Cuckoo hashing

- In cuckoo hashing we have two hash tables of the same size, each of them more than half empty and each hash table has its own hash function (so we have two different hash functions).
- For each element to be added we can compute two positions: one from the first hash table and one from the second. In case of cuckoo hashing, it is guaranteed that an element will be on one of these positions.
- Search is simple, because we only have to look at these two positions.
- Delete is simple, because we only have to look at these two positions and set to empty the one where we find the element.

- When we want to insert a new element we will compute its position in the first hash table. If the position is empty, we will place the element there.

- If the position in the first hash table is not empty, we will kick out the element that is currently there, and place the new element into the first hash table.

- The element that was kicked off, will be placed at its position in the second hash table. If that position is occupied, we will kick off the element from there and place it into its position in the first hash table.

- We repeat the above process until we will get an empty position for an element.

- If we get back to the same location with the same key we have a cycle and we cannot add this element $\Rightarrow$ resize, rehash

Cuckoo hashing - example

- Assume that we have two hash tables, with $m = 11$ positions and the following hash functions:
  - h1(k) = k % 11
  - h2(k) = (k div 11) % 11

Position	0	1	2	3	4	5	6	7	8	9	10
T											

Position	0	1	2	3	4	5	6	7	8	9	10
T											

Cuckoo hashing - example

- Insert key 20
  - $h1(20) = 9$ - empty position, element added in the first table
- Insert key 50
  - $h1(50) = 6$ - empty position, element added in the first table

Position	0	1	2	3	4	5	6	7	8	9	10
T							50			20	

Position	0	1	2	3	4	5	6	7	8	9	10
T											

Cuckoo hashing - example

- Insert key 53
  - h1(53) = 9 - occupied
  - 53 goes in the first hash table, and it sends 20 in the second to position h2(20) = 1

Position	0	1	2	3	4	5	6	7	8	9	10
T							50			53	

Position	0	1	2	3	4	5	6	7	8	9	10
T		20									

Cuckoo hashing - example

- Insert key 75
  - h1(75) = 9 - occupied
  - 75 goes in the first hash table, and it sends 53 in the second to position h2(53) = 4

Position	0	1	2	3	4	5	6	7	8	9	10
T							50			75	

Position	0	1	2	3	4	5	6	7	8	9	10
T		20			53						

Cuckoo hashing example

- Insert key 100
  - $h1(100) = 1$ - empty position
- Insert key 67
  - $h1(67) = 1$ - occupied
  - 67 goes in the first hash table, and it sends 100 in the second to position $h2(100) = 9$

Position	0	1	2	3	4	5	6	7	8	9	10
T		67					50			75	

Position	0	1	2	3	4	5	6	7	8	9	10
T		20			53					100	

Cuckoo hashing example

- Insert key 105
  - h1(105) = 6 - occupied
  - 105 goes in the first hash table, and it sends 50 in the second to position h2(50) = 4
  - 50 goes in the second hash table, and it sends 53 to the first one, to position h1(53) = 9
  - 53 goes in the first hash table, and it sends 75 to the second one, to position h2(75) = 6

Position	0	1	2	3	4	5	6	7	8	9	10
T		67					105			53	

Position	0	1	2	3	4	5	6	7	8	9	10
T		20			50		75			100	

Cuckoo hashing example

- Insert key 3
  - h1(3) = 3 - empty position
- Insert key 36
  - h1(36) = 3 - occupied
  - 36 goes in the first hash table, and it sends 3 in the second to position h2(3) = 0

Position	0	1	2	3	4	5	6	7	8	9	10
T		67		36			105			53	

Position	0	1	2	3	4	5	6	7	8	9	10
T	3	20			50		75			100	

Cuckoo hashing example

- Insert key 39
  - $h1(39) = 6$ - occupied
  - 39 goes in the first hash table and it sends 105 in the second to position $h2(105) = 9$
  - 105 goes to the second hash table and it sends 100 in the first to position $h1(100) = 1$
  - 100 goes in the first hash table and it sends 67 in the second to position $h2(67) = 6$
  - 67 goes in the second hash table and it sends 75 in the first to position $h1(75) = 9$
  - 75 goes in the first hash table and it sends 53 in the second to position $h2(53) = 4$
  - 53 goes in the second hash table and it sends 50 in the first to position $h1(50) = 6$
  - 50 goes in the first hash table and it sends 39 in the second to position $h2(39) = 3$

Cuckoo hashing example

Position	0	1	2	3	4	5	6	7	8	9	10
T		100		36			50			75	

Position	0	1	2	3	4	5	6	7	8	9	10
T	3	20		39	53		67			105	

- It can happen that we cannot insert a key because we get in a cycle. In these situation we have to increase the size of the tables and rehash the elements.
- While in some situation insert moves a lot of elements, it can be shown that if the load factor of the tables is below 0.5, the probability of a cycles is low and it is very unlikely that more than $O(log_2 n)$ elements will be moved.

- If we use two tables and each position from a table holds one element at most, the tables have to have load factor below 0.5 to work well.
- If we use three tables, the tables can have load factor of 0.91 and for 4 tables we have 0.97

Perfect hashing

- Assume that we know all the keys in advance and we use *separate chaining* for collision resolution $\Rightarrow$ the more lists we make, the shorter the lists will be (reduced number of collisions) $\Rightarrow$ if we could make a large number of list, each would have one element only (no collision).
- How large should we make the hash table to make sure that there are no collisions?
- If $M = N^2$, it can be shown that the table is collision free with probability at least $1/2$.
- Start building the hash table. If you detect a collision, just choose a new hash function and start over (expected number of trials is at most 2).

Perfect hashing

- Having a table of size N^2 is impractical.
- Solution instead:
  - Use a hash table of size N (*primary* hash table).
  - Instead of using linked list for collision resolution (as in separate chaining) each element of the hash table is another hash table (*secondary hash table*)
  - Make the secondary hash table of size ${n_j}^2$, where n_j is the number of elements from this hash table.
  - Each secondary hash table will be constructed with a different hash function, and will be reconstructed until it is collision free.
- This is called **perfect hashing**.
- It can be shown that the total space needed for the secondary hash tables is at most $2N$.

- Perfect hashing requires multiple hash functions, this is why we use *universal hashing*.
- Let p be a prime number, larger than the largest possible key.
- The universal hash function family $\mathcal{H}$ can be defined as:

$$\mathcal{H} = \{H_{a,b}(x) = ((a * x + b) \ \% \ p) \ \% \ m)$$

$$\text{where } 1 \leq a \leq p-1, 0 \leq b \leq p-1$$

- a and b are chosen randomly when the hash function is initialized.

Perfect hashing - example

- Insert into a hash table with perfect hashing the letters from "PERFECT HASHING EXAMPLE". Since we want no collisions at all, we are going to consider only the unique letters: "PERFCTHASINGXML"
- Since we are inserting $N = 15$ elements, we will take $m = 15$.
- For each letter, the *hashCode* is the index of the letter in the alphabet.

Letter	P	E	R	F	C	T	H	A	S	I	N	G	X	M	L
HashCode	16	5	18	6	3	20	8	1	19	9	14	7	24	13	12

Perfect hashing - example

- p has to be a prime number larger than the maximum key $\Rightarrow$ 29
- The hash function will be:

$$H_{a,b}(x) = ((a * x + b) \ \% \ p) \ \% \ m$$

- where a will be 3 and b will be 2 (chosen randomly).

Letter	P	E	R	F	C	T	H	A	S	I	N	G	X	M	L
HashCode	16	5	18	6	3	20	8	1	19	9	14	7	24	13	12
H(HashCode)	6	2	12	5	11	4	11	5	1	0	0	8	1	12	9

- Collisions:
  - position 0 - I, N
  - position 1 - S, X
  - position 2 - E
  - position 4 - T
  - position 5 - F, A
  - position 6 - P
  - position 8 - G
  - position 9 - L
  - position 11 - C, H
  - position 12 - R, M

Perfect hashing - example

- For the positions where we have no collision (only one element hashed to it) we will have a secondary hash table with only one element, and hash function $h(x) = 0$
- For the positions where we have two elements, we will have a secondary hash table with 4 positions and different hash functions, taken from the same universe, with different random values for a and b.
- For example for position 0, we can define $a = 4$ and $b = 11$ and we will have:
  $h(I) = h(9) = 2$
  $h(N) = h(14) = 1$

Perfect hashing - example

- Assume that for the secondary hash table from position 1 we will choose $a = 5$ and $b = 2$.
- Positions for the elements will be:
$h(S) = h(19) = ((5 * 19 + 2)\%29)\%4 = 2$
$h(X) = h(24) = ((5 * 24 + 2)\%29)\%4 = 2$
- In perfect hashing we should not have collisions, so we will simply chose another hash function: another random values for a and b. Choosing for example $a = 2$ and $b = 13$, we will have h(S) = 2 and h(X) = 3.

Perfect hashing

- When perfect hashing is used and we search for an element we will have to check at most 2 positions (position in the primary and in the secondary table).
- This means that worst case performance of the table is $\Theta(1)$.
- But in order to use perfect hashing, we need to have static keys: once the table is built, no new elements can be added.