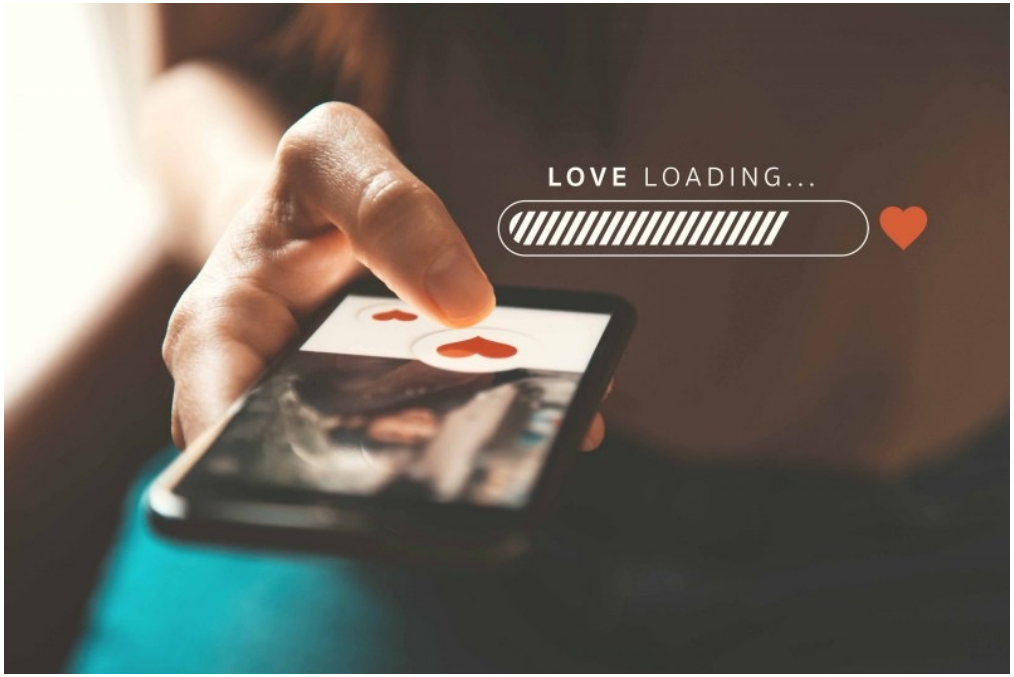
Data Structure Programming Project #4

Dating/Social Networking Service

- Users want:
- High-quality matches
- Low latency of services



Dating/Social Networking Service

- The service servers:
- Obtain users' observable feature
  - Age, gender, location, …
- Fetch users' hidden feature based on history
  - Get a 100-float vector for each user
- Compute similarities between users
- Calculate users' preference based on history
- Generate ranked, suggested matches for recommendations based on user's preference and user similarities

Dating/Social Networking Service

- Filter out already seen recommendations
- We cannot show the same recommendation twice…

- Use set to store seen users:
  - Inefficient comparison: $O(\log n)$ for each recommendation
  - Increasing list size: $O(n)$ for each user

- Not good enough… Is $O(1)$ possible?

Dating/Social Networking Service

- Filter out already seen recommendations
- We cannot show the same recommendation twice…

- In fact, we can risk never showing someone a user he/she hasn't seen
- However, that probability should be low

Programming Project #4: Max Unseen Recommendations

- Given:
  - n User IDs that are input in sequence
  - Limited memory size: m
  - Prime p
- Goal: Maximize the number of admitted unseen recommendations
- Constraint:
  - Cannot show the same user twice
  - Limited memory: $O(m)$ for all recommendations
  - Limited time: $O(1)$ for each recommendation
- The grade is proportional to the # admitted unseen recommendations

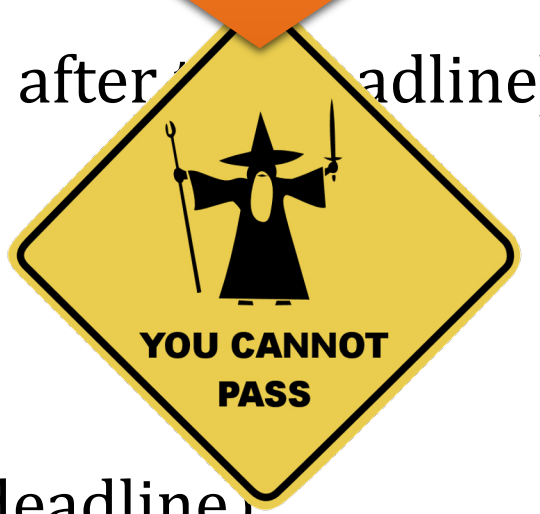
The Competition

- The grade is proportional to the # admitted unseen recommendations
- Basic: 75 (deadline)
  - A feasible solution (no same user twice)
  - The # admitted unseen recommendations is at least that by simple algorithm
- Performance ranking (decided after the deadline)
  - [0%, 50%) (bottom): +0
  - [50%, 75%): + 5
  - [75%, 90%): + 9
  - [90%, 95%): + 12
  - [95%, 100%] (top): + 15
- Homework assistant (superb deadline)
  - +10

The Competition

- The grade is proportional to the # admitted unseen recommendations
- Basic: 75 (deadline)
  - A feasible solution (no
  - The # admitted unseen recomm … by simple algorithm
- Performance ranking (decided after … adline)
  - [0%, 50%) (bottom): +0
  - [50%, 75%): + 5
  - [75%, 90%): + 9
  - [90%, 95%): + 12
  - [95%, 100%] (top): + 15
- Homework assistant (superb deadline)
  - +10

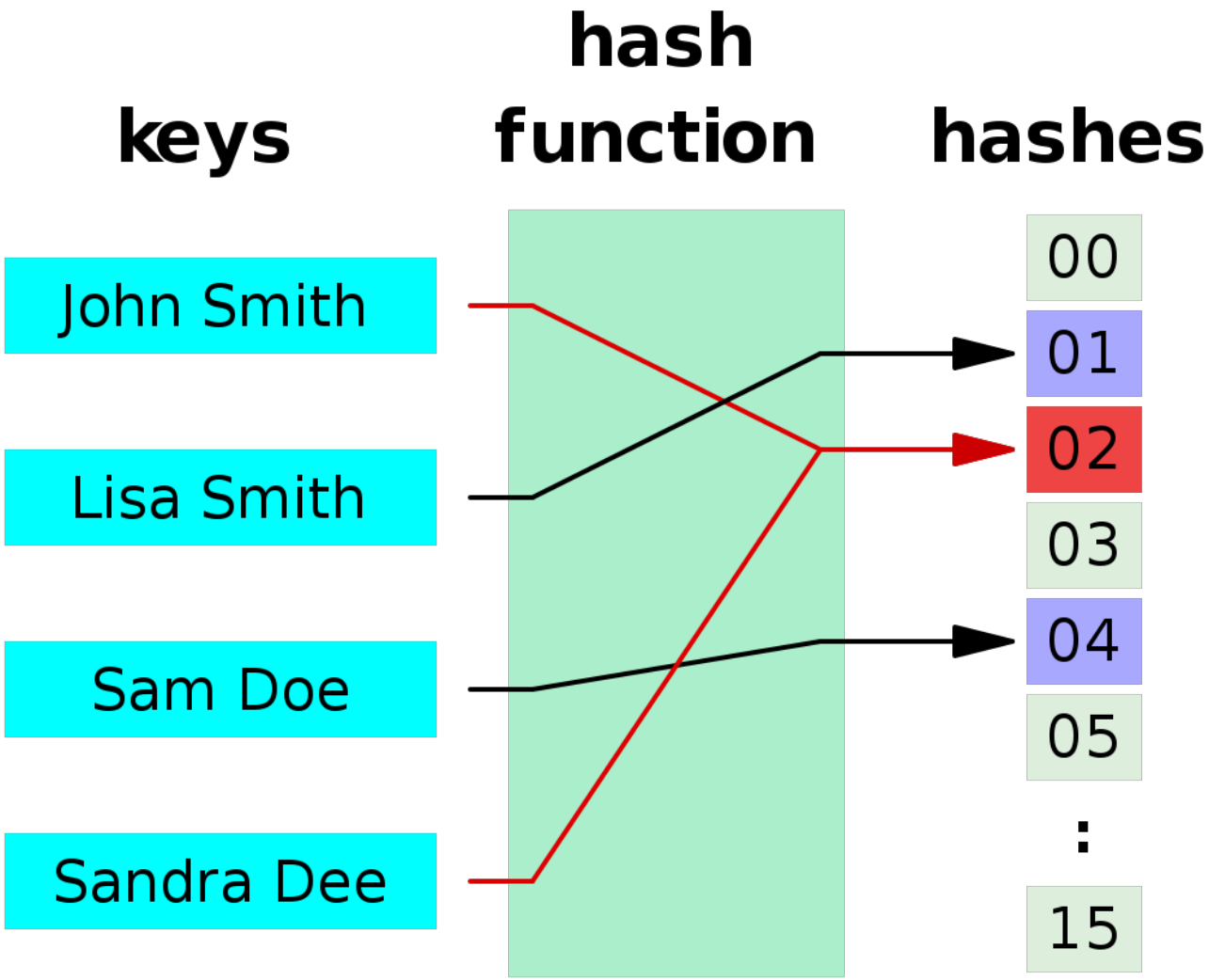
We have MEMORY and TIME LIMIT!



Simple Algorithm

- The trick: don't store every seen user ID
- Create a bit array of length m initially filled with **false** values
- Each incoming recommendation ID gets mapped to a number (i.e., an index) between 0 and $m - 1$ (i.e., hashing)
- The corresponding bit in the array is set to **true**
- To query a recommendation's bit, simply return the bit value at it's hashing position

Simple Algorithm



Some properties of Simple Algorithm

- No same recommendation
- Never show someone a specific user
- Constant memory and a constant time

- Modify Simple Algorithm by any technique
- Note that the results should be **deterministic** (nothing random)

Hashing Function Implementation

unsigned int myhash (unsigned int userID, unsigned int p, unsigned int m)

```
// m is the number of bits in the bit array
1. Return userID * userID % p % m;
```

Note: You are allowed to implement your own hashing function. However, you should not use anything "random" in the homework.

The results should be **deterministic.**

Input Sample: use scanf

```
p m n
UserID1
UserID2
UserID3
…
```

Note: A user's ID may appear twice or more in the sequence

Output Sample: use printf

```
UserID1: bool1
UserID2: bool2
UserID3: bool3
…

Read the user ID from input
Never seen → true (or false)
Seen → false (a must)

Note: The bool value indicates
admitted (true) or denied
(false)
```

Note

- Superb deadline: 1/4 Tue
- Deadline: 1/11 Tue
- Pass the test of our online judge platform
- Submit your code to E-course2
- Demonstrate your code remotely with TA
- **C Source code (i.e., only .c)**
- Show a good programming style