CPSC 365 Notes

1/13/15

goals/foci of the course: 1. provably correct algorithms 2. faster algorithms 3. seemingly hard problems 4. truly hard problems (NP-complete) 5. deal with these problems by identifying right sub-problem(s) 6. asymptotic (large inputs) view of everything - example: Linear Program solvers + Bixby studied difference in LP solvers btwn. '88 and '03 + found 44k fold increase in speed over that interval * 1k - hardware * 43k - algorightms

topics of the course 0. stable matchings 1. greedy algorithms - easiest to design/implement 2. divide and conquer - break problem into parts - solve individual parts 3. dynamic programming - akin to wizardry 4. flow problems 5. NP-completeness - often sound like "give me the *best* answer to something" 6. approximation algorithms 7. randomized algorithms

mechanics - prerequisites 1. CPSC 202/MATH 244 2. CPSC 223 (not strictly necessary) - book: **Kleinberg & Tardos** - 2 tests (tentatively, no final) - 8 problem sets (majority of course's work/evaluation) + each set contains 4 problems + the problems will be hard and require insight. - **limited** collaboration is allowed + "Gilligan's island policy": ok to talk to people to understand the problem + no taking notes during these meetings + all solutions should be written alone and collaborators reported.

Stable Perfect Matchings

- n people, n jobs
- goal: assign each person to a job
- each person ranks the jobs 1-n in order of decreasing preference
- matching: a set of (person,job) pairs such that each person <= 1 job and each job <= 1 person
- perfect: everyone gets a job <=> every job gets a person
- instability: a pair (p,j) and (p',j') s.t. p prefers j' to j and j' prefers p to p'
- stable: matching that contains no instabilities

jobs = {x,y,z} poeple = {a,b,c}

	1	2	3
a	x	y	z
b	x	y	z
c	z	x	y

	1	2	3
x	b	c	a
y	b	c	a
z	c	a	b

(a,x) (a,y) (b,y) (b,x) is stable (c.x) (c,z) **Theorem:** there is always a perfet stable matching

- Init: everyone unmatched, S:=null
- while (there exists a job that is unmatched and has not been proposed to every person):
- let j be such a job
- let p be person ranked highest by j, to whom j has not yet proposed.
- propose(j,p)

def propose(j,p): - if p unmatched, **accept** + add (p,j) to S - else, let j' be match of p + if p prejers j to j' * **accept**: add (p,j) to S and remove (p,j') from S + else * **reject**: no change

Example: - x –> b, **accept** S:=(b,x) - y –> b, **reject** S:=(b,x) - y –> c, **accept** S:=(b,x),(c,y) - z –> c, **accept** S:=(b,x),(c,z) - y –> a, **accept** S:=(a,y),(b,x),(c,z)

Terminates: only a finite number of possible proposals, and none happens twice.

Observation 1: once a person is matched, (s)he stays matched, and only gets better pairs. if (p,j) in output, p prefers job j to any job that proposed to p.

Lemma 2: output is a matching. number of unmatched people equals the number of unmatched jobs. so, if not perfect, there must be an unmatched person and an unmatched job. by observation 1, j never proposed to p. algorithm couldn't have stopped. contradiction.