CSE 208 Offline 3: Hashing Report

Name: Fatima Sad Sudipta

Student ID: 2205063

This comprehensive report analyzes the performance of three hash table collision resolution methods (Separate Chaining with BST, Linear Probing, and Double Hashing) using two hash functions (Polynomial Rolling Hash and DJB2) across load factors 0.4–0.9. Key findings include:

- **Double Hashing** outperforms others at high load factors ($\alpha \geq 0.7$).
- **Linear Probing** degrades catastrophically (almost 30+ probes at α=0.9).
- **Separate Chaining** provides stable O(1) search times but with higher memory overhead.

Here's a short description of the special resolution methods:

(a) Separate Chaining using Red-Black Tree

In this method, each slot of the hash table contains a pointer to a data structure (a balanced binary tree). All elements that hash to the same index are stored in this tree. The average time complexity for both successful and unsuccessful searches is around O(1 + load factor). The tree helps keep the search time efficient even when collisions happen.

(b) Open Addressing using Double Hashing

In this approach, data is stored directly in the hash table. If a slot is already filled, a second hash function is used to calculate a step size to find the next available slot. This method spreads values more evenly and avoids clustering.

- For successful searches, the time complexity is approximately $O(1/\alpha + \ln(1 / (1 - \alpha)))$.
- For unsuccessful searches, the time complexity is around $O(1 / (1 - \alpha))$.
  Double hashing generally provides better performance at higher load factors than linear probing.

(c) Open Addressing using Linear Probing

Here, each element is also placed directly in the hash table. If a collision occurs, the index is increased by a fixed amount (usually 1) until an empty slot is found.
However, this method suffers from primary clustering, where long chains of filled slots are formed, making insertion and search slower.
The time complexity can go up to $O(1 / (1 - \alpha)^2)$ for unsuccessful searches and insertions.

Experimental Setup

- **Dataset:** 10,000 unique words
- **Hash Table Size:** Fixed for all tests (implicit via load factor)- To keep the **load factor (α)** consistent, the table size was calculated using the formula:
  table size = number of elements / load factor.
- **Metrics Tracked:**
  - Collisions during insertion
  - Average search time (before/after deletion)
  - Average probe counts
  - Uniqueness ratio (unique keys/total keys)

Hash Functions:

Here, two different hash functions were implemented:

1. Hash 1: Polynomial Rolling Function:

The Polynomial Rolling Hash function is a widely used technique for efficiently hashing strings. It treats a string as a number in a specific base and computes the hash as a polynomial:

$\text{Hash}(s) = s_0.p^0 + s_1.p^1 + s_2.p^2 + \ldots\ \ldots\ \ldots + s_n.p^n \mod m$

Where:

- S_i is the numeric value of the i-th character
- p is a chosen prime base (commonly 31 or 53)
- m is a large prime modulus to avoid overflow and collisions

This method ensures a good distribution of hash values and is especially efficient for rolling or substring hashes.

2. Hash 2: DJB2 Hash Function:

The DJB2 hash function is a simple and efficient algorithm developed by Daniel J. Bernstein. It starts with a large initial value and repeatedly multiplies the current hash by 33 and adds the ASCII value of the next character in the string. This process is repeated for each character, resulting in a hash value that is well-distributed for many types of input. DJB2 is known for its speed and good performance on small to medium-sized datasets, making it popular in many practical applications.

Collision Measurement

Collisions per insertion were counted for better comparison. This method was chosen for the following reasons:

1. **Separate Chaining (with Red-Black Trees)**
    - When collisions occur, elements are placed in a balanced binary tree.
    - Although operations on the tree have logarithmic time complexity, for large table sizes the effect becomes close to constant.
    - Counting collisions shows how often tree operations are needed.
2. **Open Addressing (Linear Probing and Double Hashing)**
    - In these methods, every collision leads to additional probing.
    - Therefore, the number of collisions directly reflects the search or insertion cost.

By using this definition, the collision handling performance of different methods can be compared more clearly and fairly.

Key Observations

1. Collision Rate Trends

- **Collisions increase with load factor**, regardless of method or hash function.
- **Linear Probing** exhibited **the highest number of collisions**, especially at higher load factors (e.g., over 50,000 collisions at load factor 0.9 using Hash1).
- **Double Hashing** significantly reduced collisions compared to Linear Probing, showing the benefit of using a secondary hash function to break clustering.

2. Probe and Time Complexity

- **Separate Chaining** maintained constant probe count (O(1)), with average search times fluctuating based on the structure of the internal BST and hash uniqueness.
- **Linear Probing** suffered from **primary clustering**, leading to sharp increases in probe count, reaching **over 36 probes** at load factor 0.9 after deletion.
- **Double Hashing** consistently kept probe counts lower, around **2–4 probes**, even at high load factors, due to its resistance to clustering.

3. Effect of Deletion

- After deletion, both **search times and probe counts** increased in open addressing methods. This is expected because probing chains cannot be terminated early during failed searches.

- **Separate Chaining** was unaffected by deletion regarding probes but showed **variation in search time**, likely due to BST restructuring.

4. Hash Function Performance

- **DJB2 consistently outperformed Polynomial Hash** in generating more unique keys across all load factors.
    - At **load factor 0.4**, DJB2 had a uniqueness ratio of **82.29%** compared to **82.24%** for Polynomial Hash.
    - At **load factor 0.9**, DJB2 maintained a slight edge (**65.94%** vs. **65.79%**).
- Despite small numerical differences, **DJB2 demonstrated more stability** in search performance post-deletion.

Observations and Conclusions from Load Factor Analysis

- **Collision Trends**
    - The number of collisions increased proportionally with the rise in load factor across all methods.
    - **Linear Probing** experienced the most rapid increase in collisions, particularly after a load factor of 0.6.
    - **Double Hashing** showed a more controlled collision rate due to the distribution advantage provided by the secondary hash function.
- **Search and Probe Efficiency**
    - **Separate Chaining** with balanced binary search trees maintained consistent performance regardless of load factor, with stable and low probe counts.
    - **Double Hashing** outperformed linear probing in terms of probing efficiency, maintaining low average probe counts even under higher load.
    - **Linear Probing** became significantly less efficient at high load factors due to **primary clustering**, which led to longer chains and increased average probe counts.
- **Effect of Deletion**
    - After deletion, both **linear probing** and **double hashing** exhibited increased average probe counts, as deleted elements forced longer probe sequences during unsuccessful searches.
    - **Separate Chaining** was not affected by deletions in terms of probe count, as deleted nodes are efficiently handled within the internal tree structure.

- **Hash Function Performance and Uniqueness Ratio**
    - The **DJB2 hash function** consistently generated a **higher uniqueness ratio** than the Polynomial Rolling Hash across all load factors.
    - For instance, at a load factor of 0.4, DJB2 achieved a uniqueness ratio of **82.29%**, compared to **82.24%** with Polynomial Hash.
    - Although the differences may appear small, even slight improvements in uniqueness led to fewer collisions and better performance, particularly in separate chaining.
    - As the load factor increased, uniqueness ratios declined for both hash functions, contributing to the rise in collision frequency.

Overall Recommendations

- For applications requiring **frequent deletions and stable search performance** == **Separate Chaining with DJB2** is the most suitable choice.
- For scenarios with **limited memory availability** where open addressing is preferred == **Double Hashing with DJB2** provides the best trade-off between collision handling and probe efficiency.
- Given its consistently **higher uniqueness ratio**, the **DJB2 hash function** is generally preferred over the polynomial rolling hash for improved distribution and reduced collision rates.

For load factor 0.4:

Method	Hash1 Function					Hash2 Function				
	# of Collisions during insertions	Before deletion		After deletion		# of Collisions during insertions	Before deletion		After deletion	
		Average search time	Average probes	Average search time	Average probes		Average search time	Average probes	Average search time	Average probes
Separate chaining with balanced BST	1783	1004.00	N/A	1239.00	N/A	1646	1373.00	N/A	1178.00	N/A
Linear probing with step adjustment	3288	585.00	1.322	0.00	2.476	3198	0.00	1.298	1000.00	2.466
Double Hashing	2033	0.00	1.262	0.00	2.072	2033	0.00	1.262	0.00	2.072

For load factor 0.5:

Method	Hash1 Function					Hash2 Function				
	# of Collisions during insertions	Before deletion		After deletion		# of Collisions during insertions	Before deletion		After deletion	
		Average search time	Average probes	Average search time	Average probes		Average search time	Average probes	Average search time	Average probes
Separate chaining with balanced BST	2092	1003.00	N/A	0.00	N/A	2167	1091.00	N/A	0.00	N/A
Linear probing with step adjustment	4959	0.00	1.504	0.00	2.965	5247	0.00	1.491	0.00	3.074
Double Hashing	2472	0.00	1.399	0.00	2.410	2472	0.00	1.399	0.00	2.410

For load factor 0.6:

Method	Hash1 Function					Hash2 Function				
	# of Collisions during insertions	Before deletion		After deletion		# of Collisions during insertions	Before deletion		After deletion	
		Average search time	Average probes	Average search time	Average probes		Average search time	Average probes	Average search time	Average probes
Separate chaining with balanced BST	2466	0.00	N/A	0.00	N/A	2446	0.00	N/A	0.00	N/A
Linear probing with step adjustment	7739	0.00	1.758	1099.00	3.845	7396	0.00	1.738	1088.00	3.732
Double Hashing	2984	0.00	1.550	0.00	2.847	2984	0.00	1.550	0.00	2.847

For load factor 0.7:

Method	Hash1 Function					Hash2 Function				
	# of Collisions during insertions	Before deletion		After deletion		# of Collisions during insertions	Before deletion		After deletion	
		Average search time	Average probes	Average search time	Average probes		Average search time	Average probes	Average search time	Average probes
Separate chaining with balanced BST	2791	0.00	N/A	1074.00	N/A	2786	0.00	N/A	0.00	N/A
Linear probing with step adjustment	11144	0.00	2.020	0.00	5.137	10580	0.00	2.138	0.00	5.716
Double Hashing	3484	1308.00	1.714	0.00	3.341	3484	1308.00	1.714	0.00	3.341

For load factor 0.8:

Method	Hash1 Function					Hash2 Function				
	# of Collisions during insertions	Before deletion		After deletion		# of Collisions during insertions	Before deletion		After deletion	
		Average search time	Average probes	Average search time	Average probes		Average search time	Average probes	Average search time	Average probes
Separate chaining with balanced BST	3044	0.00	N/A	1091.00	N/A	3138	0.00	N/A	0.00	N/A
Linear probing with step adjustment	18974	1027.00	2.692	0.00	9.814	18430	0.00	2.761	0.00	9.825
Double Hashing	3956	1000.00	1.925	1003.00	4.529	3956	1000.00	1.925	1003.00	4.529

For load factor 0.9:

Method	Hash1 Function					Hash2 Function				
	# of Collisions during insertions	Before deletion		After deletion		# of Collisions during insertions	Before deletion		After deletion	
		Average search time	Average probes	Average search time	Average probes		Average search time	Average probes	Average search time	Average probes
Separate chaining with balanced BST	3423	0.00	N/A	0.00	N/A	3477	0.00	N/A	0.00	N/A
Linear probing with step adjustment	47628	0.00	6.019	1097.00	38.555	46125	0.00	6.660	1091.00	30.086
Double Hashing	4529	0.00	2.534	1003.00	7.595	4529	0.00	2.534	1003.00	7.595