CS70 In Simpler Terms - Note 2

Kevin Liu

June 25, 2017

1 Stable Marriage Theorem

In this section I will provide a few key tips and examples for you to remember. I will not run through the algorithm, since you can refer to the course notes for that.

Important things to remember:

1. There are no rogue couples, and if it's a male optimal pairing, then it must be a female pessimal pairing.
2. A stable marriage pairing is always *stable* and always *paired*, thus the name.
3. Halting Lemma: The algorithm terminates within n^2 days, where n is the number of pairs.
4. Improvement Lemma: the man that a woman has on a string can only get more preferable over time.
5. When tackling Stable Marriage problems, look to make use of contradiction, induction, or rogue couples in your proofs.

Examples:

M	F
A	$1 > 2$
B	$2 > 1$

F	M
1	$B > A$
2	$A > B$

M	F
A	1
B	2

F	M
1	B
2	A

MALE PREFERENCES AND FEMALE PREFERENCES ARE EXACT OPPOSITES
Male Preferences, Female Preferences, Male Optimal Pairing, Female Optimal Pairing, respectively.

M	F
A	$1 > 2 > 3$
B	$1 > 2 > 3$
C	$1 > 2 > 3$

F	M
1	$A > B > C$
2	$B > A > C$
3	$C > B > A$

M	F
A	1
B	2
C	3

ALL MALE PREFERENCES THE SAME
Male Preferences, Female Preferences, Male Optimal Pairing, respectively.

M	F
A	$1 > 2 > 3$
B	$1 > 2 > 3$
C	$2 > 1 > 3$

F	M
1	$C > B > A$
2	$A > B > C$
3	$A > B > C$

M	F
A	2
B	3
C	1

NO MALE GETS FIRST CHOICE
Male Preferences, Female Preferences, Male Optimal Pairing, respectively.

M	F
A	$2 > 1 > 3$
B	$3 > 2 > 1$
C	$1 > 3 > 2$

F	M
1	$B > A > C$
2	$C > B > A$
3	$A > C > B$

M	F
A	2
B	3
C	1

M	F
A	3
B	1
C	2

M	F
A	1
B	2
C	3

3 STABLE PAIRINGS
Male Preferences, Female Preferences, Male Optimal Pairing, Female Optimal Pairing, Stable Pairing.

The next few sections are mostly just definitions that you need to know. Other than being comfortable working with these terms and putting them on your cheat sheet, there isn't much else here.

2 Traveling Directions

1. Path - any sequence of edges where no vertex is repeated.
2. Walk - Like a *path*, but vertices can be repeated.
3. Cycle - Like a *path* that starts and ends on the same vertex.
4. Tour - Like a *walk* that start and ends on the same vertex.
5. Eulerian tour- a *tour* that visits all edges once (no repeating edges, but can repeat vertices). Graph must be undirected, have even degree, and be connected.
6. Hamiltonian/Rudrata cycle- a *cycle* that visits each vertex once.

Often misunderstood definitions: *A Path is a Walk, and a Cycle is a tour.*

3 Graphs and Trees

1. Sum of degrees in a graph is $2e$, so it must be even.
2. Tree - an undirected graph (n-vertex tree):
   - $n - 1$ edges
   - connected, *no cycles*
   - remove 1 edge, and the tree becomes disconnected
   - add 1 edge and it creates a cycle in the tree

- sum of all degrees = $2(n-1)$
- More likely than not, for proofs regarding the existence of a path in a tree, try to use the above information and proof by contradiction.

3. Planar Graph - a graph that can be drawn without crossings.

   - no K_5 or $K_{3,3}$
   - number of faces is equivalent to the number of cycles.
   - $v + f = e + 2$
   - $\sum_{i=1}^{f} s_i = 2e$
   - $e \leqslant 3v - 6$ - *Know how this is derived!*
     Derivation: *Every face has at least 3 sides, so* $s_i \geqslant 3$
     Therefore, from the above equation, $2e \geqslant 3f$
     Since $f = e + 2 - v$*, subtitute* $f \Rightarrow 2e \geqslant 3e + 6 - 3v$
     Rearranging, we get the desired result: $e \leqslant 3v - 6$
     This result tells us that planar graphs can't have too many edges.

4. Complete Graph K_n - a graph with the maximum number of edges possible

   - has $\frac{n(n-1)}{2}$ edges.
   - the opposite of a tree, which has the minimum number of edges.

5. Hypercubes - Whenever you see a problem about a hypercube, just think: *Bit Strings*! Each time you go to a neighboring vertex, you are just flipping a bit in a bit string. The following definitions apply to a hypercube with dimension n, which can be thought of as an n-bit string.

   - each vertex has degree n
   - $n * 2^{n-1}$ edges
   - 2^n vertices (you have n bits, and each bit can be either 1 or 0)
   - $(n+1)$th dimension hypercube = combine 2 n-dimension cubes