Data structures and algorithms

Tutorial 10

Amr Keleg

Faculty of Engineering, Ain Shams University

May 8, 2019

Contact: amr_mohamed@live.com

Outline

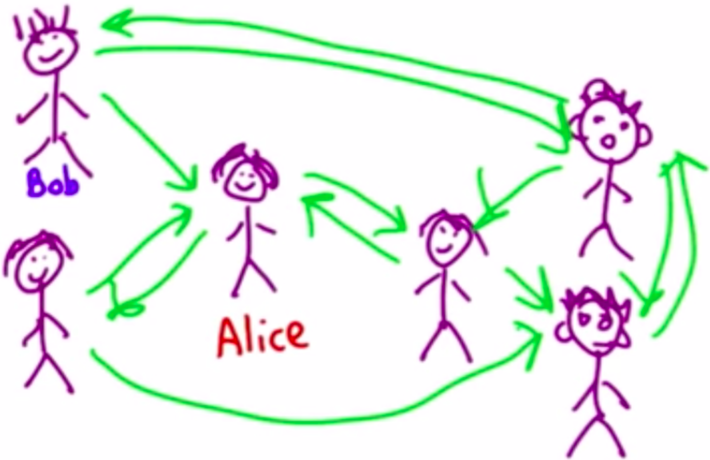
Outline

PageRank (PR) is:

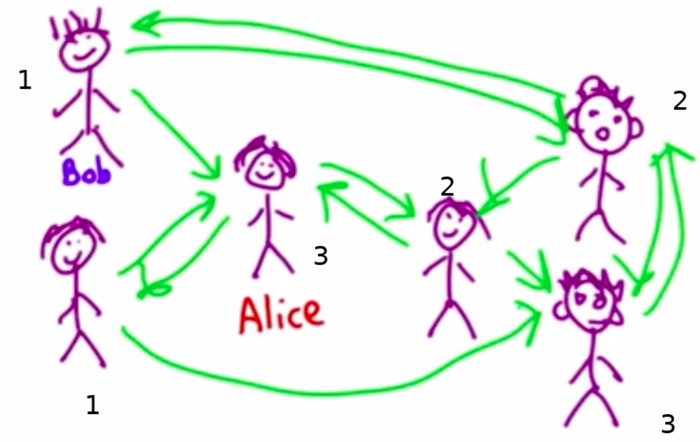
- an algorithm used by Google Search to rank web pages in their search engine results.
- named after Larry Page (one of the founders of Google).

- How to rank the web pages?
- How to give scores to the webpages?

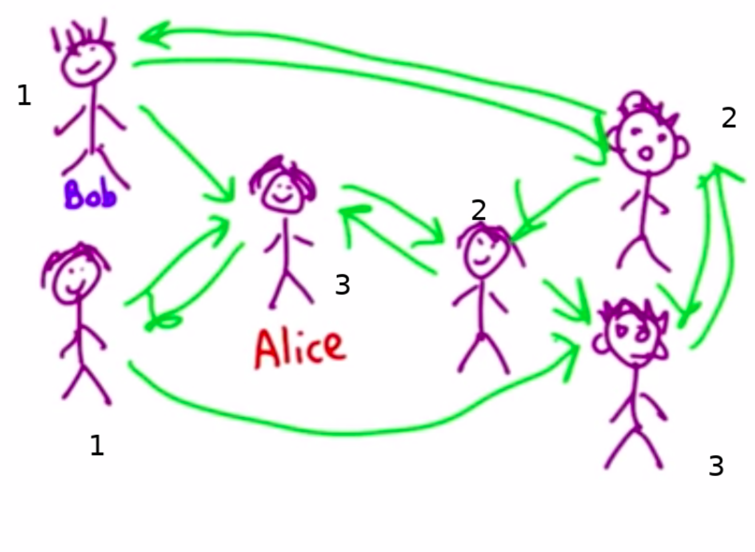
School/University Analogy:
Who is the most popular one?
Note: The fact that A is a friend of B doesn't imply that B is a friend of A.

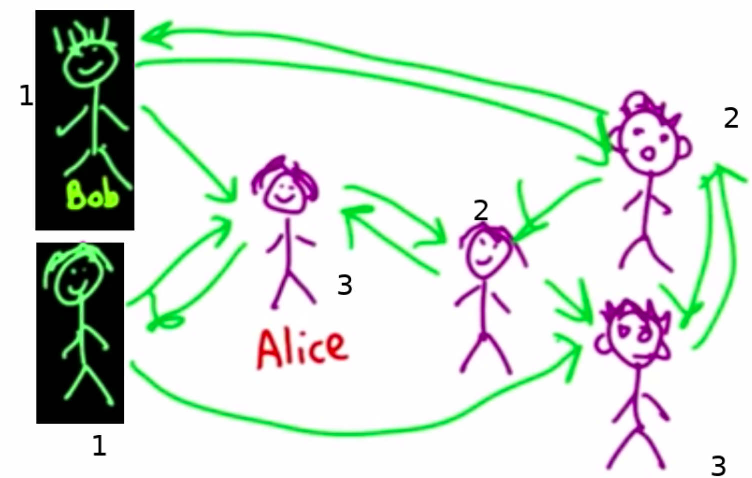


First definition of popularity:
The popularity of student A is proportional to the number of students who consider A to be their friend.
Popularity(A) = No of students who consider A to be their friend



Can we do better?




1
Bob
1
3
Alice
2
2
3

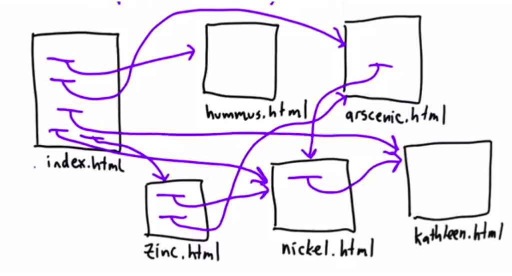
- If you are a friend to non-popular people then you aren't popular.
- If you are a friend of popular people then you are popular.
- If you are the only friend to someone then you should get higher scores.

So, don't just count the number of incoming edges, Give weight to these edges.

$$Popularity(A) = \sum_{s \in B_A} \frac{Popularity(s)}{Ns}$$

- B_A is the set of students who consider A to be their friend.
- Ns is the no of students that s consider to be their friend.

The same idea applies for web pages:
`https://udacity.github.io/cs101x/urank/`



PageRank(hummus.html) = PageRank(index.html) / 4
PageRank(aresenic.html) = PageRank(index.html) / 4 + PageRank(zinc.html) /2

How to compute these dependent values?

Solution:

- Start with a guess for the PageRank for each page.
- Recompute the PageRank given the definition.
- Continue until PageRanks start to converge (don't change).

Outline

For a fair dice, what is the probability that you get a 5?

How to calculate the probability of getting a 5?

- Get a dice.
- Roll it for a very very large number of attempts.
- Count the number of times you get a 5 and divide it by the number of trials.
- Voila!

Let's simulate it.

```
#include <bits/stdc++.h>
using namespace std;

int main(){
  int no_of_times[7] = {};
  int no_of_trials = 1000000;
  for (int i=0; i< no_of_trials; i++){
    int dice_no = 1 + (rand() % 6);
    no_of_times[dice_no]++;
  }

  for (int i=1; i<=6; i++){
    cout<<"P("<<i<<") is "<<
      1.0*no_of_times[i]/no_of_trials<<endl;
  }

  return 0;
}
```

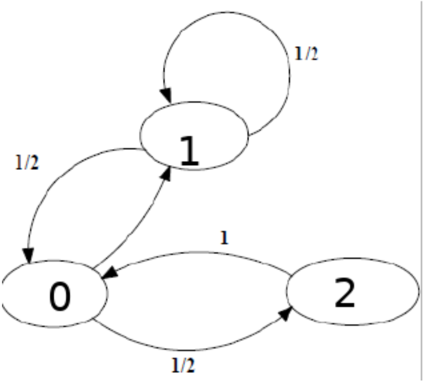
- P(1) is 0.166511
- P(2) is 0.166655
- P(3) is 0.167279
- P(4) is 0.166835
- P(5) is 0.166365
- P(6) is 0.166355

Outline

- A surfer moves through the Internet randomly.
- At first, They enter a URL.
- Then,they follow a series of successive links for a very long time.
- In a random surfer model, it is assumed that the link which is clicked next is selected at random.

The page rank of Page P is the probability that the random surfer will end the walk at page P.

How to calculate the page rank for the following pages?



Simulation code:

```python
import random

nodes = [0, 1, 2]
edges = {0: [1, 2], 1:[0, 1], 2:[0]}
no_of_times = {}

for node in nodes:
  no_of_times[node] = 0

def random_walk(node, timestamp, limit):
  if timestamp == limit:
    no_of_times[node] += 1
  else:
    next_node_indx = random.randint(0, len(edges[node])-1)
    random_walk(edges[node][next_node_indx], timestamp +1, limit)

if __name__ == '__main__':
  no_of_walks = 100000
  max_walk_length = 10

  for _ in range(no_of_walks):
    start_node = random.randint(0, len(nodes)-1)
    random_walk(start_node, 0, max_walk_length)

  for node in nodes:
    no_of_times[node] /= no_of_walks

  for node in nodes:
    print('Probability of ending at node ({}) is: {}'.format(node, no_of_times[node]))
```

Simulation results:

- Probability of ending at node (0) is: 0.38572
- Probability of ending at node (1) is: 0.40654
- Probability of ending at node (2) is: 0.20774

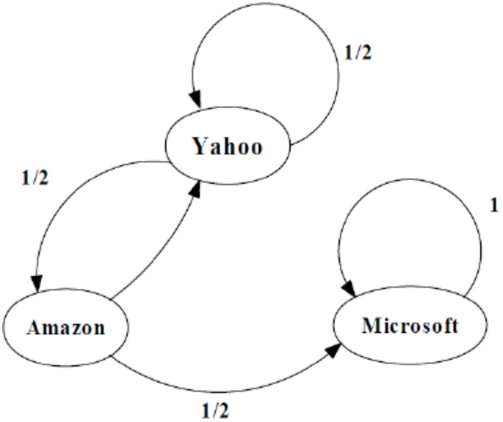
Formal Definition:

- $P(\text{start at page } 0) = 1/3$
- $P(\text{start at page } 1) = 1/3$
- $P(\text{start at page } 2) = 1/3$

What is the probability that the surfer is at page 1 after one click?

- P(page 0, time t) $=$ 0.5 * P(page 1, time t-1) $+$ 1 * P(page 2, time t-1)
- P(page 1, time t) $=$ 0.5 * P(page 0, time t-1) $+$ 0.5 * P(page 1, time t-1)
- P(page 2, time t) $=$ 0.5 * P(page 0, time t-1)

Defects in the model:



Outline

- A surfer moves through the Internet randomly.
- At first, They enter a URL.
- Then,they may follow a series of successive links or use a bookmark to go directly to a webpage.
- In a random surfer model, it is assumed that the link which is clicked next is selected at random.

Probability that the user continues surfing is d.
Probability that the user uses a bookmark is 1-d.

Formal Definition:

- P(start at page 0) $= 1/3$
- P(start at page 1) $= 1/3$
- P(start at page 2) $= 1/3$

What is the probability that the surfer is at page 0 at time t?
How to model the direct navigation to a certain link?

- P(page 0, time t) = 0.5 * P(page 1, time t-1)
- P(page 0, time t) = d * (0.5 * P(page 1, time t-1)) + (1-d) * (1/N)

The final page ranks are:

- PR(Amazon) $= 7/33$
- PR(Yahoo) $= 5/33$
- PR(Microsoft) $= 21/33$

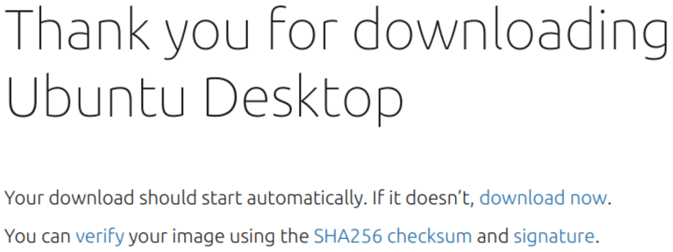
Outline

Outline

- Hashing is a type of algorithm which takes any size of data and turns it into a fixed-length of data.
- Hashing is a one way mapping, e.g: You can find the hash for a certain value BUT You can't find the value given its hash.

Applications:

- Instead of storing passwords in a database, store the corresponding hashes.
- To ensure that a file has been downloaded correctly, `https://www.ubuntu.com/download/desktop/thank-you?version=18.04.2&architecture=amd64`



Outline

The main application that we will discuss is:
Given a set of key-value pairs, store these values such that the searching complexity is optimised.
Example:

- Name: Section
- Chelsea: 1
- Marawan: 3
- Omar: 2
- Mariam: 3
- Nada: 2
- Asim: 1

Options:

```
// Option 1
vector<string> names;
vector<int> section;

// Option 2
map<string, int> section;

// Option 3
unordered_map<string, int> section;
```

How does option 3 really work??

Create an array of size 6 (in practice the array should have a size bigger than the available data).
The hash function will be:

- Map each character to an int (a - 0 , b - 1, c - 2, ...).
- Add the values of the last two characters for each key(name).
- Use the modulus (%6) to make the hash in range 0-5

```
def compute-hash(name):
  name = name.lower()
    score = 0
    for c in name[-2:]:
      score += ord(c) - ord('a')
    return score%6
```

- Name: Hash(Name)
- Chelsea: 4
- Marawan: 1
- Omar: 5
- Mariam: 0
- Nada: 3
- Asim: 2

What if we had only two names:

- Amir
- Marawan

Using the same hash function they will both have hash value of 1!
A COLLISION !
How to solve collisions?

Open Hashing:
Each bucket is a linked list instead of a single index in the array.
(Array of linked lists)
What is the average/worse case complexity for looking for a key?
N items and d buckets.

Closed Hashing:
If there is a collision, find another empty place for the key.

Linear Probing.

TO BE CONTINUED.

Feedback form: `https://forms.gle/XNLtCp29M767ZMY59`