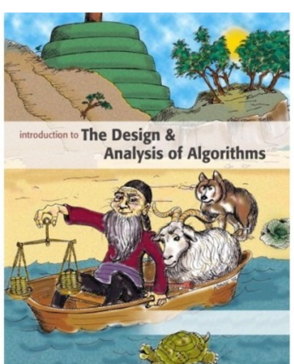
Introduction to

Algorithm Design and Analysis

[9] Hashing



Yu Huang

http://cs.nju.edu.cn/yuhuang
Institute of Computer Software
Nanjing University

In the last class…

- **The searching problem**
  - o Organization of data
- **Binary search**
  - o *logn* search over sorted data
- **Red-black tree**
  - o *logn* search over BST
    - Definition
    - Insertion, deletion

Hashing

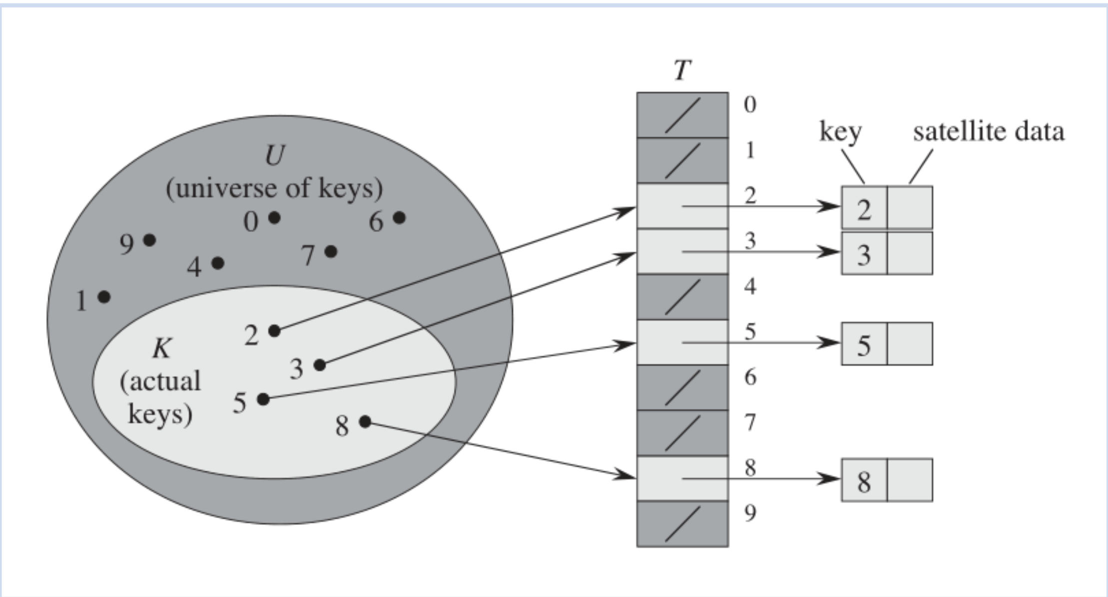
- **The searching problem**
  - The ambition of hashing
- **Hashing**
  - Direct-address table
  - Basic idea of hashing
- **Collision Handling for Hashing**
  - Closed Address Hashing
  - Open Address Hashing
- **Array Doubling and Amortized Analysis**

The Searching Problem

- **Searching vs. Selection**
  - o *Search* for “Alice” or ”Bob”
    - The key itself matters
  - o *Select* the “rank 2” student
    - The partial order relation matters
- **The ambition of hashing**
  - o Brute force case: O(n)
  - o Ideal case: O(1)
  - o Hashing: $O(1+\alpha)$

Searching - a Brute Force Approach

- **Direct-address table**
  - Take into account the *whole universe* of keys



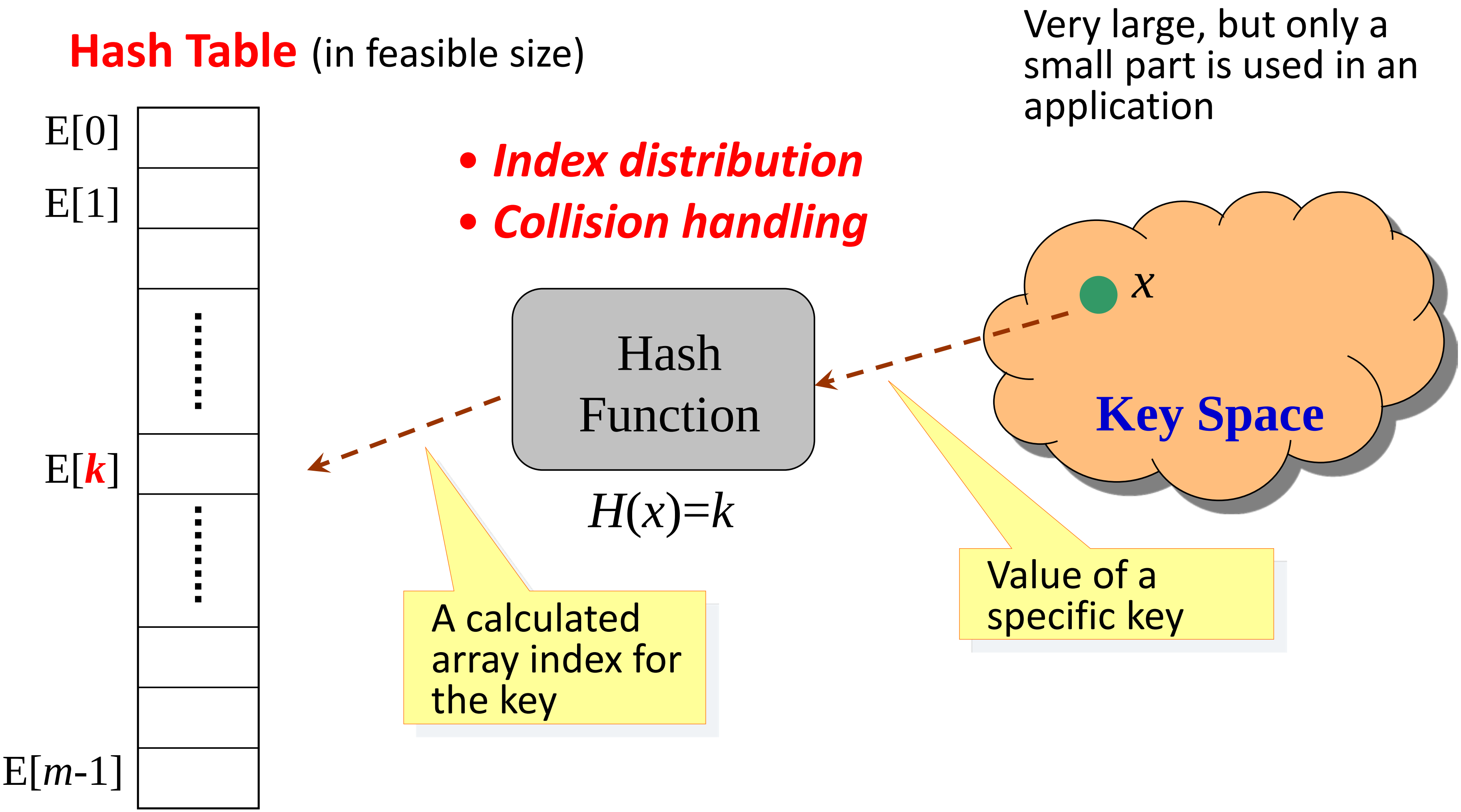
Direct-address Table

DIRECT-ADDRESS-SEARCH (T, k)
  return T[k]
DIRECT-ADDRESS-INSERT (T, x)
  T[key[x]] := x
DIRECT-ADDRESS-DELETE (T, x)
  T[key[x]] := NIL

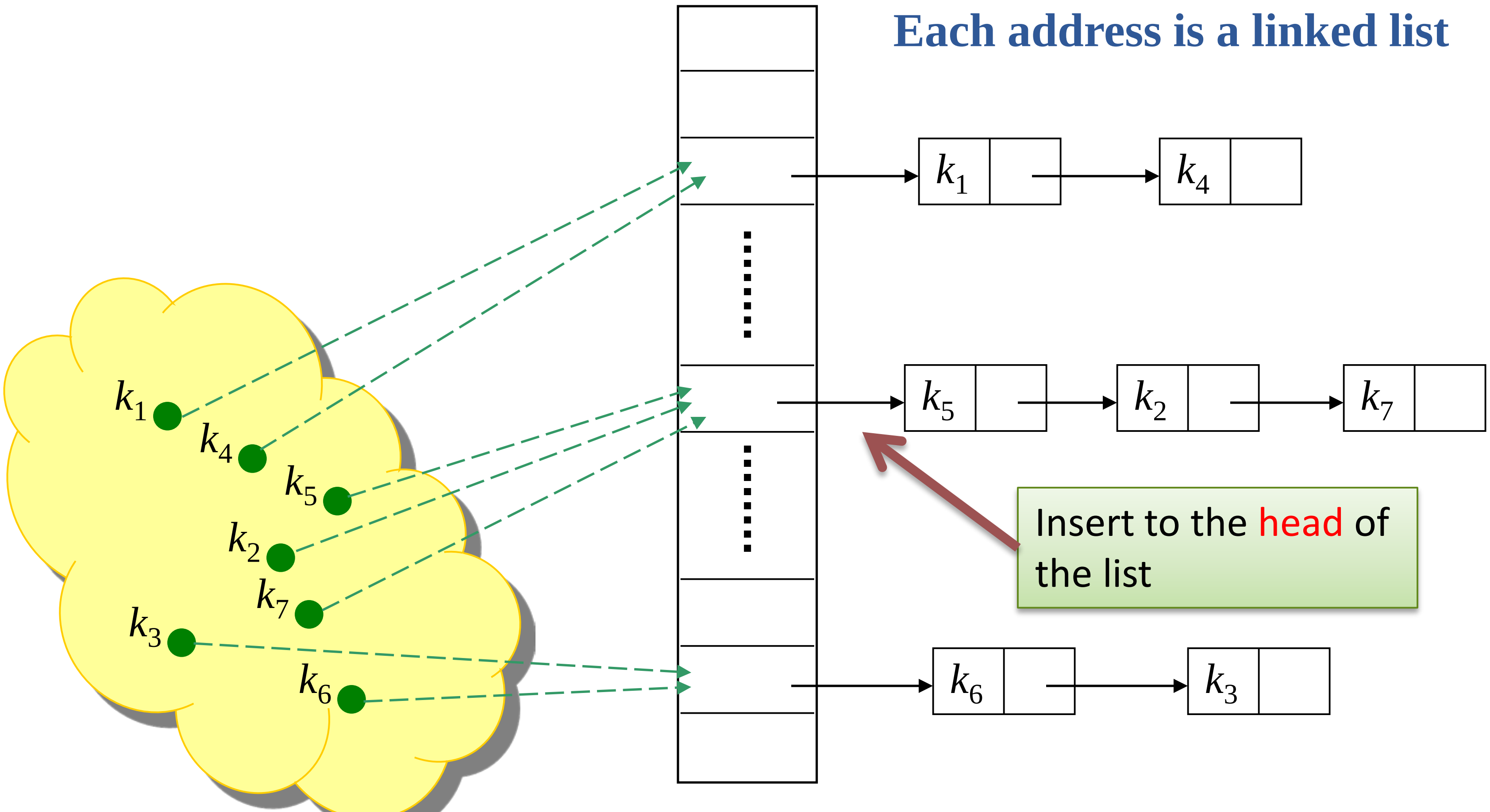
Hashing: the Idea

Hash Table (in feasible size)

- ***Index distribution***
- ***Collision handling***



Collision Handling: Closed Address



Closed Address - Analysis

- **Assumption - simple uniform hashing**
  - For j=0,1,2,…,m-1, the average length of the list at $E[j]$ is n/m.
- **The average cost of an unsuccessful search:**
  - Any key that is not in the table is equally likely to hash to any of the m address.
  - Total cost $\Theta(1+ n/m)$
    - The average cost to determine that the key is not in the list $E[h(k)]$ is the cost to search to the end of the list, which is n/m.

Closed Address - Analysis

- **For successful search** **(assuming that x_i is the i^{th} element inserted into the table, i=1,2,…,n)**
  - For each i, the probability of that x_i is searched is $1/n$.
  - For a specific x_i, the number of elements examined in a successful search is $t+1$, where t is the number of elements inserted into the same list as x_i, after x_i has been inserted

$$\frac{1}{n}\sum_{i=1}^{n}(1+t)$$

- **How to compute t?**
  - Consider the *construction* process of the hash table

Closed Address - Analysis

- **For successful search:** **(assuming that x_i is the i^{th} element inserted into the table, i=1,2,…,n)**
  - For each i, the probability of that x_i is searched is $1/n$.
  - For a specific x_i, the number of elements examined in a successful search is t+1, where t is the number of elements inserted into the same list as x_i, after x_i has been inserted. And for any j, the probability of that x_j is inserted into the same list of x_i is $1/m$. So, the cost is:

Cost for computing hashing

$$1 + \frac{1}{n}\sum_{i=1}^{n}\left(1 + \sum_{j=i+1}^{n}\frac{1}{m}\right)$$

Expected number of elements in front of the searched one in the same linked list.

Closed Address: Analysis

- **The average cost of a successful search:**
  - Define α=n/m as ***load factor***,

$$\text{The average cost of a successful search is:}$$

$$\frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n}\frac{1}{m}\right)=1+\frac{1}{nm}\sum_{i=1}^{n}(n-i)=1+\frac{1}{nm}\sum_{i=1}^{n-1}i$$

$$=1+\frac{n-1}{2m}=1+\frac{\alpha}{2}-\frac{\alpha}{2n}=\Theta(1+\alpha)$$

Number of elements in front of the searched one in the same linked list

Collision Handling: Open Address

- **All elements are stored in the hash table**
  - o No linked list is used
  - o The load factor α can not be larger than 1.
- **Collision is settled by "rehashing"**
  - o A function is used to get a new hashing address for each collided address
    - The hash table slots are *probed* successively, until a valid location is found.
- **The probe sequence can be seen as a permutation of (0,1,2,..., *m*-1)**

Commonly Used Probing

Linear probing:

Given an ordinary hash function h', which is called an auxiliary hash function, the hash function is: **(clustering may occur)**

$$h(k,i) = (h'(k)+i) \bmod m \quad (i=0,1,\ldots,m-1)$$

Quadratic Probing:

Given auxiliary function h' and nonzero auxiliary constant c_1 and c_2, the hash function is: **(secondary clustering may occur)**

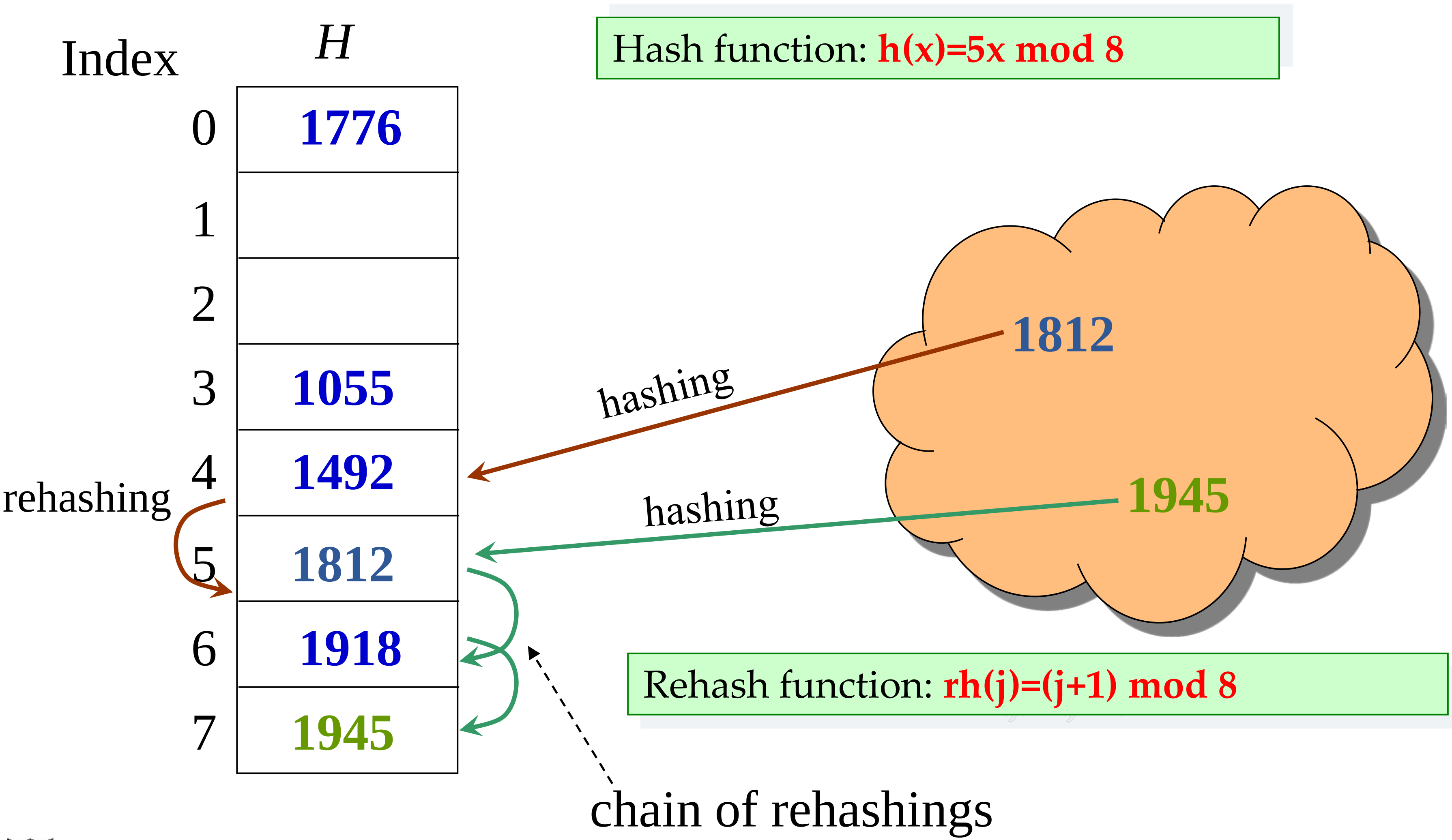
$$h(k,i) = (h'(k)+c_1 i+ c_2 i^2) \bmod m \quad (i=0,1,\ldots,m-1)$$

Double hashing:

Given auxiliary functions h_1 and h_2, the hash function is:

$$h(k,i) = (h_1(k)+ ih_2(k)) \bmod m \quad (i=0,1,\ldots,m-1)$$

Linear Probing: An Example



Equally Likely Permutations

- **Assumption**
  - Each key is equally likely to have any of the m! permutations of (1,2...,m-1) as its probe sequence.
- **Note**
  - Both linear and quadratic probing have only m distinct probe sequence, as determined by the first probe.

Analysis for Open Address Hashing

- **The average number of probes in an unsuccessful search is at most 1/(1-α) (α=*n*/*m*<1)**
  - Assuming uniform hashing

The probability of the first probed position being occupied is $\frac{n}{m}$, and that of the $j^{th}(j>1)$ position occupied is $\frac{n-j+1}{m-j+1}$. So the probability of the number of probes no less than i will be:

$$\frac{n}{m}\cdot\frac{n-1}{m-1}\cdot\frac{n-2}{m-2}\cdots\cdot\frac{n-i+2}{m-i+2}\leq(\frac{n}{m})^{i-1}=\alpha^{i-1}$$

The the average number of probe is: $\sum_{i=1}^{\infty}\alpha^{i-1}=\sum_{i=0}^{\infty}\alpha^{i}=\frac{1}{1-\alpha}$

Analysis for Open Address Hashing

- **The average cost of probes in an successful search is at most** $\frac{1}{\alpha}\ln\frac{1}{1-\alpha}$ **($\alpha = n/m < 1$)**
  - Assuming uniform hashing

To search for the $(i+1)^{th}$ inserted element in the table, the cost is the same as that for inserting it when there are just i elements in the table. At that time, $\alpha = \frac{i}{m}$. So the cost is $\frac{1}{1-\frac{i}{m}} = \frac{m}{m-i}$.
So the average cost for a successful search is:

$$\frac{1}{n}\sum_{i=0}^{n-1}\frac{m}{m-i} = \frac{m}{n}\sum_{i=0}^{n-1}\frac{1}{m-i} = \frac{1}{\alpha}\sum_{i=m-n+1}^{m}\frac{1}{i}$$

$$\leq \frac{1}{\alpha}\int_{m-n}^{m}\frac{dx}{x} = \frac{1}{\alpha}\ln\frac{m}{m-n} = \frac{1}{\alpha}\ln\frac{1}{1-\alpha}$$

For your reference:
Half full: 1.387;
90% full: 2.559

Hash Function

- **A good hash function satisfies the assumption of simple uniform hashing.**
  - o Heuristic hashing functions
    - The division method: $h(k)=k \bmod m$
    - The multiplication method: $h(k)=\lfloor m(kA \bmod 1)\rfloor$ $(0<A<1)$
  - o No single function can avoid the worst case $\Theta(n)$.
    - So, "Universal hashing" is proposed.
  - o Rich resource about hashing function
    - Gonnet and Baeza-Yates: Handbook of Algorithms and Data Structures, Addison-Wesley, 1991.

Array Doubling

- **Cost for search in a hash table is $\Theta(1+\alpha)$**
  - If we can keep α constant, the cost will be $\Theta(1)$
- **What if the hash table is more and more loaded?**
  - Space allocation techniques such as array doubling may be needed.
- **The problem of "unusually expensive" individual operation.**

Looking at the Memory Allocation

- hashingInsert(HASHTABLE *H*, ITEM *x*)
- integer *size*=0, *num*=0;
- if *size*=0 then **allocate a block of size 1**; *size*=1;
- if *num*=*size* then
- allocate a block of size 2*size*;
- move all item into new table;
- *size*=2*size*;
- insert *x* into the table;
- *num*=*num*+1;
- return

Insertion with expansion: cost *size*

Elementary insertion: cost 1

Worst-case Analysis

- **For n execution of insertion operations**
  - A bad analysis: the worst case for one insertion is the case when expansion is required, up to n
  - So, the worst case cost is in $O(n^2)$.

Of course NOT!

- **Note the expansion is required during the ith operation only if $i=2^k$, and the cost of the ith operation**

$$c_i = \begin{cases} i & \text{if } i-1 \text{ is exactly the power of 2} \\ 1 & otherwise \end{cases}$$

So the total cost is: $\sum_{i=1}^{n} c_i \leq n + \sum_{j=0}^{\lfloor \log n \rfloor} 2^j < n + 2n = 3n$

Amortized Analysis – Why?

- **Unusually expensive operations**
  - E.g., Insert-with-array-doubling
- **Relation between expensive and usual operations**
  - Each piece of the doubling cost corresponds to some previous insert

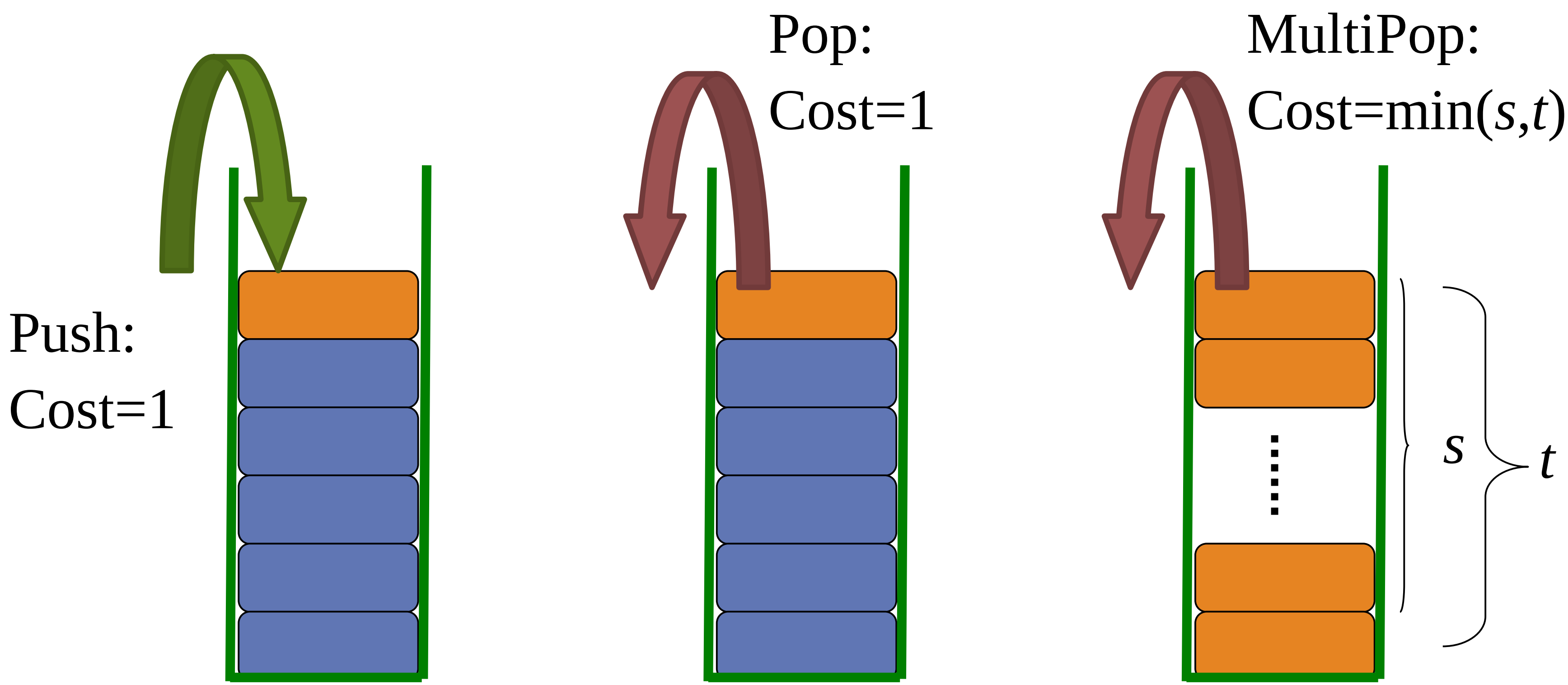
Amortized Analysis

- **Amortized equation:**

  amortized cost = actual cost + accounting cost

- **Design goals for accounting cost**
  - In any legal sequence of operations, the sum of the accounting costs is nonnegative.
  - The amortized cost of each operation is fairly regular, in spite of the wide fluctuate possible for the actual cost of individual operations.

Amortized Analysis: MultiPop Stack



Amortized cost: push:2; pop, multipop: 0

Amortized Analysis: Binary Counter

0	0 0 0 0 0 0 0 0	0
1	0 0 0 0 0 0 0 1	1
2	0 0 0 0 0 0 1 0	3
3	0 0 0 0 0 0 1 1	4
4	0 0 0 0 0 1 0 0	7
5	0 0 0 0 0 1 0 1	8
6	0 0 0 0 0 1 1 0	10
7	0 0 0 0 0 1 1 1	11
8	0 0 0 0 1 0 0 0	15
9	0 0 0 0 1 0 0 1	16
10	0 0 0 0 1 0 1 0	18
11	0 0 0 0 1 0 1 1	19
12	0 0 0 0 1 1 0 0	22
13	0 0 0 0 1 1 0 1	23
14	0 0 0 0 1 1 1 0	25
15	0 0 0 0 1 1 1 1	26
16	0 0 0 1 0 0 0 0	31

Cost measure: bit flip

amortized cost:

set 1: 2

set 0: 0

Accounting Scheme for Stack Push

- **Push operation with array doubling**
  - No resize triggered: 1
  - Resize($n \rightarrow 2n$) triggered: $nt+1$ (t is a constant)
- **Accounting scheme (specifying accounting cost)**
  - No resize triggered: $2t$
  - Resize($n \rightarrow 2n$) triggered: $-nt+2t$
- **So, the amortized cost of each individual push operation is $1+2t \in \Theta(1)$**

Thank you!

Q & A

Yu Huang

http://cs.nju.edu.cn/yuhuang