CS1510 Greedy Problem 15, Dynamic Programming 1

Rebecca Negley, Sean Myers

September 16, 2011

15. Stable Marriage Problem

(a) Give an upper bound as a function of n of the number of stages in this algorithm.

Solution: The algorithm completes in $\leq [n * (n-2) + 2]$ stages, so the algorithm is bounded by n^2. At least one man must get rejected at every stage before completion (or else the algorithm will halt because every woman will be paired to one man). Observe that there cannot be a woman with a man on her porch at stage i but no man on her porch at stage $i+1$ because the only way her suitor will leave her porch is if he is kicked off by a more preferable suitor. Then, at least one woman must not have any man on her porch until the concluding stage (because, again, the algorithm will halt if every woman has a match). This gives that the algorithm must halt the step after some man has been rejected $(n-1)$ times because he will go to the last woman who has had no former suitors, and every woman will then have a match. Therefore, the worst case is that only one man gets rejected at each stage before completion, and each man first gets rejected (n-2) times. Next, one man must get rejected for an (n-1)st time, forcing that man to move to the last woman, the one with 0 men on her porch. Thus, at stage $[n * (n-2) + 2]$, every man must be with his final match.

(b) A marriage assignment is stable if there does not exist a man x and a woman y such that x prefers y to his assigned mate, and y prefers x to her assigned mate. Clearly adultery is a risk if a marriage assignment is not stable. Prove that this algorithm leads to a stable marriage.

Solution: Suppose the algorithm leads to some unstable marriage. Then, y prefers x to his mate, and x prefers y to her mate for some man x and some woman y. Since y prefers x to his mate, y would have visited x's porch before his mate's porch. Then, x must have kicked y off of her porch. However, this means x chose a man y_2 whom she prefers more than y. Then, the only way y_2 would have been removed is if x found someone she preferred even more. However, this means that x must have preferred her eventual mate more than y, which contradicts the assumption. Hence, the algorithm leads to a stable marriage.

(c) A stable marriage M is man optimal if for every man x, M is the best possible stable marriage. That is, in every stable marriage other than M, x ends up with a woman no more preferable to him than the woman he is married to in M. Prove or disprove the above algorithm produces a man optimal stable marriage.

Solution: The algorithm is man optimal. Suppose not. Then, for some man x_1, there exists a more preferable stable marraige, M_b, than the one returned by the greedy algorithm, M_a. This means that, supposing the M_a pairs x_1 with y_1, there is some y_2 more preferable than y_1 with

whom x_1 can have a stable marriage in M_b. Then, since x_1 prefers y_2 to y_1, he would have visited y_2 before y_1 in the greedy algorithm and eventually been rejected. y_2 would have ended up paired with x_2 in M_a, whom y_2 prefers to x_1. Then, in order to prevent an unstable marriage in in M_b, x_2 must have ended up with someone he prefers to y_2. Clearly, this pattern emerges of x_j needing to be paired with a more preferable woman than y_j, whom he was paired with in M_a. This will repeat until: a) x_j prefers y_j to every other woman and therefore can find no more preferable woman, giving a contradiction, or b) x_j prefers y_1 to y_j and completes a cycle. For simplicity, let $y_p = y_{pmod(h)}$ where x_h is matched with y_1 in M_b so that $y_{h+1} = y_1$. If a cycle is completed, every man in the cycle is with a woman he prefers more than the one he is paired with in M_a. Then, for each x_i in the cycle, x_i visits y_{i+1} (his new partner in M_b) before visiting y_i (his partner in M_a) in the greedy algorithm. Then, each x_i must get rejected by y_{i+1} in the greedy algorithm. Some man x_k must be rejected by y_{k+1} in favor of x', where $x' \neq x_{k+1}$. (If $x' = x_{k+1}$, some other man must have caused x_{k+1} to be rejected from y_{k+2}, so we can find where this chain of events starts.) Hence, y_{k+1} prefers x' to x_k. Similarly, x' prefers y_{k+1} to y', his partner in M_a. If x' is paired with y' in M_b, this creates an unstable marriage, which is a contradiction. Then x' must be matched with someone more preferable to y_{k+1} in M_b to avoid the unstable marriage. However, this leads to the same condition above as above. We will either find a contradiction or find another cycle. Since there are a finite number of men, we can continue with this pattern until every man is matched with a more preferable woman in M_a than in M_b (or we reach a contradiction). Then, supposing we have not yet reached a contradiction, we must have some man x_{k2} rejected by y_{k2+1} in favor of $x_{k3} \neq x_{k2+1}$ in the greedy algorithm. Otherwise, no man could have started the cycle of kicking the men off the more preferable stable porches. Then, x_{k2} goes to y_{k2}, causing y_{k2-1} to reject x_{k2-2}. However, one of two things must eventually happen. Either, x_{k3} is in the same loop, and when x_{k3+1} gets rejected by x_{k3+2} and goes to y_{k3+1}, y_{k3+1} has no mate and accepts him. Then, there are some men in the loop who are still with their more preferable women in M_a, which is a contradiction. The other option is that x_{k3} is in a different loop and the loop current completes itself, kicking x_{k3} out. However, this cannot happen with every loop because eventually the disturbing man is a member of the last loop. Hence, there must be some men who are with their more preferable women in M_a, which contradicts our assumption of who they were originally with. Hence, M_a is man optimal.

(d) A stable marriage M is woman optimal if for every woman y, M is the best possible stable marriage. That is, in every stable marriage other than M, y ends up with a man no more preferable to her than the man she is married to in M. Prove or disprove the above algorithm produces a woman optimal stable marriage.

Solution: Consider men x_1, x_2 and women y_1, y_2. Suppose x_1 prefers y_1 to y_2, x_2 prefers y_2 to y_1, y_1 prefers x_2 to x_1, and y_2 prefers x_1 to x_2 The algorithm pairs x_1 with y_1 and x_2 with y_2 to give stable marriage M_a. Consider the stable marriage M_b, though, where x_1 is paired with y_2 and x_2 with y_1. Then, both woman have the most preferable match, so the marriage must be stable, and each woman prefers her mate in M_b to her mate in M_a. Then, we can produce an example where there exists at least one woman who is not in the best possible stable marriage in M_a, so M_a is not woman optimal.

(e) A stable marriage M is man pessimal if for every man x, M is the worst possible stable marriage. That is, in every stable marriage other than M, x ends up with a woman no less preferable to him than the woman he is married to in M. Prove or disprove the above algorithm produces a man pessimal stable marriage.

Solution: Consider the same counterexample as part (d). In M_b, both x_1 and x_2 end up with less preferable mates than in M_a, so we can produce an example where there exists at least one man who is not in the least preferable stable marriage in M_a, so M_a is not man pessimal.

(f) A stable marriage M is woman pessimal if for every woman y, M is the worst possible stable marriage. That is, in every stable marriage other than M, y ends up with a man no less preferable to her than the man she is married to in M. Prove or disprove the above algorithm produces a woman pessimal stable marriage.

Solution: If a woman ends up with someone worse, that means the man she ended up with in M_a ends up with someone better (otherwise there would be an unstable marriage). Since we know from (c) that we cannot have a man ending up in a stable marriage with someone better, we have that the algorithm produces a woman pessimal stable marriage.

1. Consider the recurrent relation $T(0) = T(1) = 2$ and for $n > 1$

$$T(n) = \sum_{i=1}^{n-1} T(i)T(i-1)$$

We consider the problem of computing $T(n)$ from n.

(a) Show that if you implment this recursion directly in say the C programming language, that the program would use exponentially, in n many arithmetic operations.

Solution: The C recursion would look like:

```
dyn(int n):
    if(n ==1 or n == 0): return 2
    sum = 0
    for (i = 1; i ¡ n; i++):
        sum+= dyn(i) * dyn(i-1)
    }
    return sum
```

For this problem, we can see that at each iteration of the for loop, there will be a binary branching. Let us only focus on the largest tree that is produced, which would be a tree of $dyn(n-1) * dyn(n-2)$. In each recursive call, let us solely focus on the largest branching factor in the for loop– so only the last iteration. At each recursive call, the largest possible branch level will be n. With a branching factor of two (more if we considered the rest of the for loop). So that means the largest branch factor of the leftmost branch in all the for loop levels, will be 2^n. The least of the branches will be $2^{n/2}$, this coming from the fact that dyn(n-2) removes twice as fast as dyn(n-1). That means the operations of just the last iterations of every recursive level is $2^(n-1)/2) \leq ops \leq 2^(n-1)$. It can be concluded that since a small subset of the algorithm already has an exponential set of operations, that the entire algorithm is greater or equal to exponential as well.

(b) Explain how, by not recomputing the $T(i)$ value twice, one can obtain an algorithm for this problem that only uses $O(n^2)$ arithmetic operations.

Solution: So if we only calculate $dyn(i)$ once, that calculation will require some $k * i$ operations, but the k is negligible, so let's not worry about that. So the number of operations will be:

$$\sum_{i=1}^{n-1} k * i_{ops}$$

This summation is well known to be $O(n^2)$ operations.

(c) Give an algorithm for this problem that only uses $O(n)$ arithmetic operations.

Solution:

```
Dyn(int n):
    int Dyn[n+1]
    Dyn[0] = 2
    Dyn[1] = 2
    Dyn[2]= Dyn[0]*Dyn[1]
    for(int i = 3; i ¡= n; i++):
        Dyn[i] = Dyn[i-1]+Dyn[i-1]*Dyn[i-2]
    return Dyn[n]
```