldots, 15$ (ignoring the b), and r is the row number of the blank square. (Row one is at the top, and the blank is initially at the bottom, so initially $r = 4$.)

   Notice that for the initial position, inversions$(1\ 2\ \ldots 15) = 0$ and $r = 4$, so k_{init} is even. For the target position, inversions$(15\ 14 \ldots 2\ 1) = 14 + 13 + \cdots + 1 = (15 \cdot 14)/2 = 105$ and $r = 4$ so k_{target} is odd, so the invariant implies the target pattern is not reachable from the initial one.

To show that the value of k is constant, note that horizontal moves change nothing, and vertical moves both change r by 1, and move a tile three places forward or back in the current permutation, p. This reverses the order of three pairs in p, changing the number of inversions by 3 or 1, but always by an odd amount.

3. Think-a-dot: flip-flops the three rows of 3,2,3 (need picture to show connections). Invariant is parity of total number of left-leaning flip-flops in the top and bottom rows – because a marble dropped in any of the three holes on top must fall thru, and therefore flip, exactly one flip-flop in each of the top and bottom rows (skips the middle row half the time). In fact, if two patterns of the 8 flip-flops have the same parity, one can be turned into the other by a series of marble drops.

4. **Extended GCD** may have been done in some recitations instead of the one of the previous ones. It is an interesting example of the invariant method, and it also establishes a very useful fact about gcd's: the gcd of two integers, a and b, can be expressed as a linear combination with integer coefficients of a and b.

   Given integers a, b the following procedure returns the gcd of a and b, and also coefficients $k, l \in \mathbb{N}$ so $ka + lb = \gcd(a, b)$. (This procedure and its invariants are taken from Aho, Hopcroft, and Ullman's text on algorithms.)

```
u := 1; v := 0; k := 0; l := 1; x := a; y := b;
loop:
if y|x DONE;
q := floor(x/y);
** SIMULTANEOUS ASSIGNMENT:
 x := y,
 y := x - q * y, **same as x (mod y)
 u := k,
 v := l,
 k := u - q * k,
 l := v - q * l;
goto loop;
```

   NOTE: x, y behave exactly as in the standard Euclidean algorithm from lecture, except that this procedure stops one step sooner. This procedure terminates for the same reason as the Euclidean algorithm: y decreases (fast actually) each time around the loop.

   The invariants we need to justify this procedure are:

   (a) $\gcd(x, y) = \gcd(a, b)$,

   (b) $ka + lb = y$, and

   (c) $ua + vb = x$.

   The procedure terminates when $y|x$. Then invariants (a) and (b) imply $\gcd(a, b) = \gcd(x, y) = y = ka + lb$. So we have the gcd in register `y` and the desired coefficients in `k`, `l`. To verify the invariants:

Proof. Invariant (a) is the same one as for the Euclidean algorithm, which we proved in lecture. To check the other two invariants, note that at the start $k = 0, l = 1, y = b$ and $0a + 1b = b$ as required; also $u = 1, v = 0, x = a$ and $1a + 0b = a$ as required.

So we now assume all the invariants hold for u, v, k, l, x, y, and prove they hold for new values u', v', k', l', x', y'. Invariant (c) holds for $u' = k, v' = l, x' = y$ because of invariant (b) for k, l, y. Now $k' = u - qk$ and $l' = v - ql$, where $q = \lfloor x/y \rfloor$, so

$$k'a + l'b = (u - qk)a + (v - ql)b = ua + vb - q(ka + lb) = x - qy = y',$$

so invariant (b) holds for k', l'. ∎

Team Problems

Problem 1. Fast multiplication by adding and doubling from F97 pset4, problem 2.

Consider the following multiplication algorithm, known as the "Russian peasant's multiplication" to multiply numbers $x, y \in \mathbb{N}$.

The accumulator for the answer is register `a`; registers `r`, `s` are for temporary storage.

```
r:=x; s:=y; a:=0;
Do until s=0:
 if even(s) then {r := 2r; s := s/2};
 if odd(s) then {a := a + r; r := 2r; s := (s-1)/2};
```

The answer xy will be left in the accumulator `a` when the procedure terminates.

Here is an example of the execution of the algorithm.

x	y	r	s	a
5	9	5	9	0
		10	4	5
		20	2	5
		40	1	5
		80	0	45

Model the computation as a state machine. Prove it terminates using a derived variable. Prove that it is correct using an invariant.

Solution: The state is given by the value in (r, s, a). x and y can be included as part of the state or viewed as universal constants.

There are two types of state transitions (even or odd) as described in the algorithm. The set of initial states consists of the single state $q_0 = (x, y, 0)$.

Theorem 0.2 *The algorithm terminates in at most $\lceil \log_2 y \rceil$ steps.*

Proof. Define the derived variable $f(r,s,a) = \lceil \log_2 s \rceil$. Notice that since at each step s' becomes $\frac{s}{2}$ if s is even and $\frac{s-1}{2}$ if s is odd, f is decreasing. So the length of an execution is $\leq f(q_0) = f(x,y,0) \leq \lceil \log_2 y \rceil$. ■

We prove the invariant $P(r,s,a) : xy = rs + a$.

Base case: In the initial state $r = x$ and $s = y$, so $P(x,y,0)$ asserts that $xy = xy + 0$, which is true.

Inductive step: Consider a transition from (r,s,a) to (r',s',a'). We need to prove that $P(r,s,a) \implies P(r',s',a')$. There are two cases based on the type of action, *even* or *odd.*

even: Doubling r while halving s doesn't change the product rs, and a doesn't change. So $r's'+a' = rs + a = xy$.

odd: $r' = 2r$, $s' = (s-1)/2$, and $a' = a + r$. So $r's' + a' = 2r(s-1)/2 + a + r$. Reorganizing terms yields $r's' + a' = rs + a = xy$. ■

Corollary: When $s = 0$, $a = xy$.

Proof. By our invariant, we have $P(r,0,a)$, that is $r0 + a = xy$, so $a = xy$. ■

Problem 2. Unisex Marriage stability counterexample.

We might better call this the Buddy Pairing Problem: an even number of men, each with a preference ranking of all the others, to be paired as "buddies" in a stable way.

Derive an example of preferences for this unisex version of the mating problem which shows that it is *not* possible to always pair all the players so that the marriages satisfy the stability criterion. *Hint*: Consider just four men.

Solution: Let the buddy preferences of four men A, B, C, D be
A: BCD
B: CAD
C: ABD
D: ? ? ?

Let Y and Z be any two among A, B, and C, and let X be the remaining one. Notice that the first choices of A, B, C are arranged in a cycle, so that X must be the first choice of either Y or Z.

Now for any matching into pairs, let Y and Z be the two among A, B, C who are paired, and X be the one paired with D. So one of Y and Z, say Y, has X as his first choice. But since A, B, C each rank D last, X prefers Y to his own buddy, D. So X, Y are a rogue couple. Notice that the ranking of A, B, C by D doesn't matter.

Problem 3. The judge says to the guilty defendant,

> "Here is what is called a *Petition Process*:
>
> > You may name a number of times you may petition me. Your petitions may request a number of days to wait until you file your next petition, or until this whole petition process ends. Your petitions will be granted. When the number of days named in your last petition is reached, the process will end.

> Now you may request a number of petition processes in which you would like to participate. Your request will be granted, but at the end of the last process, your head will be cut off."

The defendant responds, "I want to participate in 2 processes." The judge says "OK", and the defendant asks to submit 4 petitions. The judge says "OK," and the defendant files a petition asking for 10 days. The judge says "OK." 10 days later, the defendant files a petition asking for 20 days. 20 days later the defendant asks for 30 days, and 30 days later for 40 days.

40 days later, he starts his second petition process, this time asking to submit 3 petitions. The first petition is for 100 days. 100 days later, he asks for 200 days. 200 days later he asks for 300 days. 300 days later his head gets cut off `:-(`.

Of course a smarter prisoner could have named much larger numbers and lived out his natural lifetime. But for the sake of argument, suppose the prisoner (and the judge and hatchetman) will live as long as it takes to get executed. Prove that the prisoner will always eventually get his head cut off.

Solution: By induction on the number of petition processes the defendant first chooses. The induction hypothesis is itself proved by a subinduction on the number of petitions chosen at the start of a process. The subinduction hypothesis is proved by induction on the number of days the defendant chooses to wait before submitting the next petition.

Problem 4. Consider the Stable Marriage Problem when there are more men than women. Define what a set of stable marriages should mean in the presence of bachelors. Show that the Morning-Afternoon-Evening Protocol described in lecture solves this Marriage Problem.

Solution: The definition of *rogue couple* is extended to include a pair consisting of a bachelor and a married woman who prefers the bachelor to her spouse. A solution is a set of marriages in which all woman are married (so all bachelors are men) and there are no rogue couples. The proof that the protocol yields a solution is now a routine repeat of the case of equal numbers of men and women from lecture.