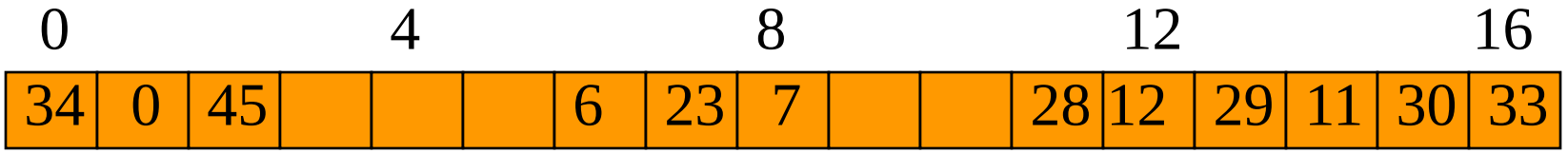
Overflow Handling

- An overflow occurs when the home bucket for a new pair (key, element) is full.
- We may handle overflows by:
  - Search the hash table in some systematic fashion for a bucket that is not full.
    - Linear probing (linear open addressing).
    - Quadratic probing.
    - Random probing.
  - Eliminate overflows by permitting each bucket to keep a list of all pairs for which it is the home bucket.
    - Array linear list.
    - Chain.

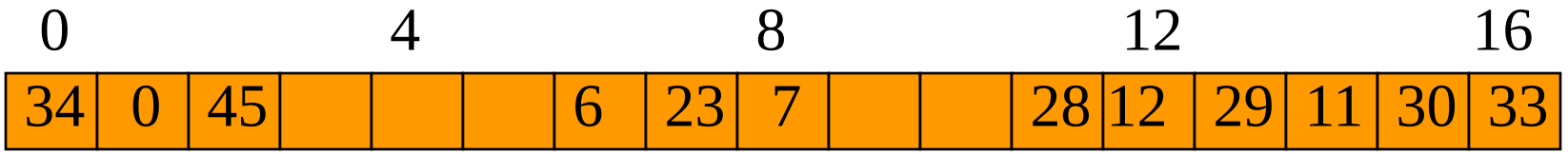
Linear Probing – Get And Insert

- divisor = b (number of buckets) = 17.
- Home bucket = key % 17.

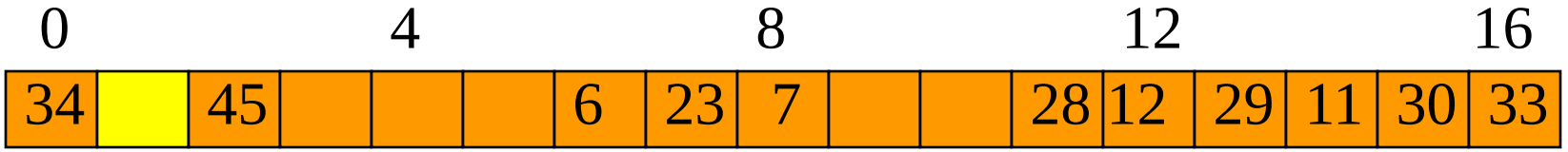


- Insert pairs whose keys are 6, 12, 34, 29, 28, 11, 23, 7, 0, 33, 30, 45

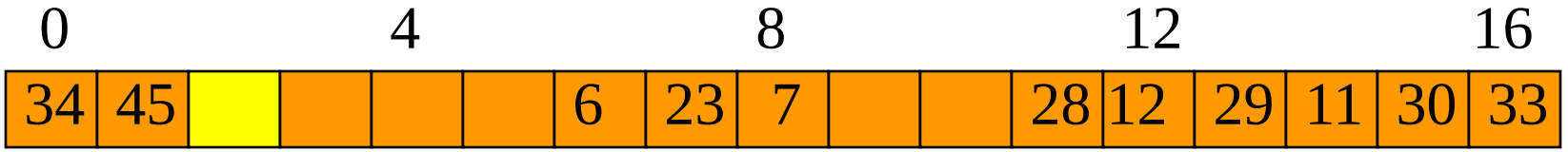
Linear Probing – Delete



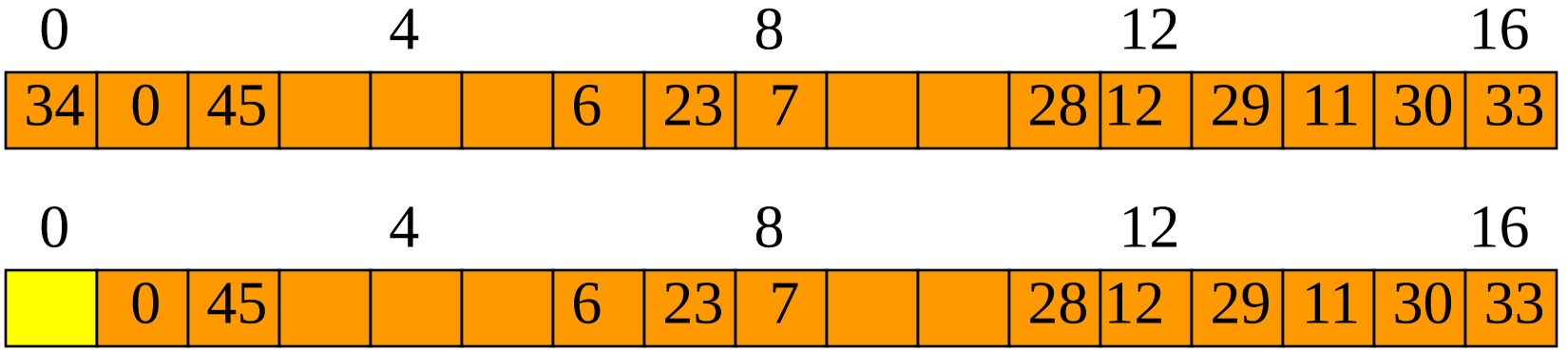
- Delete(0)



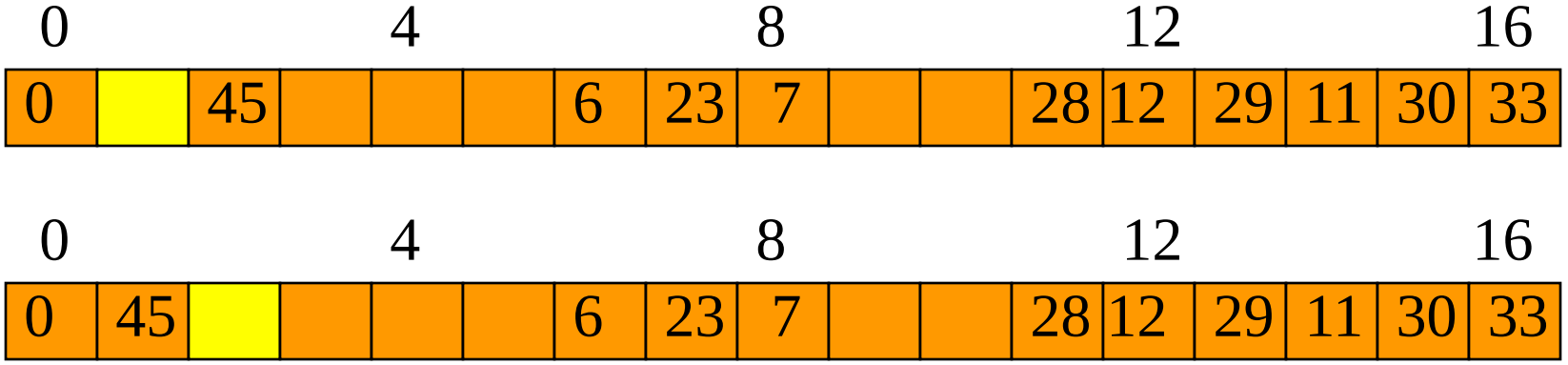
- Search cluster for pair (if any) to fill vacated bucket.



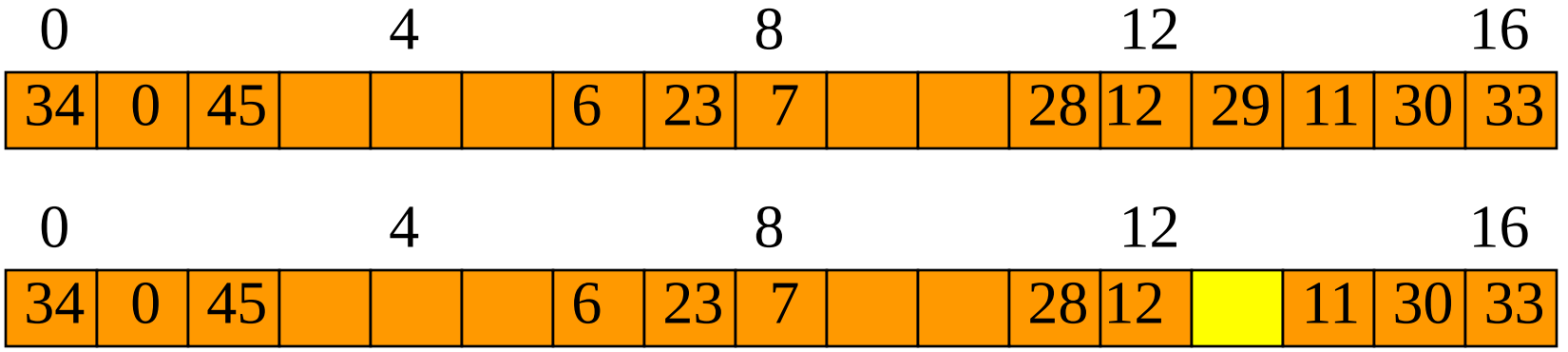
Linear Probing – Delete(34)



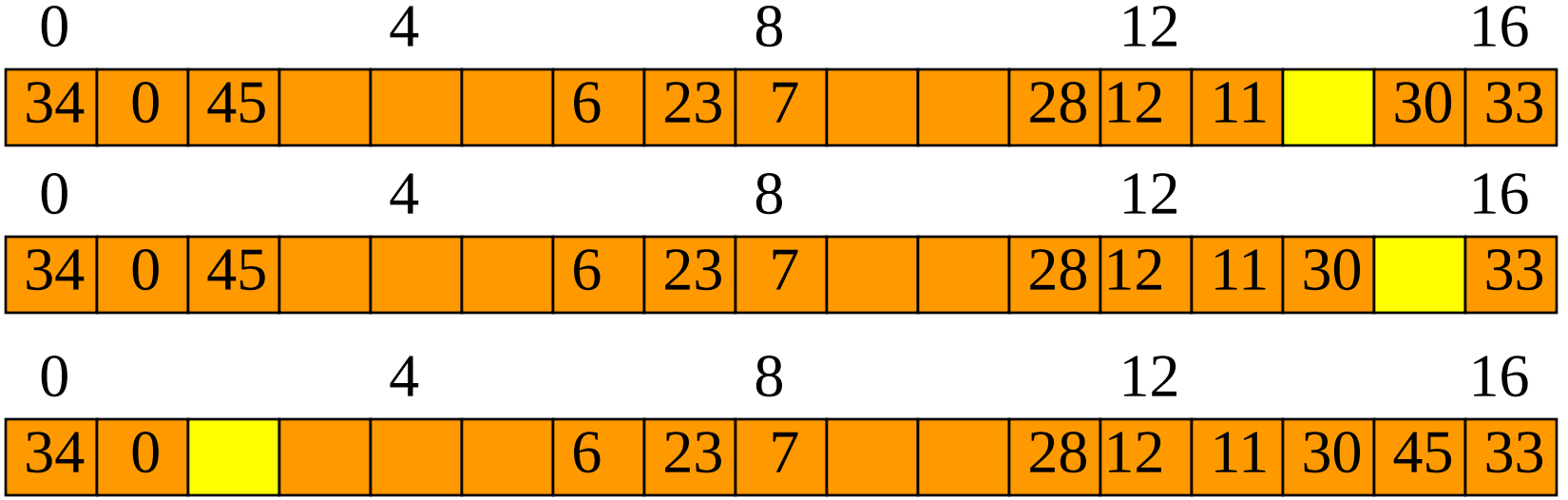
- Search cluster for pair (if any) to fill vacated bucket.



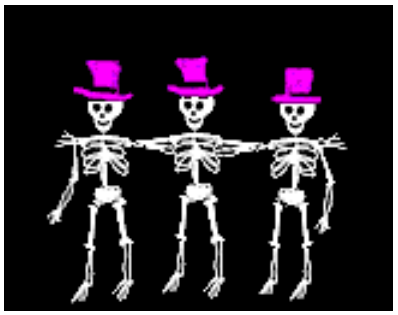
Linear Probing – Delete(29)



- Search cluster for pair (if any) to fill vacated bucket.



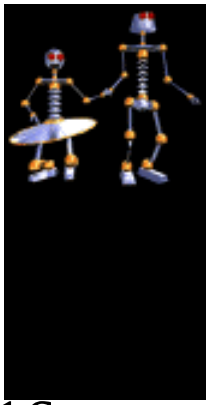
Performance Of Linear Probing



0				4				8				12				16
34	0	45				6	23	7			28	12	29	11	30	33

- Worst-case find/insert/erase time is $\Theta(n)$, where n is the number of pairs in the table.
- This happens when all pairs are in the same cluster.

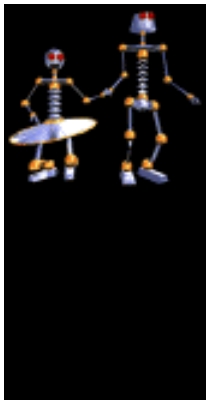
Expected Performance



0			4			8					12					16
34	0	45				6	23	7			28	12	29	11	30	33

- α = loading density = (number of pairs)/b.
  - α = 12/17.
- S_n = expected number of buckets examined in a successful search when n is large
- U_n = expected number of buckets examined in a unsuccessful search when n is large
- Time to put and remove governed by U_n.

Expected Performance



- $S_n \sim \frac{1}{2}(1 + 1/(1 - \alpha))$
- $U_n \sim \frac{1}{2}(1 + 1/(1 - \alpha)^2)$
- Note that $0 <= \alpha <= 1$.

alpha	S_n	U_n
0.50	1.5	2.5
0.75	2.5	8.5
0.90	5.5	50.5

$\alpha <= 0.75$ is recommended.

Hash Table Design

- Performance requirements are given, determine maximum permissible loading density.
- We want a successful search to make no more than 10 compares (expected).
  - $S_n \sim \frac{1}{2}(1 + 1/(1 - \alpha))$
  - $\alpha <= 18/19$
- We want an unsuccessful search to make no more than 13 compares (expected).
  - $U_n \sim \frac{1}{2}(1 + 1/(1 - \alpha)^2)$
  - $\alpha <= 4/5$
- So $\alpha <= \min\{18/19, 4/5\} = 4/5$.

Hash Table Design

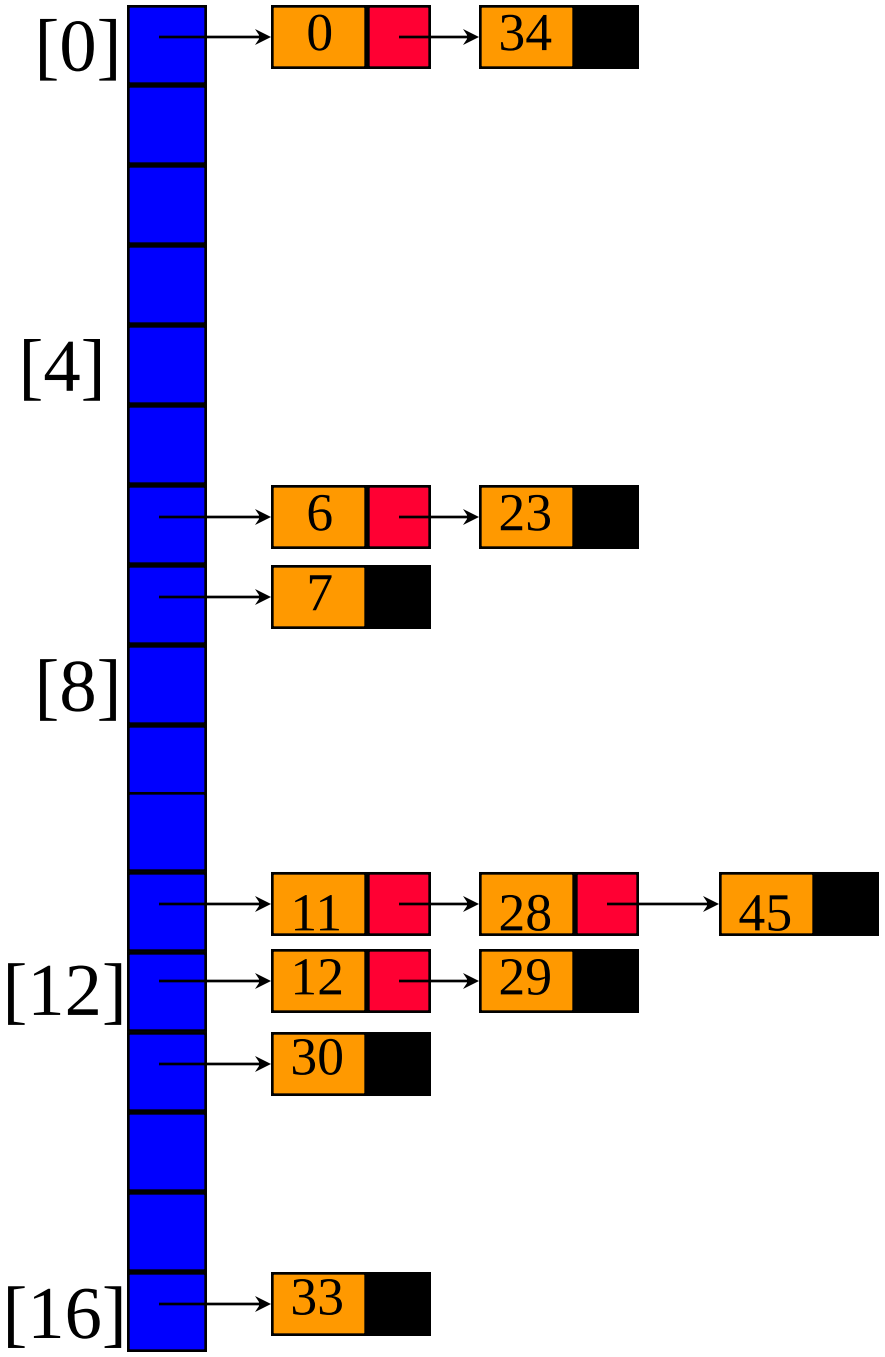
- Dynamic resizing of table.
  - Whenever loading density exceeds threshold (4/5 in our example), rehash into a table of approximately twice the current size.
- Fixed table size.
  - Know maximum number of pairs.
  - No more than 1000 pairs.
  - Loading density <= 4/5 => b >= 5/4*1000 = 1250.
  - Pick b (equal to divisor) to be a prime number or an odd number with no prime divisors smaller than 20.

Linear List Of Synonyms

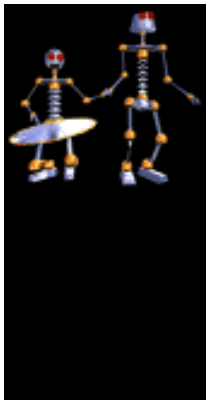
- Each bucket keeps a linear list of all pairs for which it is the home bucket.
- The linear list may or may not be sorted by key.
- The linear list may be an array linear list or a chain.

Sorted Chains

- Put in pairs whose keys are 6, 12, 34, 29, 28, 11, 23, 7, 0, 33, 30, 45
- Home bucket = key % 17.



Expected Performance



- Note that $\alpha >= 0$.
- Expected chain length is α.
- $S_n \sim 1 + \alpha/2$.
- $U_n \sim \alpha$