Datenstrukturen und Algorithmen

Exercise 5

FS 2019

Program of today

1. Feedback of last exercise
2. Repetition theory
3. Programming Task

Amortized analysis: push_back

Strategy: double if array is full.

Amortized analysis: push_back

Strategy: double if array is full.

Let $i \in \mathbb{N}$ be the number of elements appended and let $n_i \in \mathbb{N}$ be the array size allocated after appending i.

It holds that

$$n_i = \begin{cases} 1 & \text{if } i = 1 \text{ [Start]} \\ 2 \cdot n_{i-1} & \text{if } i - 1 \in \{2^k : k \in \mathbb{N}\} \text{ [Array full]} \\ n_{i-1} & \text{otherwise} \end{cases}$$

$$n_i = 2^{\lceil \log_2 i \rceil}$$

i	n_i
1	1
2	2
3	4
4	4
5	8
6	8
..	..

Amortized analysis: push_back

Strategy: double if array is full.

[1] According to the task description: 2n initialisations, n copies, 1 new element

Amortized analysis: push_back

Strategy: double if array is full.

Real costs

$$t_i = \begin{cases} 1 & \text{if } i = 1 \text{ [Start]} \\ 3n_{i-1} + 1 & \text{if } i - 1 \in \{2^k : k \in \mathbb{N}\} \text{ [Array full]}^1 \\ 1 & \text{otherwise} \end{cases}$$

[1] According to the task description: 2n initialisations, n copies, 1 new element

Amortized analysis: push_back

Strategy: double if array is full.

Real costs

$$t_i = \begin{cases} 1 & \text{if } i = 1 \text{ [Start]} \\ 3n_{i-1} + 1 & \text{if } i - 1 \in \{2^k : k \in \mathbb{N}\} \text{ [Array full]}^1 \\ 1 & \text{otherwise} \end{cases}$$

Find potential function such that the amortized costs are constant:

$$a_i = t_i + \Phi_i - \Phi_{i-1}$$

[1] According to the task description: 2n initialisations, n copies, 1 new element

Amortized analysis: push_back

Strategy: double if array is full.

Find potential function such that the amortized costs are constant:

$$a_i = t_i + \Phi_i - \Phi_{i-1}$$

Amortized analysis: push_back

Strategy: double if array is full.

Find potential function such that the amortized costs are constant:

$$a_i = t_i + \Phi_i - \Phi_{i-1}$$

$$\begin{aligned}\Phi_i &= 6 \cdot \text{number of elements in the upper half of the array} \\ &= 6 \cdot (i - \frac{n_i}{2}) = 6i - 3n_i\end{aligned}$$

Amortized analysis: push_back

Strategy: double if array is full.

Find potential function such that the amortized costs are constant:

$$a_i = t_i + \Phi_i - \Phi_{i-1}$$

$$\begin{aligned}\Phi_i &= 6 \cdot \text{number of elements in the upper half of the array} \\ &= 6 \cdot (i - \frac{n_i}{2}) = 6i - 3n_i\end{aligned}$$

$$\Phi_i - \Phi_{i-1} = \begin{cases} 6 + 3n_{i-1} - 3 \overbrace{n_i}^{2 \cdot n_{i-1}} & \text{if } i - 1 \in \{2^k : k \in \mathbb{N}\} \text{ [Array full]} \\ 6 & \text{otherwise} \end{cases}$$

Amortized analysis: push_back

Strategy: double if array is full.

Find potential function such that the amortized costs are constant:

$$
\begin{aligned}
a_i &= t_i + \Phi_i - \Phi_{i-1} \\
&= \begin{cases} 3n_{i-1} + 1 + 6 - 3n_{i-1} & \text{if } i-1 \in \{2^k : k \in \mathbb{N}\} \text{ [Array full]} \\ 1 + 6 & \text{otherwise} \end{cases} \\
&\leq 7 \quad \text{for all } i
\end{aligned}
$$

Amortized analysis: pop_back

Strategy: halve if array is three quarters empty.

Amortized analysis: pop_back

Strategy: halve if array is three quarters empty.

$$t_i = \begin{cases} 1 & \text{if array is more than quarter full} \\ \frac{n_{i-1}}{2} + \frac{n_{i-1}}{4} = \frac{3}{4} n_{i-1} & \text{otherwise, then } n_i = \frac{n_{i-1}}{2} \end{cases}$$

Amortized analysis: pop_back

Strategy: halve if array is three quarters empty.

$$t_i = \begin{cases} 1 & \text{if array is more than quarter full} \\ \frac{n_{i-1}}{2} + \frac{n_{i-1}}{4} = \frac{3}{4}n_{i-1} & \text{otherwise, then } n_i = \frac{n_{i-1}}{2} \end{cases}$$

Find potential function such that the amortized costs are constant:

$$a_i = t_i + \Phi_i - \Phi_{i-1}$$

Amortized analysis: pop_back

Strategy: halve if array is three quarters empty.

$$t_i = \begin{cases} 1 & \text{if array is more than quarter full} \\ \frac{n_{i-1}}{2} + \frac{n_{i-1}}{4} = \frac{3}{4}n_{i-1} & \text{otherwise, then } n_i = \frac{n_{i-1}}{2} \end{cases}$$

Find potential function such that the amortized costs are constant:

$$a_i = t_i + \Phi_i - \Phi_{i-1}$$

Let k_i be the number of elements in the array in step i

$$\Phi_i = 3 \cdot \text{number of empty elements in the lower half of array } (1, \ldots, \frac{n}{2})$$

$$= 3 \cdot (\frac{n_i}{2} - k_i)$$

Amortized analysis: pop_back

Strategy: halve if array is three quarters empty.

$$t_i = \begin{cases} 1 & \text{if array is more than quarter full} \\ \frac{n_{i-1}}{2} + \frac{n_{i-1}}{4} = \frac{3}{4} n_{i-1} & \text{otherwise, then } n_i = \frac{n_{i-1}}{2} \end{cases}$$

Find potential function such that the amortized costs are constant:

$$a_i = t_i + \Phi_i - \Phi_{i-1}$$

Let k_i be the number of elements in the array in step i

$\Phi_i = 3 \cdot$ number of empty elements in the lower half of array $(1, \ldots, \dfrac{n}{2})$

$$= 3 \cdot (\frac{n_i}{2} - k_i)$$

Amortized analysis: pop_back

Strategy: halve if array is three quarters empty. Find potential function such that the amortized costs are constant:

$$a_i = t_i + \Phi_i - \Phi_{i-1}$$

$$\Phi_i = 3 \cdot (\frac{n_i}{2} - k_i)$$

$$\Phi_i - \Phi_{i-1} = \begin{cases} 3 & \text{if array is more than quarter full} \\ 3 \cdot \left(1 + \frac{n_{i-1}}{4} - \frac{n_{i-1}}{2})\right) & \text{otherwise} \end{cases}$$

Amortized analysis: pop_back

Strategy: halve if array is three quarters empty. Find potential function such that the amortized costs are constant:

$$a_i = t_i + \Phi_i - \Phi_{i-1}$$

$$\Phi_i = 3 \cdot (\frac{n_i}{2} - k_i)$$

$$\Phi_i - \Phi_{i-1} = \begin{cases} 3 & \text{if array is more than quarter full} \\ 3 \cdot \left(1 + \frac{n_{i-1}}{4} - \frac{n_{i-1}}{2})\right) & \text{otherwise} \end{cases}$$

$$\Rightarrow 4 \geq a_i \text{ (in both cases)}$$

Amortized analysis: pop and push

$$\Phi_i = 6 \cdot \text{number elements in the upper half}$$
$$+ 3 \cdot \text{number empty slots in the lower half}$$

2. Repetition theory

Dictionary in $C{+}{+}$

Associative Container `std::unordered_map<>`

```
// Create an unordered_map of strings that map to strings
std::unordered_map<std::string, std::string> u = {
        {"RED","#FF0000"}, {"GREEN","#00FF00"}
};

u["BLUE"] = "#0000FF"; // Add

std::cout << "The HEX of color RED is: " << u["RED"] << "\n";

for( const auto& n : u ) // iterate over key-value pairs
  std::cout << n.first << ":" << n.second << "\n";
```

Resolving Collisions: Chaining

Example $m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \bmod m$.

Keys 12

Direct Chaining of the Colliding entries

hash table

0	1	2	3	4	5	6

Colliding entries

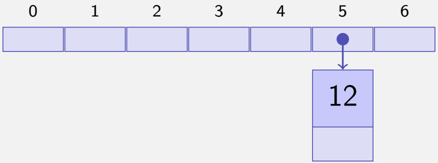
Resolving Collisions: Chaining

Example $m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \bmod m$.

Keys 12 , 55

Direct Chaining of the Colliding entries

hash table



Colliding entries

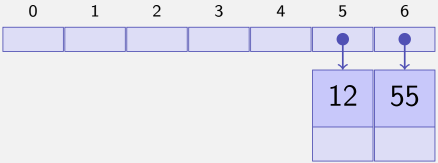
Resolving Collisions: Chaining

Example $m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \bmod m$.

Keys 12 , 55 , 5

Direct Chaining of the Colliding entries

hash table



Colliding entries

Resolving Collisions: Chaining

Example $m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \bmod m$.

Keys 12 , 55 , 5 , 15

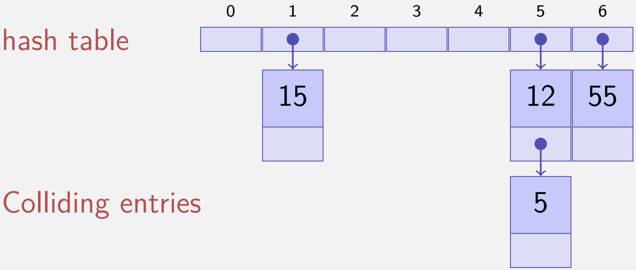
Direct Chaining of the Colliding entries

Resolving Collisions: Chaining

Example $m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \bmod m$.

Keys 12 , 55 , 5 , 15 , 2

Direct Chaining of the Colliding entries

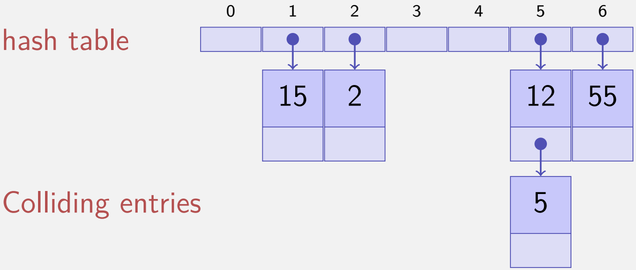


Resolving Collisions: Chaining

Example $m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \bmod m$.

Keys 12 , 55 , 5 , 15 , 2 , 19

Direct Chaining of the Colliding entries

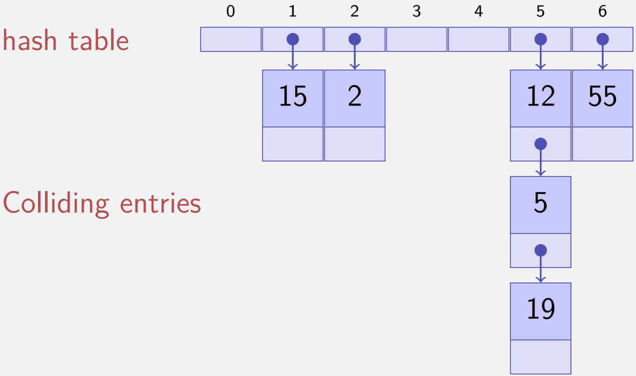


Resolving Collisions: Chaining

Example $m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \bmod m$.

Keys 12 , 55 , 5 , 15 , 2 , 19 , 43

Direct Chaining of the Colliding entries

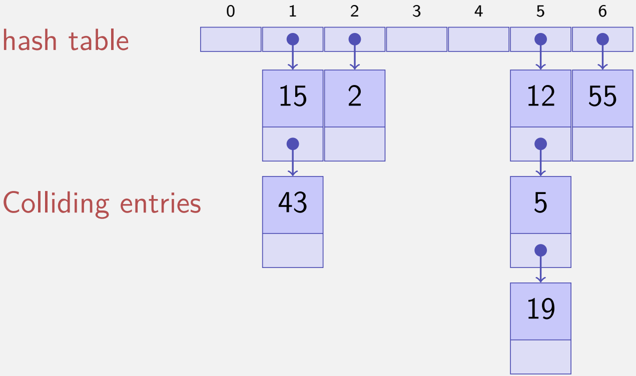


Resolving Collisions: Chaining

Example $m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \bmod m$.

Keys 12 , 55 , 5 , 15 , 2 , 19 , 43

Direct Chaining of the Colliding entries



Examples of popular Hash Functions

Division method

$$h(k) = k \bmod m$$

- m prime, not too close[2] to powers of 2 or 10

[2] Example: let $m = 2^k - 1$, then $h(k) = h(k \cdot 2^k)$

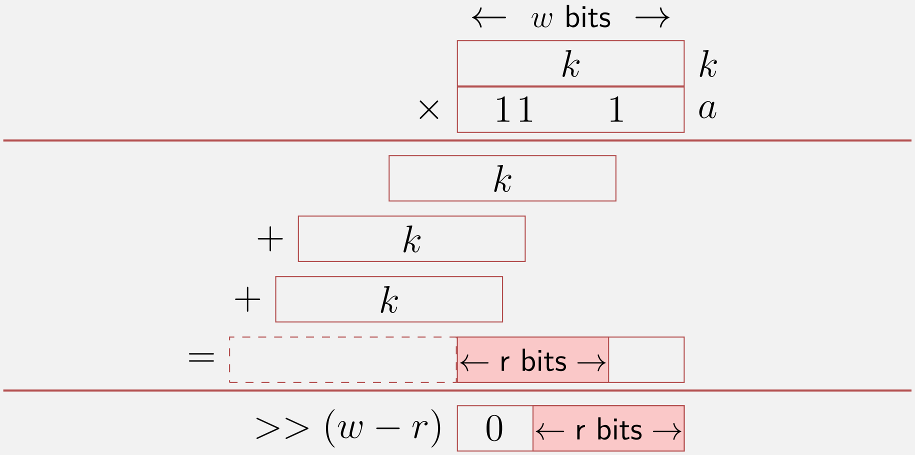
Examples of popular Hash Functions

Multiplication method

$$h(k) = \left\lfloor (a \cdot k \bmod 2^w)/2^{w-r} \right\rfloor \bmod m$$

- $m = 2^r$, $w =$ size of the machine word in bits.
- Multiplication adds k along all bits of a, integer division with 2^{w-r} and $\mathrm{mod} m$ extract the upper r bits.
- Written as code `a * k >> (w-r)`
- A good value of a: $\left\lfloor \frac{\sqrt{5}-1}{2} \cdot 2^w \right\rfloor$: Integer that represents the first w bits of the fractional part of the irrational number.

Illustration



Open Addressing

Store the colliding entries directly in the hash table using a *probing function* $s : \mathcal{K} \times \{0, 1, \ldots, m-1\} \to \{0, 1, \ldots, m-1\}$

Key table position along a *probing sequence*

$$S(k) := (s(k,0), s(k,1), \ldots, s(k,m-1)) \mod m$$

Probing sequence must for each $k \in \mathcal{K}$ be a permutation of $\{0, 1, \ldots, m-1\}$

Algorithms for open addressing

- `search`(k) Traverse table entries according to $S(k)$. If k is found, return true. If the probing sequence is finished or an empty position is reached, return false.
- `insert`(k) Search for k in the table according to $S(k)$. If k is not present, insert k at the first free position in the probing sequence. [3]
- `delete`(k) Search k in the table according to $S(k)$. If k is found, mark the position of k with a `deleted` flag

[3] A position is also free when it is non-empty and contains a `deleted` flag.

Linear Probing

$s(k,j) = h(k) + j \Rightarrow S(k) = (h(k), h(k)+1, \ldots, h(k)+m-1) \mod m$

Linear Probing

$s(k, j) = h(k) + j \Rightarrow S(k) = (h(k), h(k) + 1, \ldots, h(k) + m - 1) \mod m$

Example $m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \mod m$.
Key 12

0	1	2	3	4	5	6

Linear Probing

$s(k, j) = h(k) + j \Rightarrow S(k) = (h(k), h(k) + 1, \ldots, h(k) + m - 1) \mod m$

Example $m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \mod m$.
Key 12 , 55

0	1	2	3	4	5	6
					12	

Linear Probing

$s(k, j) = h(k) + j \Rightarrow S(k) = (h(k), h(k) + 1, \ldots, h(k) + m - 1) \bmod m$

Example $m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \bmod m$.
Key 12 , 55 , 5

0	1	2	3	4	5	6
					12	55

Linear Probing

$s(k,j) = h(k) + j \Rightarrow S(k) = (h(k), h(k)+1, \ldots, h(k)+m-1) \mod m$

Example $m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \mod m$.

Key 12 , 55 , 5 , 15

0	1	2	3	4	5	6
5					12	55

Linear Probing

$s(k, j) = h(k) + j \Rightarrow S(k) = (h(k), h(k) + 1, \ldots, h(k) + m - 1)$
$\bmod m$

Example $m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \bmod m$.
Key 12 , 55 , 5 , 15 , 2

0	1	2	3	4	5	6
5	15				12	55

Linear Probing

$$s(k, j) = h(k) + j \Rightarrow S(k) = (h(k), h(k) + 1, \dots, h(k) + m - 1) \mod m$$

Example $m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \mod m$.
Key 12 , 55 , 5 , 15 , 2 , 19

0	1	2	3	4	5	6
5	15	2			12	55

Linear Probing

$s(k, j) = h(k) + j \Rightarrow S(k) = (h(k), h(k) + 1, \dots, h(k) + m - 1) \mod m$

Example $m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \mod m$.
Key 12 , 55 , 5 , 15 , 2 , 19

0	1	2	3	4	5	6
5	15	2	19		12	55

Quadratic Probing

$$s(k,j) = h(k) + \lceil j/2 \rceil^2 (-1)^{j+1}$$
$$S(k) = (h(k), h(k)+1, h(k)-1, h(k)+4, h(k)-4, \ldots) \mod m$$

Quadratic Probing

$$s(k, j) = h(k) + \lceil j/2 \rceil^2 (-1)^{j+1}$$
$$S(k) = (h(k), h(k) + 1, h(k) - 1, h(k) + 4, h(k) - 4, \dots) \mod m$$

Example $m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \bmod m$.

Keys 12

0	1	2	3	4	5	6

Quadratic Probing

$$s(k,j) = h(k) + \lceil j/2 \rceil^2 (-1)^{j+1}$$
$$S(k) = (h(k), h(k)+1, h(k)-1, h(k)+4, h(k)-4, \dots) \mod m$$

Example $m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \bmod m$.

Keys 12 , 55

0	1	2	3	4	5	6
					12	

Quadratic Probing

$$s(k,j) = h(k) + \lceil j/2 \rceil^2 (-1)^{j+1}$$
$$S(k) = (h(k), h(k)+1, h(k)-1, h(k)+4, h(k)-4, \dots) \mod m$$

Example $m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \mod m$.

Keys 12 , 55 , 5

0	1	2	3	4	5	6
					12	55

Quadratic Probing

$$s(k,j) = h(k) + \lceil j/2 \rceil^2 (-1)^{j+1}$$
$$S(k) = (h(k), h(k)+1, h(k)-1, h(k)+4, h(k)-4, \dots) \mod m$$

Example $m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \bmod m$.

Keys 12 , 55 , 5 , 15

0	1	2	3	4	5	6
				5	12	55

Quadratic Probing

$$s(k,j) = h(k) + \lceil j/2 \rceil^2 (-1)^{j+1}$$
$$S(k) = (h(k), h(k)+1, h(k)-1, h(k)+4, h(k)-4, \dots) \mod m$$

Example $m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \bmod m$.

Keys 12 , 55 , 5 , 15 , 2

0	1	2	3	4	5	6
	15			5	12	55

Quadratic Probing

$$s(k,j) = h(k) + \lceil j/2 \rceil^2 (-1)^{j+1}$$
$$S(k) = (h(k), h(k)+1, h(k)-1, h(k)+4, h(k)-4, \dots) \mod m$$

Example $m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \bmod m$.
Keys 12 , 55 , 5 , 15 , 2 , 19

0	1	2	3	4	5	6
	15	2		5	12	55

Quadratic Probing

$$s(k,j) = h(k) + \lceil j/2 \rceil^2 (-1)^{j+1}$$
$$S(k) = (h(k), h(k)+1, h(k)-1, h(k)+4, h(k)-4, \dots) \mod m$$

Example $m = 7$, $\mathcal{K} = \{0, \dots, 500\}$, $h(k) = k \bmod m$.

Keys 12 , 55 , 5 , 15 , 2 , 19

0	1	2	3	4	5	6
19	15	2		5	12	55

Double Hashing

Two hash functions $h(k)$ and $h'(k)$. $s(k, j) = h(k) + j \cdot h'(k)$.

$S(k) = (h(k), h(k) + h'(k), h(k) + 2h'(k), \ldots, h(k) + (m-1)h'(k)) \mod m$

Double Hashing

Two hash functions $h(k)$ and $h'(k)$. $s(k, j) = h(k) + j \cdot h'(k)$.
$S(k) = (h(k), h(k) + h'(k), h(k) + 2h'(k), \ldots, h(k) + (m-1)h'(k)) \mod m$

Example:
$m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \bmod 7$, $h'(k) = 1 + k \bmod 5$.
Keys 12

0	1	2	3	4	5	6

Double Hashing

Two hash functions $h(k)$ and $h'(k)$. $s(k,j) = h(k) + j \cdot h'(k)$.
$S(k) = (h(k), h(k)+h'(k), h(k)+2h'(k), \ldots, h(k)+(m-1)h'(k)) \mod m$

Example:
$m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \bmod 7$, $h'(k) = 1 + k \bmod 5$.
Keys 12 , 55

0	1	2	3	4	5	6
					12	

Double Hashing

Two hash functions $h(k)$ and $h'(k)$. $s(k,j) = h(k) + j \cdot h'(k)$.
$S(k) = (h(k), h(k) + h'(k), h(k) + 2h'(k), \ldots, h(k) + (m-1)h'(k)) \mod m$

Example:
$m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \bmod 7$, $h'(k) = 1 + k \bmod 5$.
Keys 12 , 55 , 5

0	1	2	3	4	5	6
					12	55

Double Hashing

Two hash functions $h(k)$ and $h'(k)$. $s(k,j) = h(k) + j \cdot h'(k)$.

$S(k) = (h(k), h(k) + h'(k), h(k) + 2h'(k), \ldots, h(k) + (m-1)h'(k)) \mod m$

Example:

$m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \bmod 7$, $h'(k) = 1 + k \bmod 5$.

Keys 12 , 55 , 5 , 15

0	1	2	3	4	5	6
5					12	55

Double Hashing

Two hash functions $h(k)$ and $h'(k)$. $s(k,j) = h(k) + j \cdot h'(k)$.

$S(k) = (h(k), h(k) + h'(k), h(k) + 2h'(k), \ldots, h(k) + (m-1)h'(k)) \mod m$

Example:

$m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \bmod 7$, $h'(k) = 1 + k \bmod 5$.

Keys 12 , 55 , 5 , 15 , 2

0	1	2	3	4	5	6
5	15				12	55

Double Hashing

Two hash functions $h(k)$ and $h'(k)$. $s(k,j) = h(k) + j \cdot h'(k)$.

$S(k) = (h(k), h(k) + h'(k), h(k) + 2h'(k), \ldots, h(k) + (m-1)h'(k)) \mod m$

Example:

$m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \bmod 7$, $h'(k) = 1 + k \bmod 5$.

Keys 12 , 55 , 5 , 15 , 2 , 19

0	1	2	3	4	5	6
5	15	2			12	55

Double Hashing

Two hash functions $h(k)$ and $h'(k)$. $s(k, j) = h(k) + j \cdot h'(k)$.
$S(k) = (h(k), h(k) + h'(k), h(k) + 2h'(k), \ldots, h(k) + (m-1)h'(k)) \mod m$

Example:
$m = 7$, $\mathcal{K} = \{0, \ldots, 500\}$, $h(k) = k \mod 7$, $h'(k) = 1 + k \mod 5$.
Keys 12 , 55 , 5 , 15 , 2 , 19

0	1	2	3	4	5	6
5	15	2	19		12	55

3. Programming Task

Finding a Sub-Array

- Given: two integer arrays $A = (a_0, \ldots, a_{n-1})$ and $B = (b_0, \ldots, b_{k-1})$
- Task: Find position of B in A.

Finding a Sub-Array

- Given: two integer arrays $A = (a_0, \ldots, a_{n-1})$ and $B = (b_0, \ldots, b_{k-1})$
- Task: Find position of B in A.
- Naive: Loop through A, check whether the following k entries match B.

Finding a Sub-Array

- Given: two integer arrays $A = (a_0, \ldots, a_{n-1})$ and $B = (b_0, \ldots, b_{k-1})$
- Task: Find position of B in A.
- Naive: Loop through A, check whether the following k entries match B.
  - $O(nk)$ comparison operations

Finding a Sub-Array

- Given: two integer arrays $A = (a_0, \ldots, a_{n-1})$ and $B = (b_0, \ldots, b_{k-1})$
- Task: Find position of B in A.
- Naive: Loop through A, check whether the following k entries match B.
  - $O(nk)$ comparison operations
- Solution using hashing: Calculate hash $h(B)$ and compare it to $h((a_i, a_{i+1}, \ldots, a_{i+k-1}))$.
- Avoid re-computing $h((a_i, a_{i+1}, \ldots, ai + k - 1)$ for each i $\implies O(n)$ expected

Sliding Window Hash

- Possible hash function: sum of all elements:
  - Can be updated easily: subtract a_i and add a_{i+k}.
  - However: bad hash function

Sliding Window Hash

- Possible hash function: sum of all elements:
  - Can be updated easily: subtract a_i and add a_{i+k}.
  - However: bad hash function
- Better:

$$H_{c,m}((a_i, \cdots, a_{i+k-1})) = \left(\sum_{j=0}^{k-1} a_{i+j} \cdot c^{k-j-1} \right) \bmod m$$

  - $c = 1021$ prime number
  - $m = 2^{15}$ `int`, no overflows at calculations

Sliding Window Hash

```
template<typename It1, typename It2>
It1 findOccurrence(const It1 from, const It1 to,
                   const It2 begin, const It2 end)
{
  const unsigned k = end - begin;
  const unsigned M = 32768;
  const unsigned C = 1021;

  // your code here
  // ...
```

Sliding Window Hash

```
  // elements can be compared using std::equal:
  if(std::equal(window_left, window_right, begin, end))
      return current;

  // if no occurrence is found return end of array
  return to;
}
```

Sliding Window Hash

Make sure that

- the algorithm computes c^k only once,
- all computations are modulo m for all values in order not to get an overflow (recall the rules of modular arithmetic), and
- the values are always positive (e.g., by adding multiples of m).

Questions?