CSC 421: Algorithm Design & Analysis

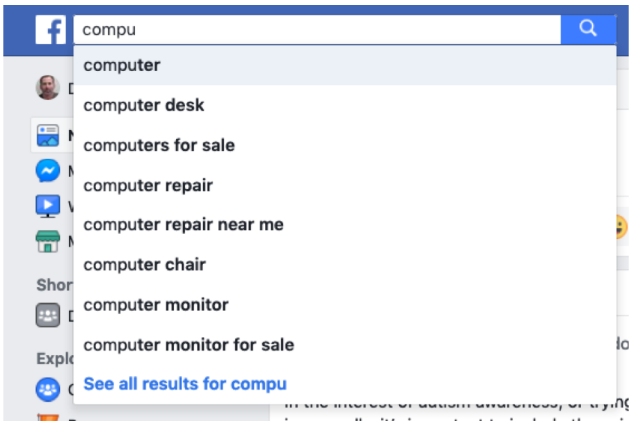
Spring 2019

Algorithm case studies

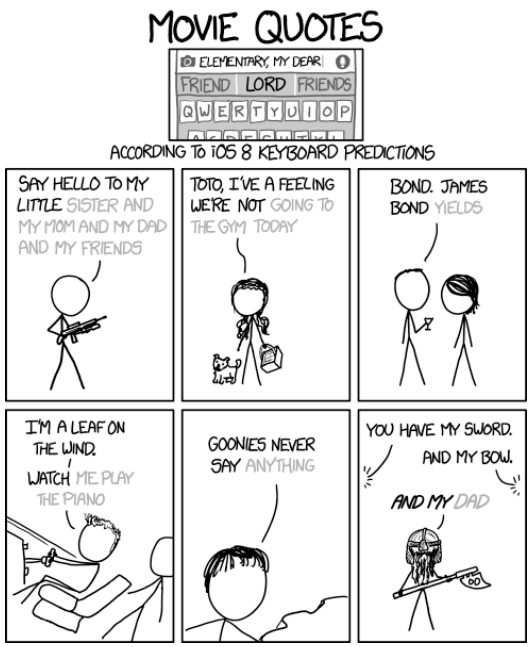
- Facebook code completion
- Google search rankings
- National Resident Matching Program

Facebook keyboard predictions

like many applications, Facebook utilizes keyboard predictions to suggest search terms as the user types



xkcd



Facebook keyboard predictions

how does Facebook do this quickly and at scale?

BINARY SEARCH!

- Facebook maintains a sorted list of keywords/phrases (let's call it POSSIBLES)
- when you type in the first letter, e.g., 'c',
  1. it performs a binary search to find the first item that starts with that letter
  2. it similarly performs a binary search to find the last item
  3. it reduces POSSIBLES to the sublist between these two words
- when you type each subsequent letter, it does the same thing
  1. perform binary search to find first item that starts with that sequence
  2. perform binary search to find the last item that starts with that sequence
  3. further reduce POSSIBLES

source: a Facebook engineer at the Facebook Faculty Summit in 2010

Facebook example

for example, suppose

POSSIBLES = [about, bell, cave, coat, compost, computer, cult, dog, elder, …]

- when the user types 'c':
  - ✓ binary searches to find "cave" and "cult"
  - ✓ reduces POSSIBLES to [cave, coat, compost, computer, cult]
- when the user types 'co':
  - ✓ binary searches to find "coat" and "computer"
  - ✓ reduces POSSIBLES to [coat, compost, computer]
- when the user types 'com':
  - ✓ binary searches to find "compost" and "computer"
  - ✓ reduces POSSIBLES to [compost, computer]

. . .

5

Efficiency?

if the initial size of the POSSIBLES list is N,

- each of the initial binary searches take O(log N)
- reducing the size of the POSSIBLES list can be done in constant time, simply by keeping low and high indices of the sublist

➔ cost of the initial 1-letter prediction is O(log N)

➔ even if POSSIBLES contains a million keywords/phrases, only requires <100 comparisons to determine & narrow the range

note that with each letter typed, the POSSIBLE list shrinks

- if the words/phrases are evenly distributed (is that a reasonable assumption?), each step would reduce the size of the list by a factor of 26
- if started with 1 million keywords/phrases,
  1,000,000 → 38,462 → 1,480 → 57 → 3

6

Google PageRank

one of the keys to Google's rise to domination of the search engine market was the superior quality of its searches

- Larry Page & Sergey Brin developed the underlying technology (including the PageRank algorithm) as part of their dissertation work at Stanford
- their main goals were to improve search quality & support data-science research

the PageRank algorithm measures the importance of a Web site and is used when ranking search results

- works by counting the number and quality of links to a page
- *basic idea*: the more sites that find a page useful, especially "important" sites, the more important the page is
- the PageRank algorithm outputs a probability distribution that models the likelihood that a person randomly clicking on links will arrive at the page
  (clearly, having lots of other sites link to a page increases its likelihood, as does having the page linked to by popular sites)

7

Simplified PageRank algorithm

terminology:

PR(p) = the PageRank for page p
In(p) = all the pages that link to p
Out(p) = all the outgoing links in u

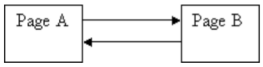
initially, all pages are assigned PR = 1

then, repeatedly iterate over pages and update ranks using this formula

$$PR(p) = .15 + .85 \sum_{q \in In(p)} \frac{PR(q)}{|Out(q)|}$$

the process will converge so that the average PageRank for each page is 1.0

suppose only two pages, which link to each other



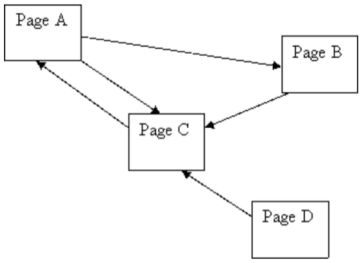
initially, PR(A) = 1.0, PR(B) = 1.0

$$PR(A) = (1 - .85) + .85\left(\frac{1.0}{1}\right) = .15 + .85 = 1.0$$
$$PR(B) = (1 - .85) + .85\left(\frac{1.0}{1}\right) = .15 + .85 = 1.0$$

8

Another example

suppose 4 pages, as shown on right



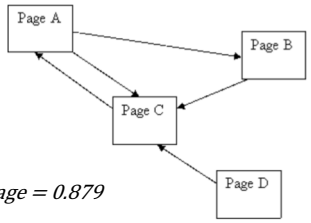
$PR(D) = .15 + .85(0) = 0.15$
$PR(A) = .15 + .85\left(\frac{1.0}{1}\right) = 1.0$
$PR(B) = .15 + .85\left(\frac{1.0}{2}\right) = 0.575$
$PR(C) = .15 + .85\left(\frac{1.0}{2} + \frac{.575}{1} + \frac{.15}{1}\right) = 1.191$ *average = 0.729*

$PR(D) = .15 + .85(0) = .15$
$PR(A) = .15 + .85\left(\frac{1.191}{1}\right) = 1.162$
$PR(B) = .15 + .85\left(\frac{1.162}{2}\right) = 0.644$
$PR(C) = .15 + .85\left(\frac{1.162}{2} + \frac{.644}{1} + \frac{.15}{1}\right) = 1.319$ *average = 0.819*

$PR(D) = .15 + .85(0) = .15$
$PR(A) = .15 + .85\left(\frac{1.319}{1}\right) = 1.271$
$PR(B) = .15 + .85\left(\frac{1.271}{2}\right) = 0.690$
$PR(C) = .15 + .85\left(\frac{1.271}{2} + \frac{.690}{1} + \frac{.15}{1}\right) = 1.404$ *average = 0.879*

9

Another example



$PR(D) = .15 + .85(0) = .15$
$PR(A) = .15 + .85\left(\frac{1.319}{1}\right) = 1.271$
$PR(B) = .15 + .85\left(\frac{1.271}{2}\right) = 0.690$
$PR(C) = .15 + .85\left(\frac{1.271}{2} + \frac{.690}{1} + \frac{.15}{1}\right) = 1.404$ *average = 0.879*

$PR(D) = .15 + .85(0) = .15$
$PR(A) = .15 + .85\left(\frac{1.404}{1}\right) = 1.343$
$PR(B) = .15 + .85\left(\frac{1.343}{2}\right) = 0.721$
$PR(C) = .15 + .85\left(\frac{1.343}{2} + \frac{.721}{1} + \frac{.15}{1}\right) = 1.461$ *average = 0.919*

.
.
.

(after ~20 iterations)

$PR(D) = 0.15$
$PR(A) = 1.49$
$PR(B) = 0.78$
$PR(C) = 1.58$

NOTE:
- C has the highest PR, since it is linked from 3 pages
- PR(A) > PR(B), since A is linked from the highest ranked page & B only gets half the credit from A
- PR(D) remains at 0.15 since it is not linked at all

average = 1.0

10

Google search results

obviously, more than just PageRank goes into determining the content & order of search results, e.g.,

- frequency and proximity of search terms in the page
- occurrence of search terms in links to the page
- timeliness of the page
- other factors that are kept proprietary

still, the heart of Google's search remains the PageRank algorithm

- Brin & Page patented the PageRank algorithm
- they generously assigned the patent to Stanford University, then gave Stanford 1.8 million shares of Google stock in exchange for exclusive use
- Stanford sold the shares when Google went public in 2004 & 2005 for a total of $336 million
- if they had kept the stock, it would be worth $2.18 billion today!

11

National Resident Matching Program

each year, the National Resident Matching Program matches 40,000+ med school graduates with residency programs

- each graduate ranks programs by order of preference
- each program ranks students by order of preference

pairing graduates & programs in a way that makes everyone (reasonably) happy is an extremely complex task

- want to ensure that the pairings are *stable*, i.e., no grad and program would prefer each other over their assigned matches

  e.g., suppose G_1 listed $P_1 > P_2$; and P_1 listed $G_1 > G_2$

  the match $\{G_1 \rightarrow P_2, G_2 \rightarrow P_1\}$ is unstable, since both G_1 and P_1 would prefer $G_1 \rightarrow P_1$

since 1952, the NRMP has utilized an algorithm for processing all residency requests and assigning stable matches to graduates

(this general problem is known as the *stable matching* or *stable marriage problem*)

12

Stable matching example

can specify preferences by two tables of rankings

grad's preferences			
	1st	2nd	3rd
G_1:	P_2	P_1	P_3
G_2:	P_2	P_3	P_1
G_3:	P_3	P_2	P_1

program's preferences			
	1st	2nd	3rd
P_1:	G_2	G_3	G_1
P_2:	G_3	G_1	G_2
P_3:	G_2	G_3	G_1

1st	2nd	3rd		1st	2nd	3rd
G_1: P_2	P_1	P_3		P_1: G_2	G_3	G_1
G_2: P_2	P_3	P_1		P_2: G_3	G_1	G_2
G_3: P_3	P_2	P_1		P_3: G_2	G_3	G_1

$G_1 \rightarrow P_1$, $G_2 \rightarrow P_2$, $G_3 \rightarrow P_3$ is unstable
- G_1 would prefer P_2 over P_1
- P_2 would prefer G_1 over G_2

1st	2nd	3rd		1st	2nd	3rd
G_1: P_2	P_1	P_3		P_1: G_2	G_3	G_1
G_2: P_2	P_3	P_1		P_2: G_3	G_1	G_2
G_3: P_3	P_2	P_1		P_3: G_2	G_3	G_1

$G_1 \rightarrow P_1$, $G_2 \rightarrow P_3$, $G_3 \rightarrow P_2$ is stable

13

Stable match algorithm (Gale-Shapley)

1. start with all the grads and programs being unassigned
2. while there are unassigned grads, select an unassigned grad (G_u):
   a. have G_u choose the next program on their preference list (P_n)
   b. if P_n is unassigned, it (tentatively) accepts G_u
   c. otherwise, it compares G_u with its current match (G_m)
      i. if P_n prefers G_u to G_m, it switches its assignment to G_u (releasing G_m)

1st	2nd	3rd		1st	2nd	3rd
G_1: P_2	P_1	P_3		P_1: G_2	G_3	G_1
G_2: P_2	P_3	P_1		P_2: G_3	G_1	G_2
G_3: P_3	P_2	P_1		P_3: G_2	G_3	G_1

suppose we select G_1
G1 chooses P_2
P_2 is unassigned, so it accepts G_1

1st	2nd	3rd		1st	2nd	3rd
G_1: P_2	P_1	P_3		P_1: G_2	G_3	G_1
G_2: P_2	P_3	P_1		P_2: G_3	G_1	G_2
G_3: P_3	P_2	P_1		P_3: G_2	G_3	G_1

suppose we next select G_2
G_2 chooses P_2
P_2 is already assigned G_1 and prefers G_1

1st	2nd	3rd		1st	2nd	3rd
G_1: P_2	P_1	P_3		P_1: G_2	G_3	G_1
G_2: P_2	P_3	P_1		P_2: G_3	G_1	G_2
G_3: P_3	P_2	P_1		P_3: G_2	G_3	G_1

$G_1 \rightarrow P_2$ assignment remains
G_2 is still unassigned

14

Stable match algorithm (Gale-Shapley)

G_1: P_2 P_1 P_3 G_2: P_2 P_3 P_1 G_3: P_3 P_2 P_1	P_1: G_2 G_3 G_1 P_2: G_3 G_1 G_2 P_3: G_2 G_3 G_1	suppose we select G_2 again G_2 next chooses P_3 P_3 is unassigned, so it accepts G_2
G_1: P_2 P_1 P_3 G_2: P_2 P_3 P_1 G_3: P_3 P_2 P_1	P_1: G_2 G_3 G_1 P_2: G_3 G_1 G_2 P_3: G_2 G_3 G_1	we now select G_3 G_3 chooses P_3 P_3 is already assigned G_2 and prefers G_2
G_1: P_2 P_1 P_3 G_2: P_2 P_3 P_1 G_3: P_3 P_2 P_1	P_1: G_2 G_3 G_1 P_2: G_3 G_1 G_2 P_3: G_2 G_3 G_1	$G_2 \rightarrow P_3$ assignment remains G_3 is still unassigned
G_1: P_2 P_1 P_3 G_2: P_2 P_3 P_1 G_3: P_3 P_2 P_1	P_1: G_2 G_3 G_1 P_2: G_3 G_1 G_2 P_3: G_2 G_3 G_1	we again select G_3 G_3 next chooses P_2 P_2 is already assigned G_1 but prefers G_3
G_1: P_2 P_1 P_3 G_2: P_2 P_3 P_1 G_3: P_3 P_2 P_1	P_1: G_2 G_3 G_1 P_2: G_3 G_1 G_2 P_3: G_2 G_3 G_1	switch to $G_3 \rightarrow P_2$ G_1 is now unassigned

15

Stable match algorithm (Gale-Shapley)

G_1: P_2 P_1 P_3 G_2: P_2 P_3 P_1 G_3: P_3 P_2 P_1	P_1: G_2 G_3 G_1 P_2: G_3 G_1 G_2 P_3: G_2 G_3 G_1	we now select G_1 G_1 chooses P_2 P_2 is already assigned G_3 and prefers G_3
G_1: P_2 P_1 P_3 G_2: P_2 P_3 P_1 G_3: P_3 P_2 P_1	P_1: G_2 G_3 G_1 P_2: G_3 G_1 G_2 P_3: G_2 G_3 G_1	$G_2 \rightarrow P_3$ assignment remains G_1 is still unassigned
G_1: P_2 P_1 P_3 G_2: P_2 P_3 P_1 G_3: P_3 P_2 P_1	P_1: G_2 G_3 G_1 P_2: G_3 G_1 G_2 P_3: G_2 G_3 G_1	we again select G_1 G_1 now chooses P_1 P_1 is unassigned, so it accepts G_1 now have stable match: $\{G_1 \rightarrow P_1, G_2 \rightarrow P_3, G_3 \rightarrow P_2\}$

16

Analysis of the Gale-Shapley Algorithm

the algorithm produces a stable matching in no more than N^2 iterations

the stable matching produced is always *graduate-optimal*, meaning each grad gets the highest rank program on his/her list under any stable matching

- the grad-optimal matching is unique for a given set of grad/program preferences
- originally, the NRMP used a variant of this algorithm with the roles reversed, producing a *program-optimal* matching

1st 2nd 3rd	1st 2nd 3rd	1st 2nd 3rd	1st 2nd 3rd	1st 2nd 3rd	1st 2nd 3rd
G_1: P_2 P_1 P_3	P_1: G_2 G_3 G_1	G_1: P_2 P_1 P_3	P_1: G_2 G_3 G_1	G_1: P_2 P_1 P_3	P_1: G_2 G_3 G_1
G_2: P_2 P_3 P_1	P_2: G_3 G_1 G_2	G_2: P_2 P_3 P_1	P_2: G_3 G_1 G_2	G_2: P_2 P_3 P_1	P_2: G_3 G_1 G_2
G_3: P_3 P_2 P_1	P_3: G_2 G_3 G_1	G_3: P_3 P_2 P_1	P_3: G_2 G_3 G_1	G_3: P_3 P_2 P_1	P_3: G_2 G_3 G_1

1st 2nd 3rd	1st 2nd 3rd	1st 2nd 3rd	1st 2nd 3rd
G_1: P_2 P_1 P_3	P_1: G_2 G_3 G_1	G_1: P_2 P_1 P_3	P_1: G_2 G_3 G_1
G_2: P_2 P_3 P_1	P_2: G_3 G_1 G_2	G_2: P_2 P_3 P_1	P_2: G_3 G_1 G_2
G_3: P_3 P_2 P_1	P_3: G_2 G_3 G_1	G_3: P_3 P_2 P_1	P_3: G_2 G_3 G_1

1st 2nd 3rd	1st 2nd 3rd	1st 2nd 3rd	1st 2nd 3rd
G_1: P_2 P_1 P_3	P_1: G_2 G_3 G_1	G_1: P_2 P_1 P_3	P_1: G_2 G_3 G_1
G_2: P_2 P_3 P_1	P_2: G_3 G_1 G_2	G_2: P_2 P_3 P_1	P_2: G_3 G_1 G_2
G_3: P_3 P_2 P_1	P_3: G_2 G_3 G_1	G_3: P_3 P_2 P_1	P_3: G_2 G_3 G_1

for this scenario, the stable matching is unique, so program-optimal = grad-optimal

Non-unique example

suppose we changed the preferences of G_1 and P_1

grad's preferences

	1st	2nd	3rd
G_1:	P_1	P_2	P_3
G_2:	P_2	P_3	P_1
G_3:	P_3	P_2	P_1

program's preferences

	1st	2nd	3rd
P_1:	G_1	G_3	G_2
P_2:	G_3	G_1	G_2
P_3:	G_2	G_3	G_1

this scenario has multiple stable matchings

- the grad-optimal matching assigns each graduate their first choice
- the program-optimal matching assigns each program its first choice

graduate-optimal

1st 2nd 3rd	1st 2nd 3rd
G_1: P_1 P_2 P_3	P_1: G_1 G_2 G_3
G_2: P_2 P_3 P_1	P_2: G_3 G_1 G_2
G_3: P_3 P_2 P_1	P_3: G_2 G_3 G_1

program-optimal

1st 2nd 3rd	1st 2nd 3rd
G_1: P_1 P_2 P_3	P_1: G_1 G_2 G_3
G_2: P_2 P_3 P_1	P_2: G_3 G_1 G_2
G_3: P_3 P_2 P_1	P_3: G_2 G_3 G_1

NRMP & Gale-Shapley

from 1952-1997, the NRMP used the program-optimal variant

- after complaints and lawsuits from grads, the algorithm was inverted in 1998 to produce the graduate-optimal matching

the NRMP algorithm now allows for couples to apply together

- with paired couples, a stable matching is NOT guaranteed
- as a result, the algorithm may produce a partial matching, with some left unassigned grads going through additional rounds of selection to the unfilled programs
- in fact, this more complex problem turns out to be nP-complete (LATER)

Lloyd Shapley, along with Alvin Roth, was awarded the 2012 Nobel Prize in Economics for his work and analysis of matching algorithms

19