CS2351
Data Structures

Lecture 17:

Hashing I

About this lecture

- The Hashing Problem
- Hash with Chaining
- Hash with Open Addressing

- Choosing a good Hash Function
  ** Universal Hash Function

The Hashing Problem

Hashing Problem

- Let $U = \{ 1, 2, \ldots, u \}$ be a universe

  S = n distinct keys chosen from U
- The Hashing Problem :

To store S such that the following operations can be done efficiently :

Search(x, S) : Is x in S ?

Insert(x, S) : Insert x to S

Delete(x) : Delete x from S

Hashing Problem

- Solution 1: Use a balanced BST
  Operation time : $O(\log n)$
  Space : $O(n)$

- Solution 2: Use an $O(u)$-size array
  Operation time : $O(1)$
  Space : $O(u)$

Hashing Problem

Question:

Can we have a solution that has the benefits of both ? That is, with

Operation time : $O(1)$

Space : $O(n)$ words

Answer :

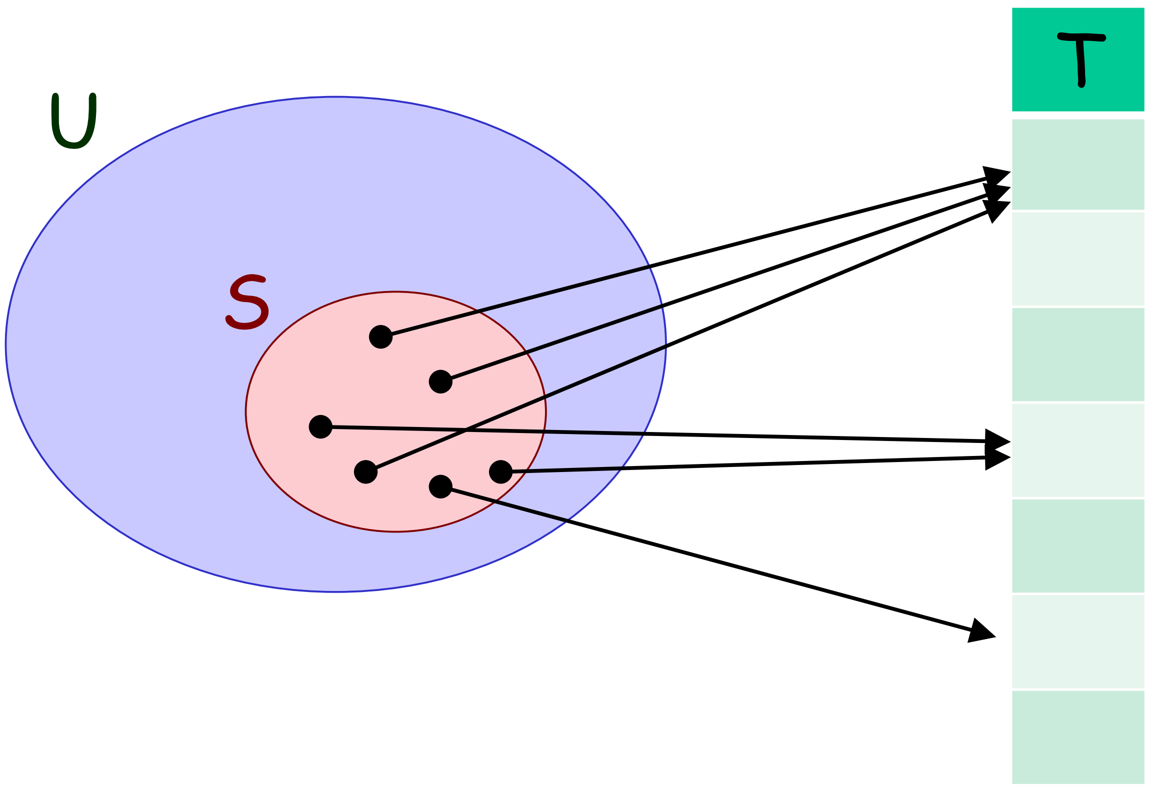
Yes, if we allow operation time to be "average case" instead of "worst case"

Hashing Problem

- To control the space, we use a hash table T of size m (m is often set to $\Theta(n)$)
- Next, we create a hash function h
  - which maps each integer in U to some integer in [1, m]
  - E.g., $h(x) = x^2 + 3x \bmod m$

- Using the hash function, each key will be mapped to some entry in the table

Hash Function



Hashing Problem

- In an ideal case, all keys are mapped to distinct entries in T

  ➔ Search is performed in O(1) time !
- In general, an entry may correspond to more than 1 key ➔ Collision occurs
- Two common ways to handle collision
  - Chaining
  - Open Addressing

Remark

- Hashing has many applications
- E.g., Our web browser (IE/Firefox) will automatically keep the accessed web pages in the hard-disk

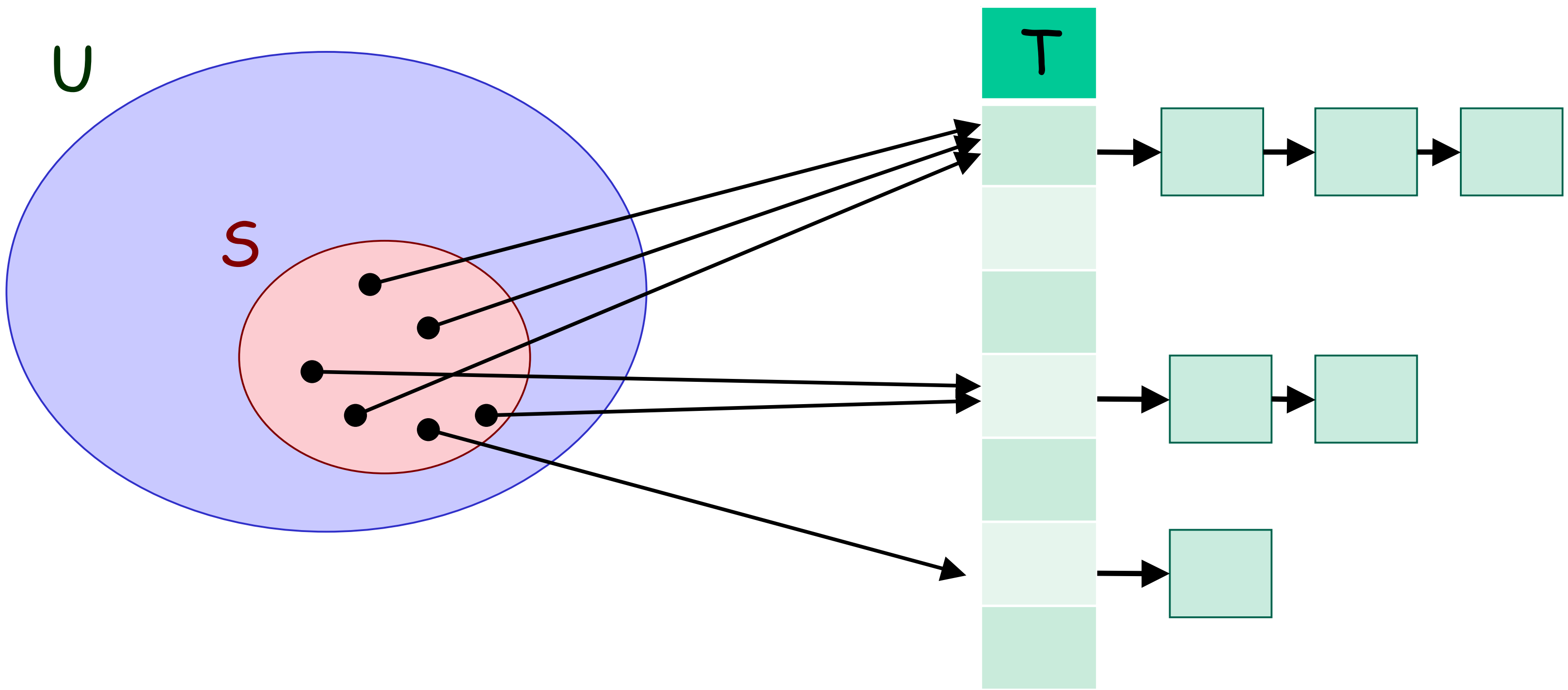
  Then if we try to visit a web page that is accessed before, it becomes faster

How can our browser know if a web page was accessed before ?

Hash with Chaining

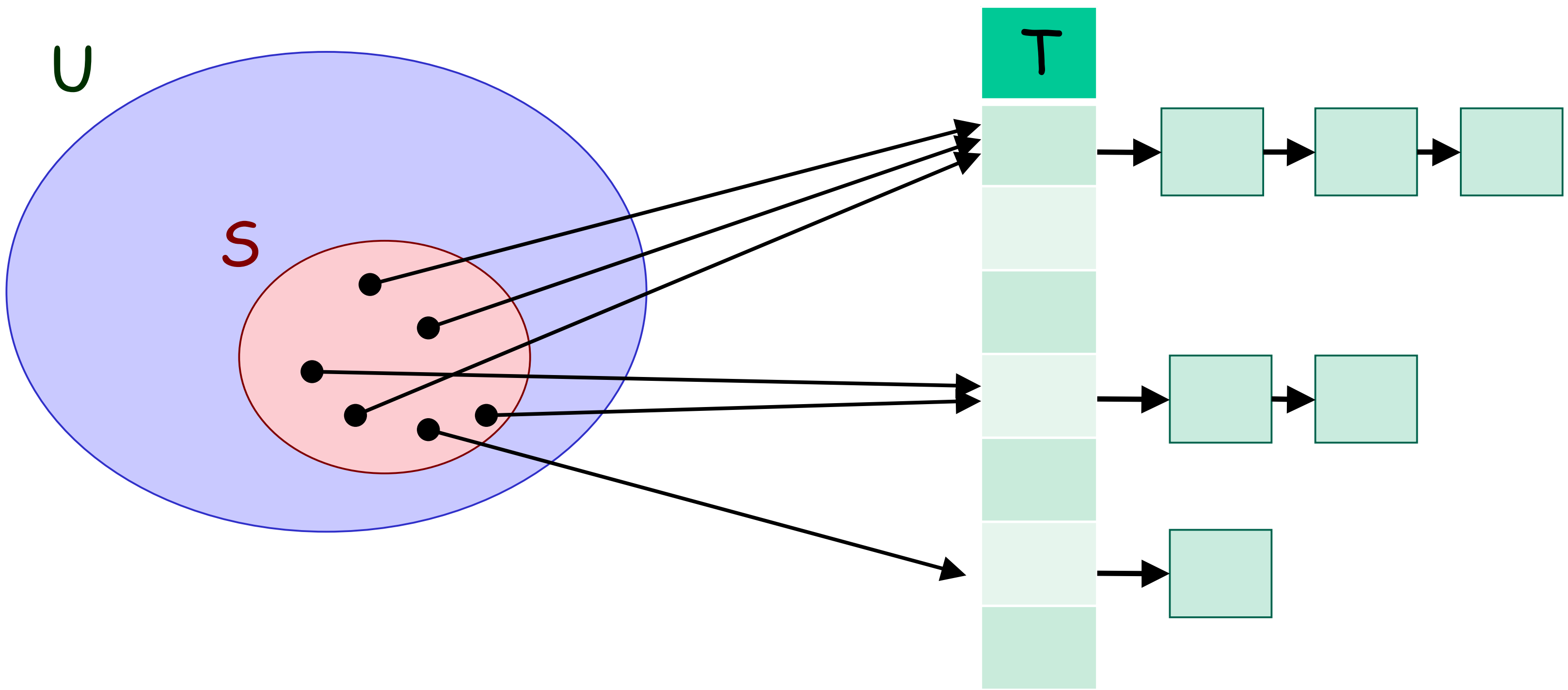
Chaining

- Chaining stores all the keys mapped to the same entry by a linked list



Chaining

- Insertion can be done in O(1) time (why?)
- How about search or delete ?



Performance of Chaining

- Recall that the hash table T has m entries, and there are n keys
- We define load factor $\alpha = n/m$
  - average # keys per entry
- The worst case of search or delete is $O(n)$ time (if all keys are in the same entry)

- How about the average case ?

Performance of Chaining

- To analyze the average case, we use the simple uniform hashing assumption :

1. Each element of U is equally likely to be mapped into any of the m entries
2. Also, it is independent of where any other element is mapped to

- Next, we analyze search and delete

Unsuccessful Search

- Suppose we search for x which is not in S
- Then, we will compute $h(x)$, access the entry $h(x)$ in the table, and traverse all the keys mapped to that entry

  ➔ Search time

  $= \Theta(1) + \Theta(\text{\# of keys traversed})$

- Let n_r be the number of keys in entry r

  ➔ $n = n_1 + n_2 + \dots + n_m$

Unsuccessful Search

Theorem:

The expected time for an unsuccessful search is $\Theta(1+\alpha)$

Proof:

The value $h(x)$ has equal chance to be any number in $[1,m]$ (why?)

➔ Expected search time

$= \Theta(1) + \Theta((\, n_1 + n_2 + \ldots + n_m \,) \,/\, m) = \Theta(1+\alpha)$

Successful Search

- Suppose we search for x which is in S
- Then, we will compute $h(x)$, access the entry $h(x)$, and traverse the keys mapped to that entry <u>as soon as x is found</u>

  ➔ Search time

  $= \Theta(1) + \Theta(\text{\# of keys traversed})$

- Let n_r be the number of keys in entry r

  ➔ $n = n_1 + n_2 + \ldots + n_m$

Successful Search

Theorem: Assuming that each key in S has equal chance to be searched

The expected time for a successful search is $\Theta(1+\alpha)$

- Though it has the same expected time as an unsuccessful search, the analysis is very different
- It is because each entry of the table is not equally likely to be searched

Successful Search

Proof :

We first ignore the $\Theta(1)$ time to compute $h(x)$ and access the entry

Expected Search Time

$$= E[(1/n)(1 + 2 + \ldots + n_1 + 1 + 2 + \ldots + n_2 + \ldots + 1 + 2 + \ldots + n_m)]$$

$$= (m/n)\, E[\, n_1 (n_1 + 1)/2 \,] \quad \text{(by symmetry)}$$

$$= (m/(2n))\, E[\, n_1^2 \,] + (1/2)$$

Successful Search

Proof (cont) :

It remains to compute $E[\, n_1^2 \,]$.

Recall that the value n_1 counts how many of the n keys are mapped to entry 1

➔ This can be expressed as

$$n_1 = Y_1 + Y_2 + \ldots + Y_n$$

where $Y_j = 1$ if key j is in entry 1, and

$Y_j = 0$ otherwise

Successful Search

Proof (cont) :

$\Rightarrow$ $E[\ n_1^2\ ] = E[\ (Y_1 + Y_2 + \ldots + Y_n)^2\ ]$

$$= E[\ Y_1^2 + Y_2^2 + \ldots + Y_n^2 +$$
$$Y_1Y_2 + Y_1Y_3 + \ldots + Y_1Y_n +$$
$$\ldots +$$
$$Y_nY_1 + Y_nY_2 + \ldots + Y_nY_{n-1}\ ]$$
$$= n\,E[\ Y_1^2\ ] + n(n-1)\,E[\ Y_1Y_2\ ]$$
$$= n/m + n(n-1)/m^2$$

Successful Search

Proof (cont) :

Combining everything, and adding back the $\Theta(1)$ time to compute h(x) and access entry, we have :

Expected Search Time

$= \Theta(1) + (m/(2n))\, E[\, n_1^2\,] + (1/2)$

$= \Theta(1) + (m/(2n))\, (n/m + n(n-1)/m^2) + (1/2)$

$= \Theta(1) + 1 + (n-1)/(2m) = \Theta(1+\alpha)$

Remark 1

- In both cases, search time is $\Theta(1+\alpha)$
- Deletion is done by search and delete

  ➔ expected time is $\Theta(1+\alpha)$
- If m is set to $\Theta(n)$
  - Space of hash table $T = \Theta(n)$
  - Expected time for each operation = $\Theta(1)$

Remark 2

- Our analysis for successful search time is different from that in the textbook
  - Though the value obtained is exactly the same
  - See the textbook for a reference

- In fact, we can use the same analysis technique to obtain the average running time for bucket sort (See Notes 5)

Hash with Open Addressing

Open Addressing

- In open addressing, each entry of the hash table contains to at most 1 key
  ➔ load factor is at most 1
- When inserting a key k, we use k to compute a sequence of entries to check, until we get an empty entry to store k
- The hash function h now contains two parameters : (1) the key, and (2) the sequence number

Open Addressing

- The insertion procedure is as follows :

```
1.  j = 0 ;
2.  while entry h(k, j) is not empty
          increase j by 1 ;
3.  Insert key k at the entry h(k, j)
```

- We often require h(k, 0), h(k, 1), … to be a permutation of 1, 2, …, m

  ➔ Allows all entries of T to be used

Open Addressing

- We assume that no delete is allowed
- In that case, search can be done in the same way as we insert
  - To search for x, we repeatedly try the entries $h(k, j)$, for $j = 0, 1, 2, \ldots$
  - We stop when we have found x or when we hit an empty entry

- What is the average insert/search time?

A Useful Formula

Lemma: Let X be a random variable that takes on non-negative integral values. Then,

$$E[X] = \sum_{i=1,2,\ldots} \Pr(X \geq i)$$

Proof:

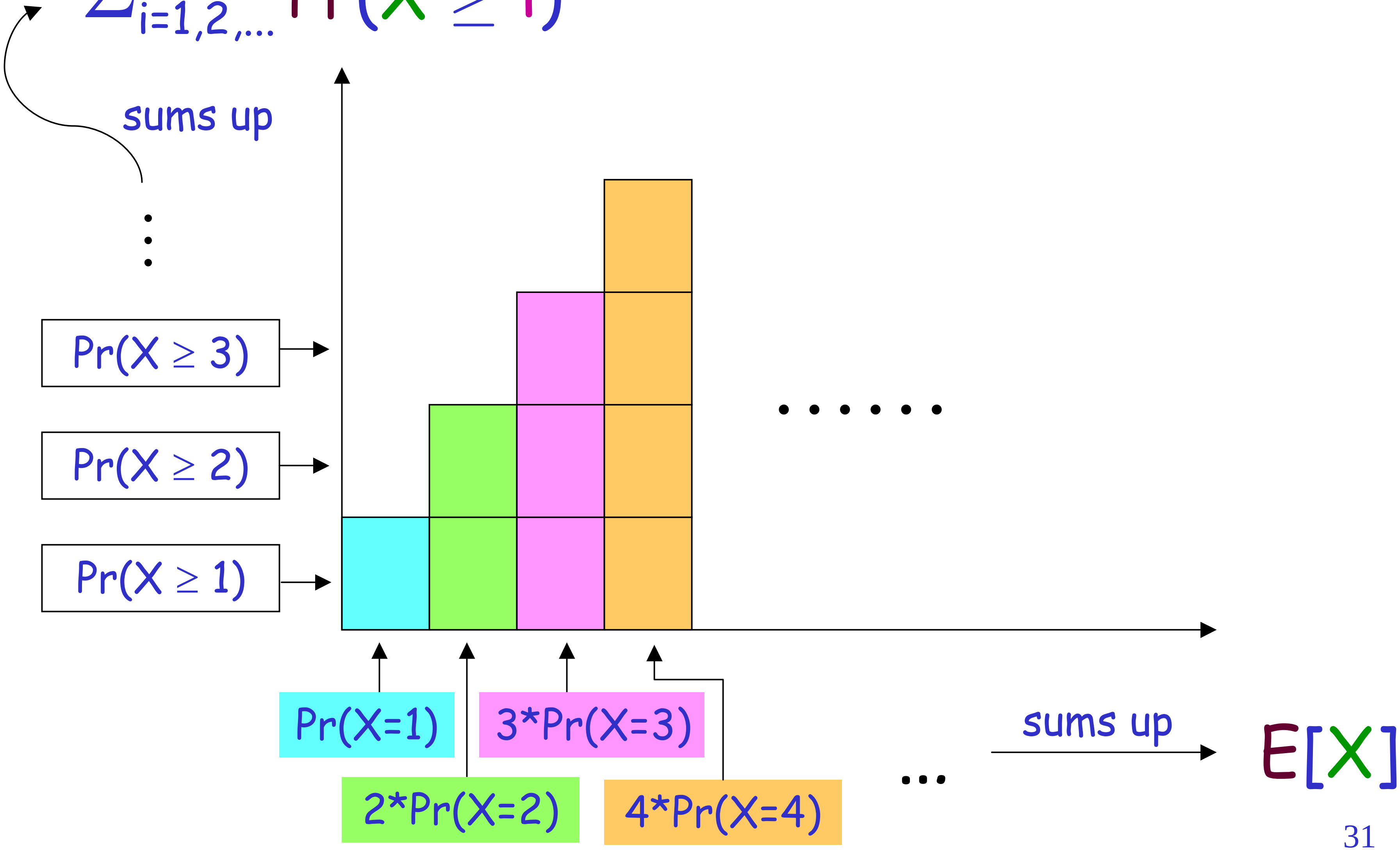
$$\sum_{i=1,2,\ldots} \Pr(X \geq i) \quad = \sum_{i=1,2,\ldots} \sum_{j=i,i+1,\ldots} \Pr(X = j)$$

$$= \sum_{j=1,2,\ldots} \sum_{i=1,2,\ldots,j} \Pr(X = j)$$

$$= \sum_{j=1,2,\ldots} j \Pr(X = j) = E[X]$$

A Useful Formula (2^{nd} proof)

$$\sum_{i=1,2,\ldots} \Pr(X \geq i)$$



Performance of Open Addressing

- To analyze the average case, we use the **uniform hashing** assumption :

1. The function $h(k, j)$ produces a random permutation of 1, 2, …, m
2. Also, each permutation is equally likely to be produced

- Consequently, $h(k,0)$ has $1/m$ chance to be in any entry. Then $h(k,1)$ has $1/(m-1)$ chance to be in any other entry apart from $h(k,0)$, and so on …

Unsuccessful Search

Theorem:

The expected time for an unsuccessful search is $O(1/(1-\alpha))$, where $\alpha = n/m$

Proof: Let X = # entries examined

$Pr(X \geq 1) = 1$, $Pr(X \geq 2) = n/m = \alpha$

$Pr(X \geq 3) = n/m \times (n-1)/(m-1) \leq \alpha^2$

$Pr(X \geq i) = n/m \times \ldots \times (n-i+2)/(m-i+2) \leq \alpha^{i-1}$

➔ $E[X] = \Sigma\, Pr(X \geq i) \leq 1 + \alpha + \alpha^2 + \ldots = 1/(1-\alpha)$

Insertion

Theorem:

Assume we never insert a key twice in S.

The expected time for an insertion is $O(1/(1-\alpha))$, where $\alpha = n/m$

Proof:

Insertion requires an unsuccessful search followed by placing the key to the first empty entry

➔ Same time as unsuccessful search

Successful Search

Theorem:

Assuming that each key in S has equal chance to be searched

The expected time for a successful search is $O\left((1/\alpha) \log \{ 1/(1-\alpha) \} \right)$

Proof:

Expected time to search the $(j+1)^{th}$ inserted key $= 1/(1-j/m) = m/(m-j)$ (why?)

Successful Search

Proof (cont) :

Expected Search Time

$= 1/n \times (m/m + m/(m-1) + \ldots + m/(m-n+1))$

$= m/n \times (1/m + 1/(m-1) + \ldots + 1/(m-n+1))$

$= m/n \times O(\log m - \log (m-n))$ [harmonic sum]

$= m/n \times O(\log \{ 1/(1 - n/m) \})$

$= 1/\alpha \times O(\log \{ 1/(1-\alpha) \})$