HASH TABLES

Gustavo Carvalho
(ghpc@cin.ufpe.br)

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil

Centro de Informática UFPE

Universidade Federal de Pernambuco

Agenda

Dictionary ADT

A set of elements with the options of searching, insertion and deletion

- Elements are indexed by comparable keys

Operations:

- void clear(Dictionary d);
- void insert(Dictionary d, Key k, E e); // reflect on: multiple entries
- E remove(Dictionary d, Key k); // reflect on: multiple entries
- E removeAny(Dictionary d); // alternative: getKeys
- E find(Dictionary d, Key k); // reflect on: multiple entries
- int size(Dictionary d);

Implementations: array, linked list, hash tables, and balanced trees

Agenda

Hashing

Space and time trade-off

- Worse space efficiency to get a better time efficiency

Hashing: distributing keys among a one-dimensional array dicionários

- Very efficient way[1] to implement dictionaries
- Hash table: $H[0..m-1]$
- Hash function: $h : Key \rightarrow 0..m-1$

h needs to satisfy somewhat conflicting requirements

- Adequate relation between m and $|Key|$
- Distribute keys as evenly as possible
- Has to be easy to compute

[1] In terms of temporal efficiency

Hashing

Limitations:

- Not ideal in the presence of multiple entries per key
- Not ideal for accessing elements based on some order
- Not ideal for searching based on a range of keys

Hash functions for integers

A trivial attempt[2]: $h(K) = K \bmod m$

- Let $m = 100$ (hash addresses within 0..99)
- Let $K = 4567$, then $h(K) = 4567 \bmod 100 = 67$

A better approach: the mid-square method

- Compute K^2, and select the r middle digits, such that $10^r - 1 < m$
- Let $m = 100$ (hash addresses within 0..99), then $r = 2$
- Let $K = 4567$, $K^2 = 20857489$, then $h(K) = 57$

[2] **Attention** to the implementation of mod : $\mathbb{Z} \to \mathbb{N}$

Hash functions for strings

A first attempt: fold

Algorithm: int h(string K)

```
1  s ← length(K);
2  sum ← 0;
3  for i ← 0 to s − 1 do
4  |    sum ← sum + K[i];
5  return abs(sum)%m; // abs = overflow and %
```

Fair/bad distribution depending on m and K

- Suppose that $length(K) = 10$ (in average)
- Let K have only capital letters
- Since $A = 65$ and $Z = 90$, $sum \in [650..900]$
- If $m \leq 100$, it is a fair distribution
- If $m \geq 1000$, it is a bad distribution

Hash functions for strings

A better approach: sfold:

Algorithm: int h(string K)

```
1  intLength ← length(K)/4;
2  sum ← 0;
3  for i ← 0 to intLength − 1 do
4  |    sub ← substring(K, i ∗ 4, (i ∗ 4) + 4);   // initial and final positions
5  |    mult ← 1;
6  |    for j ← 0 to 3 do
7  |    |    sum ← sum + sub[j] ∗ mult;
8  |    |    mult ← mult ∗ 256;
9  sub ← substring(K, intLength ∗ 4);            // initial position to the end
10 mult ← 1;
11 s ← length(sub);
12 for j ← 0 to s − 1 do
13 |    sum ← sum + sub[j] ∗ mult;
14 |    mult ← mult ∗ 256;
15 return abs(sum)%m; // abs = overflow and %
```

Centro de Informática UFPE

Universidade Federal de Pernambuco

Hash functions for strings

Algorithm: int h(string K)

```
1  intLength ← length(K)/4;
2  sum ← 0;
3  for i ← 0 to intLength − 1 do
4      sub ← substring(K, i ∗ 4, (i ∗ 4) + 4);
5      mult ← 1;
6      for j ← 0 to 3 do
7          sum ← sum + sub[j] ∗ mult;
8          mult ← mult ∗ 256;
9  sub ← substring(K, intLength ∗ 4);
10 mult ← 1;
11 s ← length(sub);
12 for j ← 0 to s − 1 do
13     sum ← sum + sub[j] ∗ mult;
14     mult ← mult ∗ 256;
15 return abs(sum)%m;
```

Let $K =$ "ALGORITHMS"
Let $m = 1000$

$fold(K) = 762$

$sfold(K) = ?$

Centro de Informática UFPE

Universidade Federal de Pernambuco

Hash functions for strings

Algorithm: int h(string K)

```
1  intLength ← length(K)/4;
2  sum ← 0;
3  for i ← 0 to intLength − 1 do
4  |    sub ← substring(K, i ∗ 4, (i ∗ 4) + 4);
5  |    mult ← 1;
6  |    for j ← 0 to 3 do
7  |    |    sum ← sum + sub[j] ∗ mult;
8  |    |    mult ← mult ∗ 256;
9  sub ← substring(K, intLength ∗ 4);
10 mult ← 1;
11 s ← length(sub);
12 for j ← 0 to s − 1 do
13 |    sum ← sum + sub[j] ∗ mult;
14 |    mult ← mult ∗ 256;
15 return abs(sum)%m;
```

Let $K =$ "ALGORITHMS"
Let $m = 1000$

$fold(K) = 762$

$sfold(K) = ?$

$intLength = 2$
$sum = 0$

Hash functions for strings

Algorithm: int h(string K)

```
1  intLength ← length(K)/4;
2  sum ← 0;
3  for i ← 0 to intLength − 1 do
4      sub ← substring(K, i ∗ 4, (i ∗ 4) + 4);
5      mult ← 1;
6      for j ← 0 to 3 do
7          sum ← sum + sub[j] ∗ mult;
8          mult ← mult ∗ 256;
9  sub ← substring(K, intLength ∗ 4);
10 mult ← 1;
11 s ← length(sub);
12 for j ← 0 to s − 1 do
13     sum ← sum + sub[j] ∗ mult;
14     mult ← mult ∗ 256;
15 return abs(sum)%m;
```

Let $K =$ "ALGORITHMS"
Let $m = 1000$

$fold(K) = 762$

$sfold(K) = ?$

$intLength = 2$
$i = 0$
$sub =$ "ALGO"
$sum = 1330072641$
$mult = 0$

UNIVERSIDADE FEDERAL DE PERNAMBUCO

Hash functions for strings

Algorithm: int h(string K)

```
1  intLength ← length(K)/4;
2  sum ← 0;
3  for i ← 0 to intLength − 1 do
4      sub ← substring(K, i ∗ 4, (i ∗ 4) + 4);
5      mult ← 1;
6      for j ← 0 to 3 do
7          sum ← sum + sub[j] ∗ mult;
8          mult ← mult ∗ 256;
9  sub ← substring(K, intLength ∗ 4);
10 mult ← 1;
11 s ← length(sub);
12 for j ← 0 to s − 1 do
13     sum ← sum + sub[j] ∗ mult;
14     mult ← mult ∗ 256;
15 return abs(sum)%m;
```

Let $K =$ "ALGORITHMS"
Let $m = 1000$

$fold(K) = 762$

$sfold(K) = ?$

$intLength = 2$
$i = 1$
$sub =$ "RITH"
$sum = -1751411309$
$mult = 0$

UNIVERSIDADE FEDERAL DE PERNAMBUCO

Hash functions for strings

Algorithm: int h(string K)

```
1  intLength ← length(K)/4;
2  sum ← 0;
3  for i ← 0 to intLength − 1 do
4      sub ← substring(K, i ∗ 4, (i ∗ 4) + 4);
5      mult ← 1;
6      for j ← 0 to 3 do
7          sum ← sum + sub[j] ∗ mult;
8          mult ← mult ∗ 256;
9  sub ← substring(K, intLength ∗ 4);
10 mult ← 1;
11 s ← length(sub);
12 for j ← 0 to s − 1 do
13     sum ← sum + sub[j] ∗ mult;
14     mult ← mult ∗ 256;
15 return abs(sum)%m;
```

Let $K =$ "ALGORITHMS"
Let $m = 1000$

$fold(K) = 762$

$sfold(K) = ?$

$intLength = 2$
$i = 2$
$sub =$ "RITH"
$sum = -1751411309$
$mult = 0$

Centro de Informática UFPE

Universidade Federal de Pernambuco

Hash functions for strings

Algorithm: int h(string K)

```
1  intLength ← length(K)/4;
2  sum ← 0;
3  for i ← 0 to intLength − 1 do
4  |    sub ← substring(K, i ∗ 4, (i ∗ 4) + 4);
5  |    mult ← 1;
6  |    for j ← 0 to 3 do
7  |    |    sum ← sum + sub[j] ∗ mult;
8  |    |    mult ← mult ∗ 256;
9  sub ← substring(K, intLength ∗ 4);
10 mult ← 1;
11 s ← length(sub);
12 for j ← 0 to s − 1 do
13 |    sum ← sum + sub[j] ∗ mult;
14 |    mult ← mult ∗ 256;
15 return abs(sum)%m;
```

Let $K =$ "ALGORITHMS"
Let $m = 1000$

$fold(K) = 762$

$sfold(K) = ?$

$intLength = 2$
$sub =$ "MS"
$s = 2$
$sum = -1751411309$
$mult = 1$

Centro de Informática UFPE

Universidade Federal de Pernambuco

Hash functions for strings

Algorithm: int h(string K)

```
1  intLength ← length(K)/4;
2  sum ← 0;
3  for i ← 0 to intLength − 1 do
4      sub ← substring(K, i ∗ 4, (i ∗ 4) + 4);
5      mult ← 1;
6      for j ← 0 to 3 do
7          sum ← sum + sub[j] ∗ mult;
8          mult ← mult ∗ 256;
9  sub ← substring(K, intLength ∗ 4);
10 mult ← 1;
11 s ← length(sub);
12 for j ← 0 to s − 1 do
13     sum ← sum + sub[j] ∗ mult;
14     mult ← mult ∗ 256;
15 return abs(sum)%m;
```

Let $K =$ "ALGORITHMS"
Let $m = 1000$

$fold(K) = 762$

$sfold(K) = ?$

$intLength = 2$
$sub =$ "MS"
$s = 2$
$j = 0$
$sum = -1751411232$
$mult = 256$

Centro de Informática UFPE

Universidade Federal de Pernambuco

Hash functions for strings

Algorithm: int h(string K)

```
1  intLength ← length(K)/4;
2  sum ← 0;
3  for i ← 0 to intLength − 1 do
4      sub ← substring(K, i ∗ 4, (i ∗ 4) + 4);
5      mult ← 1;
6      for j ← 0 to 3 do
7          sum ← sum + sub[j] ∗ mult;
8          mult ← mult ∗ 256;
9  sub ← substring(K, intLength ∗ 4);
10 mult ← 1;
11 s ← length(sub);
12 for j ← 0 to s − 1 do
13     sum ← sum + sub[j] ∗ mult;
14     mult ← mult ∗ 256;
15 return abs(sum)%m;
```

Let $K =$ "ALGORITHMS"
Let $m = 1000$

$fold(K) = 762$

$sfold(K) = ?$

$intLength = 2$
$sub =$ "MS"
$s = 2$
$j = 1$
$sum = -1751389984$
$mult = 65536$

Centro de Informática UFPE

Universidade Federal de Pernambuco

Hash functions for strings

Algorithm: int h(string K)

```
1  intLength ← length(K)/4;
2  sum ← 0;
3  for i ← 0 to intLength − 1 do
4      sub ← substring(K, i ∗ 4, (i ∗ 4) + 4);
5      mult ← 1;
6      for j ← 0 to 3 do
7          sum ← sum + sub[j] ∗ mult;
8          mult ← mult ∗ 256;
9  sub ← substring(K, intLength ∗ 4);
10 mult ← 1;
11 s ← length(sub);
12 for j ← 0 to s − 1 do
13     sum ← sum + sub[j] ∗ mult;
14     mult ← mult ∗ 256;
15 return abs(sum)%m;
```

Let $K =$ "ALGORITHMS"
Let $m = 1000$

$fold(K) = 762$

$sfold(K) = ?$

$intLength = 2$
$sub =$ "MS"
$s = 2$
$j = 2$
$sum = -1751389984$
$mult = 65536$

Centro de Informática UFPE

Universidade Federal de Pernambuco

Hash functions for strings

Algorithm: int h(string K)

```
1  intLength ← length(K)/4;
2  sum ← 0;
3  for i ← 0 to intLength − 1 do
4  |    sub ← substring(K, i ∗ 4, (i ∗ 4) + 4);
5  |    mult ← 1;
6  |    for j ← 0 to 3 do
7  |    |    sum ← sum + sub[j] ∗ mult;
8  |    |    mult ← mult ∗ 256;
9  sub ← substring(K, intLength ∗ 4);
10 mult ← 1;
11 s ← length(sub);
12 for j ← 0 to s − 1 do
13 |    sum ← sum + sub[j] ∗ mult;
14 |    mult ← mult ∗ 256;
15 return abs(sum)%m;
```

Let $K =$ "ALGORITHMS"
Let $m = 1000$

$fold(K) = 762$

$sfold(K) = 984$

Hashing: collisions

Collisions

- If $m < |\ Key\ |$, they will occur
- If $m \geq |\ Key\ |$, they still might occur

Collision resolution mechanisms

- Open hashing (separate chaining)
- Closed hashing (open addressing)

Perfect hashing

- No collisions at all
- *Key* is known and available beforehand
- Example: data access on a read-only CD

Centro de Informática UFPE

Universidade Federal de Pernambuco

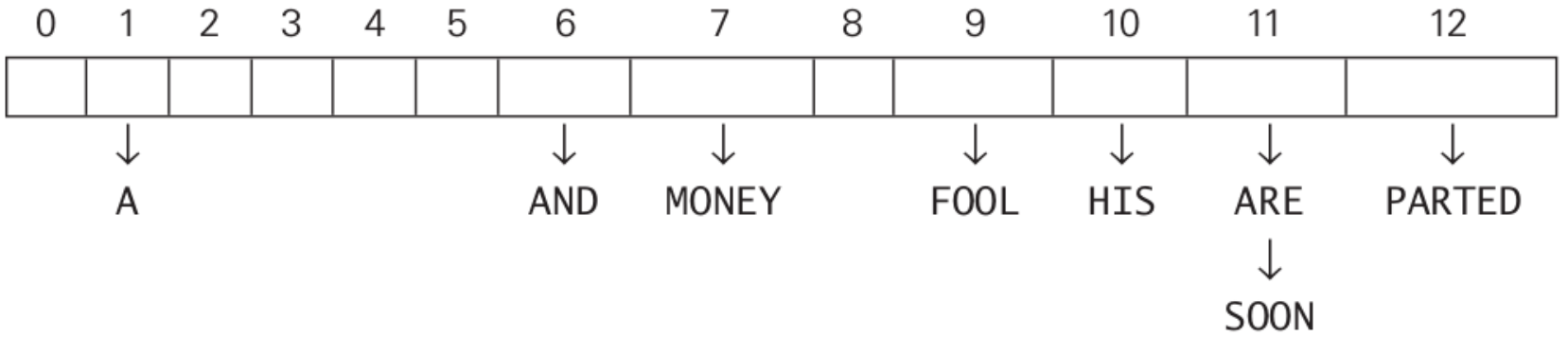
Agenda

Open hashing[3]

Each hash address is associated with a linked list

Example (considering $A = 1$ and fold):

keys	A	FOOL	AND	HIS	MONEY	ARE	SOON	PARTED
hash addresses	1	9	6	10	7	11	11	12



[3] Source: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Open hashing

Insertion:

1. Compute $h(K) = j$
2. Insert in the j-th linked list
   - Sorted list: better search, worse insertion
   - Unsorted: worse search, better insertion

Search:

1. Compute $h(K) = j$
2. If the j-th linked list is empty, element not found
3. Otherwise, search the linked list

Deletion:

- Compute $h(K) = j$
- Remove from the j-th linked list

Dictionary ADT implemented using open hashing

Composite type (Dictionary):

```
1  int m;                                  // size of the hash table
2  int cnt;          // number of elements in the dictionary
3  List[] H;                 // hash table as an array of lists
4  h : Key → 0..m − 1;                          // hash function
```

Algorithm: Dictionary create_dict(int size, h : Key → 0..m-1)

```
1  d.m ← size ; d.cnt ← 0;
2  d.H ← new List[size];
3  for i ← 0 to size − 1 do
       // a list of Entry, which combines Key and E
4      d.H[i] ← create_list();
5  d.h ← h;
6  return d;
```

Dictionary ADT implemented using open hashing

Some assumptions

- The lists are not sorted (always inserting in the end)
- If there is an entry in d with key k, nothing is done

Algorithm: void insert(Dictionary d, Key k, E e)

```
1  if find(d, k) = NULL then
2      pos ← d.h(k);              // h is the hash function
3      l ← d.H[pos];              // H is the hash table
4      entry ← create_entry(k, e);
5      append(l, entry);
```

Open hashing: temporal efficiency

Insertion (unsorted): $\Theta(1)$

Search: depends on the load factor $\alpha = \frac{n}{m}$

- $S \approx 1 + \frac{\alpha}{2}$ and $U = \alpha$, where
  S = succesful, U = unsuccessful,
  n = number of elements in the hash table
- When $\alpha \approx 1$, then in $\Theta(1)$ (in average)

Deletion: similar to the temporal efficiency of searching

Agenda

Closed hashing

All elements are stored in the hash table itself (requirement: $m \geq n$)

Strategies for defining an **offset** in the presence of collisions:

- Linear probing
- Pseudo-random probing
- Quadratic probing
- Double hashing

The strategy should ensure that every hash address can be reached when solving a collision

Closed hashing: linear probing[4]

Insertion: check the cell following the one where the collision occurs

- Probe function (**offset**): $p(K, i) = i$
- Hash table is seen as a circular array
- Example (considering $A = 1$ and fold):

keys	A	FOOL	AND	HIS	MONEY	ARE	SOON	PARTED
hash addresses	1	9	6	10	7	11	11	12

0	1	2	3	4	5	6	7	8	9	10	11	12
	A											
	A								FOOL			
	A					AND			FOOL			
	A					AND			FOOL	HIS		
	A					AND	MONEY		FOOL	HIS		
	A					AND	MONEY		FOOL	HIS	ARE	
	A					AND	MONEY		FOOL	HIS	ARE	SOON
PARTED	A					AND	MONEY		FOOL	HIS	ARE	SOON

Centro de Informática UFPE

Universidade Federal de Pernambuco

[4] Source: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Closed hashing: linear probing

Search:

1. Compute $h(K) = j$
2. If $H[j]$ is empty, element not found
3. If $H[j] = K$, element found
4. Otherwise, check the following position (go back to step 2)
   - Attention to a hash table where $m = n$

Example (considering $A = 1$ and fold):

- $h(\text{“LIT”}) = 2$, as $H[2]$ is empty, “LIT” is not in the table
- $h(\text{“KID´´”}) = 11$, search finishes just after comparing K with “ARE”, “SOON”, “PARTED”, and “A”

Closed hashing: linear probing

Deletion: need to use a special symbol

- Update insertion and search to consider this symbol

Analysis of temporal efficiency is more complicated:

- $S \approx \frac{1}{2}(1 + \frac{1}{1-\alpha})$
- $U \approx \frac{1}{2}(1 + \frac{1}{(1-\alpha)^2})$

Closed hashing: linear probing[5]

Evolution of S and U based on α

α	$\frac{1}{2}(1+\frac{1}{1-\alpha})$	$\frac{1}{2}(1+\frac{1}{(1-\alpha)^2})$
50%	1.5	2.5
75%	2.5	8.5
90%	5.5	50.5

When $\alpha \approx 1$, linear probing deteriorates (primary clustering)

- Higher probability of adding an element to a cluster
- Higher probability of coalescing two clusters

[5] Source: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Closed hashing: pseudo-random probing

Pseudo-random probing:
$p(K, i) = Perm[i - 1]$,
where $Perm$ is a
permutation of $1..m - 1$

Example ($m = 8$)

- $h(k) = k - (8 * \lfloor k/8 \rfloor)$
- $Perm = \{2, 6, 7, 3, 1, 4, 5\}$
- Values: 2, 4, 8, 16, 32, -12

0	1	2	3	4	5	6	7
		2					
		2		4			
8		2		4			
8		2		4		16	

$h(2) = 2$
$h(4) = 4$
$h(8) = 0$
$h(16) = 0$

- $p(k, 1) = Perm[0] = 2$
  $(h(16) + p(k, 1)) \bmod 8 = 2$
- $p(k, 2) = Perm[1] = 6$
  $(h(16) + p(k, 2)) \bmod 8 = 6$

Centro de Informática UFPE
Universidade Federal de Pernambuco

Closed hashing: pseudo-random probing

Pseudo-random probing:
$p(K, i) = Perm[i - 1]$,
where $Perm$ is a
permutation of $1..m - 1$

Example ($m = 8$)

- $h(k) = k - (8 * \lfloor k/8 \rfloor)$
- $Perm = \{2, 6, 7, 3, 1, 4, 5\}$
- Values: 2, 4, 8, 16, 32, -12

0	1	2	3	4	5	6	7
8		2		4		16	32

$h(32) = 0$

- $p(k, 1) = Perm[0] = 2$
  $(h(32) + p(k, 1)) \bmod 8 = 2$
- $p(k, 2) = Perm[1] = 6$
  $(h(32) + p(k, 2)) \bmod 8 = 6$
- $p(k, 3) = Perm[2] = 7$
  $(h(32) + p(k, 3)) \bmod 8 = 7$

Closed hashing: pseudo-random probing

Pseudo-random probing:
$p(K, i) = Perm[i - 1]$,
where $Perm$ is a
permutation of $1..m - 1$

Example ($m = 8$)

- $h(k) = k - (8 * \lfloor k/8 \rfloor)$
- $Perm = \{2, 6, 7, 3, 1, 4, 5\}$
- Values: 2, 4, 8, 16, 32, -12

0	1	2	3	4	5	6	7
8		2	-12	4		16	32

$h(-12) = 4$

- $p(k, 1) = Perm[0] = 2$
  $(h(-12) + p(k, 1)) \bmod 8 = 6$
- $p(k, 2) = Perm[1] = 6$
  $(h(-12) + p(k, 2)) \bmod 8 = 2$
- $p(k, 3) = Perm[2] = 7$
  $(h(-12) + p(k, 3)) \bmod 8 = 3$

Closed hashing: pseudo-random probing

Pseudo-random probing:
$p(K, i) = Perm[i - 1]$,
where $Perm$ is a
permutation of $1..m - 1$

Example ($m = 8$)

- $h(k) = k - (8 * \lfloor k/8 \rfloor)$
- $Perm = \{2, 6, 7, 3, 1, 4, 5\}$
- Values: 2, 4, 8, 16

0	1	2	3	4	5	6	7
8		2		4		16	

$h(2) = 2$
$h(4) = 4$
$h(8) = 0$
$h(16) = 0$

- $p(k, 1) = Perm[0] = 2$
  $(h(16) + p(k, 1)) \bmod 8 = 2$
- $p(k, 2) = Perm[1] = 6$
  $(h(16) + p(k, 2)) \bmod 8 = 6$

Dictionary ADT implemented using closed hashing

Composite type (Dictionary):

```
1  int m;                                  // size of the hash table
2  int cnt;              // number of elements in the dictionary
3  Entry[] H;                 // hash table as an array of Entry
4  int[] Perm;                              // permutation of 1..m-1
5  h : Key → 0..m − 1;                                  // hash function
```

Algorithm: Dictionary create_dict(int size, h : Key → 0..size-1)

```
1  d.m ← size ; d.cnt ← 0;
2  d.H ← new Entry[size];
3  d.Perm ← create_permutation(1..size − 1);
4  d.h ← h;
5  return d;
```

Centro de Informática UFPE

Universidade Federal de Pernambuco

Dictionary ADT implemented using closed hashing

Some assumptions

- d is implemented as a hash table with m positions
- If there is an entry in d with key k, nothing is done

Algorithm: void insert(Dictionary d, Key k, E e)

```
1  if size(d) < d.m ∧ find(d, k) = NULL then
2      pos ← d.h(k);                                   // h is the hash function
3      if d.H[pos] ≠ NULL ∧ d.H[pos] ≠ deleted then
4          i ← 0;
5          repeat
6              i ← i + 1;
7              offset ← d.Perm[i − 1];
8              newPos ← (pos + offset)%d.m;
9          until d.H[newPos] = NULL ∨ d.H[newPos] = deleted;
10         pos ← newPos;
11     entry ← create_entry(k, e);
12     d.H[pos] ← entry;
13     d.cnt = d.cnt + 1;
```

Centro de Informática UFPE

Universidade Federal de Pernambuco

Closed hashing: quadratic probing

Quadratic probing:
$p(K, i) = c_1 i^2 + c_2 i + c3$

$h(2) = 2$
$h(4) = 4$
$h(8) = 0$

Example ($m = 8$):

- $h(k) = k - (8 * \lfloor k/8 \rfloor)$
- $p(k, i) = \frac{i^2+i}{2}$:
- Values: 2, 4, 8, 16, 32, -12

0	1	2	3	4	5	6	7
		2					
		2		4			
8		2		4			

Closed hashing: quadratic probing

Quadratic probing:
$p(K, i) = c_1 i^2 + c_2 i + c3$

Example ($m = 8$):

- $h(k) = k - (8 * \lfloor k/8 \rfloor)$
- $p(k, i) = \frac{i^2+i}{2}$:
- Values: 2, 4, 8, 16, 32, e -12

0	1	2	3	4	5	6	7
8	16	2		4			
8	16	2	32	4			
8	16	2	32	4	-12		

$h(16) = 0$

- $p(k, 1) = \frac{1^2+1}{2} = 1$
  $(h(16) + p(k, 1)) \bmod 8 = 1$

$h(32) = 0$

- $p(k, 1) = \frac{1^2+1}{2} = 1$
  $(h(32) + p(k, 1)) \bmod 8 = 1$
- $p(k, 2) = \frac{2^2+2}{2} = 3$
  $(h(32) + p(k, 2)) \bmod 8 = 3$

$h(-12) = 4$

- $p(k, 1) = \frac{1^2+1}{2} = 1$
  $(h(-12) + p(k, 1)) \bmod 8 = 5$

Centro de Informática UFPE

Universidade Federal de Pernambuco

Closed hashing: pseudo-random | quadratic probing

We might still have secondary clustering

- If $h(k_1) = h(k_2)$, then k_1 and k_2 share the same probe sequence
- Rationale: p ignores the value of K

Alternative: double hashing

Closed hashing: double hashing

A new hash function $s(K)$ is used to determine the probe sequence

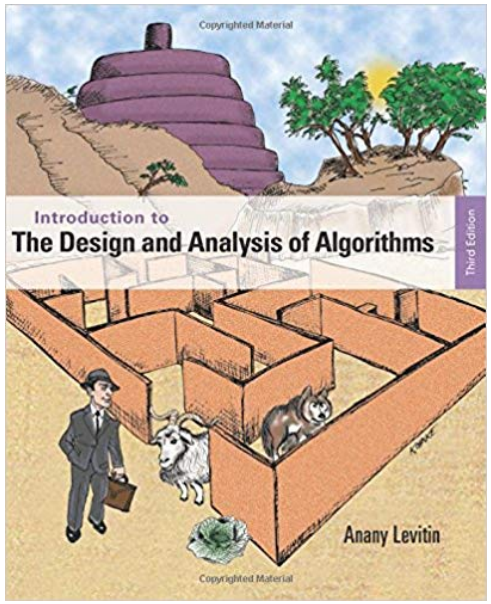
- $p(K, i) = i * s(K)$

Temporal efficiency

- Also deteriorates when $\alpha \approx 1$
- Alternative: resize the hash table and perform rehashing

Agenda

Bibliography



Chapter 1 (pp. 35–37)
Chapter 7 (pp. 253–254, 269–274) Anany Levitin.
Introduction to the Design and Analysis of Algorithms.
3rd edition. Pearson. 2011.

Chapter 4 (pp. 131–138)
Chapter 9 (pp. 314–336)
Clifford Shaffer.
Data Structures and Alg. Analysis. Dover, 2013.

Centro de Informática UFPE

Universidade Federal de Pernambuco

HASH TABLES

Gustavo Carvalho
(ghpc@cin.ufpe.br)

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil

Centro de Informática UFPE

Universidade Federal de Pernambuco