Notes

- Office hours: Tuesdays 2pm-4pm
- TA hours:
  - Cecilia Chen, Saturdays 6pm-8pm, Park231
- First assignment out (due next Monday)
- Electronic submission on Moodle
- LaTeX tutorial and templates on course website
- Sample write-up on Moodle

1

CS 340 - Analysis of Algorithms
Stable Matching Analysis

Dianna Xu

Stable Matching

- Input
  - A set of employers: E
  - A set of applicants: A
  - $|E| = |A| = n$
  - $2n$ preference lists, each of size n
  - How are the preference lists stored?
    - linked lists? arrays? hash maps? why?
- Output: M, a list/set of n pairs

3

Gale-Shapley Algorithm

all e in E and a in A are unpaired
while there is an e unmatched that hasn't made an offer to every a:
  choose such an e
  let a be the highest-ranked applicant in the preference list of e, to whom e has not made an offer
  if a is unpaired then pair a and e
  else a is currently paired with some e'
    if a prefers e' to e then e remains unpaired
    else a is paired with e and e' becomes unpaired
return the set of pairs

Example

Employers			Applicants		
Google(G)	Intel(I)	Apple(A)	Kate(K)	Clara(C)	Lisa(L)
K	K	L	A	A	G
C	L	K	G	I	I
L	C	C	I	G	A

G→K	[G-K]
I→K	rejected
I→L	[G-K, I-L]
A→L	rejected
A→K	[A-K, I-L]
G→C	[A-K, I-L, G-C]

5

Analysis

- What can we say about the progression of matches from the employers point of view?
- The applicants point of view?
- How many iterations might the algorithm execute?
- Is the matching always stable?

Observations

- Employers: applicants only get worse
- Applicants: jobs only get better and never lose a job once they have one

Proof of termination

Proof:

- An employer only proposes to a new applicant and never again
- There are at most n^2 such pairings

n^2 iterations of the while loop

Proof of correctness

- Will all employers and applicants be matched? - Is the matching perfect?

Proof (by contradiction):

- Suppose there is some employer e who didn't hire.
- Then there is some applicant a who doesn't get a job.
- So e didn't offer a a job. But e was required to offer everyone a job.

Contradiction

9

Proof of correctness: Stability

- Is the matching always stable?

Proof of stability (by contradiction):

- Suppose $e - a$ is an unstable pair. (What does this mean?)
- $e - a$ and $e' - a'$ are both unstable because e and a' want to switch
- How could this have happened?
  - Case 1: e never offered a' a job
  - Case 2: e *did* offer a' a job

Analysis

- So what is the running time of this algorithm?
  - while loop runs n^2 iterations
  - make data structure decisions and analyze based on data structure operation costs
  - consult the writeup sample provided

Gale-Shapley Again

all e in E and a in A are unpaired
while there is an e unmatched that hasn't made an offer to every a:
  choose such an e
  let a be the highest-ranked applicant in the preference list of e, to whom e has not made an offer
  if a is unpaired then pair a and e
  else a is currently paired with some e'
    if a prefers e' to e then e remains unpaired
    else a is paired with e and e' becomes unpaired
return the set of pairs

Data Structures

- Employers are assigned n unique int IDs $e_1, \dots, e_m$
- Applicants are assigned n unique int IDs $a_1, \dots, a_n$
- Preference lists are 2D `int` arrays that are indexed by the IDs, storing rank as integers

13

Data Structures

Applicant plists:

- a_k: $e_3, e_{n-1}, e_n, \dots, e_2, e_1$
- 2D array of size $n \times n$:
  - $P[a_k][e_1] = n$
  - $P[a_k][e_2] = n - 1$
  - $P[a_k][e_3] = 1$
  - …
  - $P[a_k][e_{n-1}] = 2$
  - $P[a_k][e_n] = 3$
- if a_k prefers e_i to e_j
  - `if (P[a_k][e_i]<P[a_k][e_j])`

Employer plists: n sorted arrays of size n

M: integer array of size n, indexed by the applicants and stores matched employer IDs

Queue of size n for offer making

14

Analysis

- Is this algorithm better for the employers or the applicants?
- What does "better" mean?
  - evaluated *collectively*
- Answer by lab
  - examples of different stable matchings for the same input, together with exact preference lists
  - precise definition of "better"

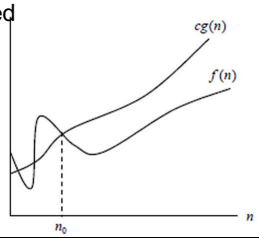
CS 340 - Analysis of Algorithms Complexity

Dianna Xu

Big O

$\exists\, n_0 \geq 0, c > 0$, if $f(n) \leq c \cdot g(n) \forall n \geq n_0$, then $f(n) = O(g(n))$

- Constant factors are ignored
- Upper bound
- $\lim_{n \to \infty} \frac{f(n)}{g(n)} \leq c$



How do these functions grow?

- $f_1(x) = 43x^2 \log^4 x + 12x^3 logx + 52xlogx = O(x^3 logx)$
- $f_2(x) = 15x^2 + 7xlog^3 x = O(x^2)$
- $f_3(x) = 3x + 4\log_5 x + 91x^2 = O(x^2)$
- $f_4(x) = 13 \cdot 3^{2x+9} + 4x^9 = O(9^x)$
- $f_5(x) = \sum_{i=0}^{x} \frac{1}{2^i} = O(1)$

19

Useful Facts

- polynomials grow faster than polylogs
  - $\lim_{n\to\infty} \frac{(logn)^a}{n^b} = 0\,, a, b > 0$
- exponentials grow faster than polynomials
  - $\lim_{n\to\infty} \frac{n^a}{b^n} = 0, a > 0, b > 1$
- log bases do not matter
  - $\lim_{n\to\infty} \frac{\log_a n}{\log_b n} = c \neq 0, a, b > 1$
- exponential bases do matter
  - $\lim_{n\to\infty} \frac{a^n}{b^n} = 0, 1 < a < b$

20

Big Ω

$\exists\, n_0 \geq 0, c > 0$, if $f(n) \geq c \cdot g(n) \forall n \geq n_0$, then $f(n) = \Omega(g(n))$

- Constant factors are ignored
- Lower bound

Big Θ

if $f(n) = O\big(g(n)\big)$ and $f(n) = \Omega\big(g(n)\big)$ then $f(n) = \Theta(g(n))$

- Constant factors are ignored
- Tight bound

Relatives

Notation	Relational Form	Limit Definition
$f(n) = o(g(n))$	$f(n) \prec g(n)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$
$f(n) = O(g(n))$	$f(n) \preccurlyeq g(n)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c, 0$
$f(n) = \Theta(g(n))$	$f(n) \approx g(n)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$
$f(n) = \Omega(g(n))$	$f(n) \succcurlyeq g(n)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c, \infty$
$f(n) = \omega(g(n))$	$f(n) \succ g(n)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$

23

$f(n) \prec\approx\succ g(n)$?

- $f(n) = 3^{\frac{n}{2}}, g(n) = 2^{\frac{n}{3}}$
- $f(n) = \log(n^2), g(n) = (logn)^2$
- $f(n) = n^{log4}, g(n) = 2^{2logn}$
- $f(n) = \max(n^2, n^3), g(n) = n^2 + n^3$
- $f(n) = \min(2^n, 2^{1000}n), g(n) = n^{1000}$

24

Summations

- Constant series
$$\sum_{i=a}^{b} 1 = \max(b - a + 1, 0)$$
- Arithmetic series
$$\sum_{i=1}^{n} i = 1 + 2 + \cdots + n = \frac{n(n+1)}{2} \in \Theta(n^2)$$
- Geometric series
$$\sum_{i=0}^{n} c^i = 1 + c + c^2 \ldots + c^n = \frac{c^{n+1} - 1}{c - 1} \in \begin{cases} \Theta(1), c < 1 \\ \Theta(c^n), c > 1 \end{cases}$$

25

Summations

- Quadratic series

$$\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + \cdots + n^2 = \frac{2n^3 + 3n^2 + n}{6}$$

$\in \Theta(n^3)$

- Linear-geometric series

$$\sum_{i=1}^{n} ic^i = c + 2c^2 + \cdots + nc^n$$

$$= \frac{(n-1)c^{n+1} - nc^n + c}{(c-1)^2} \in \Theta(nc^n)$$

26

Summations

- Harmonic series

$$H_n = \sum_{i=1}^{n} \frac{1}{i} = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} = \ln n + O(1)$$

27

Time Analysis

- Line-by-line analysis of pseudo code to write a polynomial
  - sequential work is summed
    - justify any line that is not clearly $O(1)$
  - loops (# iterations × work in body)
    - number of iterations
    - justify any line in the body that is not clearly $O(1)$
- Simplify to the dominant terms and state final big-O

28

Linear Time Algorithms: $O(n)$

- The algorithm's running time is at most a constant factor times the input size
- Process the input in a single pass spending constant time on each item
  - Max/min algorithm, linear search
- Charging scheme, each item of input is charged once
  - Merge two already sorted lists

$O(nlogn)$ time

Frequent running time in cases when algorithms involve:

- Sorting
- Divide and conquer

Quadratic Time: $O(n^2)$

- All pairs in a list
- Every pair of points is processed
  - Nearest neighbor
- Nested loops

$O(n^k)$ Time

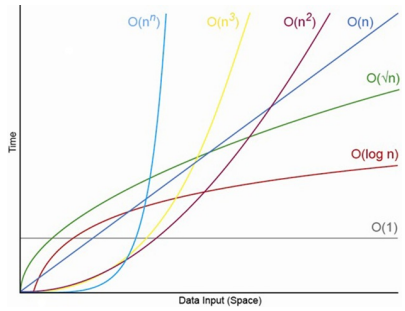
- Consider all subsets of points of size k

Slower than Polynomial Time

- All subsets of sets (power set): $O(2^n)$
- Number of ways to match n items with each other: $O(n!)$

Sublinear Time

- Query in a binary search tree: $O(\lg n)$
- In general, if we can throw away a *constant fraction* of the input with each step of the algorithm, we can achieve sublinear time.



36