|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  | | --- | | **Perfect Hashing** | | Mohamed Ibrahim Shaapan (56) | |  | |  | |

|  |
| --- |
| **Problem Statement** |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | |  |  | | --- | --- | | **Given** | Static set of n elements  (no insertion or deletion) | | **Required** | Store them in a data structure where we can have   * Lookup in O(1) (worst-case scenario) * Construction O(polynomial) * Space O(n2) or O(n) | | |  | |

|  |
| --- |
| **O(n2) Space Solution** |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Resolve the lookup time to be O(1) in O(n2) space  Data Structure used : Perfect hash map (Single Array) | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | |  | | --- | | 1 | | 2 | | 3 | | 4 | | .. | | .. | | .. | | .. | | .. | | .. | | .. | | .. | | .. | | .. | | m | | | size = m = n2 | |  | | | **Algorithm** | | Given a set of n elements   1. Create an array of size m=n2 2. Generate a universal hash function  |  | | --- | |  | | **h(x) = ((ax+b) mod p) mod m** | | Where  m : size of the array  p : prime number larger than m  a, b : are random numbers | |  |  1. Try this hash function on all elements 2. If (number of collisions at any cell >0)   Generate new hash function and repeat   1. After obtaining perfect hash function   Start mapping your element into the data structure | |  | |

|  |
| --- |
| **O(n2) Space Solution – (Code Snippet)** |
| |  | | --- | | **public** QSPHashMap(Entry dataSet[]) {  // hash function parameters  m = dataSet.length \* dataSet.length;  p = generateNextPrime(2 \* m);  //System.out.println(m);  // initialize hash table  dataTable = **new** Entry[m];  **this**.givenDataSet = dataSet;  // construct hash table  obtainHashFunction();  storeEntries();  } |  |  | | --- | | **private** **void** obtainHashFunction() {  // 00\_try different combinations of hash functions  **boolean** hashFunctionFound = **false**;  **while** (hashFunctionFound == **false**) {  // 01\_generate new hash function  generateHashFunction();  // 02\_determine the # of collisions for this hash fun.  **int** numOfCollisions = 0;  Integer[][] testTable = **new** Integer[m][2];  //first column indicates number of collisions  //second column gives the value of the stored key  **for** (Entry element : givenDataSet) {  **int** index = runHashFunction(element.getKey());  **if**(index<0){  numOfCollisions=5;  **break**;  }  **if** (testTable[index][0] == **null**) {  testTable[index][0] = 0;  testTable[index][1] = element.getKey();  } **else** {  **if**(testTable[index][1]==element.getKey()){  **continue**;  }  numOfCollisions+=1;  **break**;  }  }  // 03\_decide if hash function is good or not  **if** (numOfCollisions < 1) {  hashFunctionFound=**true**;  }  } | |

|  |
| --- |
| **O(n) Space Solution** |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Resolve the lookup time to be O(1) in O(n2) space  Data Structure used : Perfect hash map | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | |  |  | | --- | --- | | Primary  Table | Secondary  Table | | 1 | |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | | | 2 | |  |  | | --- | --- | |  |  | | | 3 | |  |  |  | | --- | --- | --- | |  |  |  | | | 4 | |  | | --- | |  | | | .. | … | | .. | … | | .. | … | | .. | … | | .. | … | | .. | … | | .. | … | | .. | … | | .. | … | | .. | … | | n | |  |  |  | | --- | --- | --- | |  |  |  | | | |  | | | **Algorithm** | | Given a set of n elements   1. Create an PRIMARY array of size n 2. Generate a universal hash function  |  | | --- | |  | | **h(x) = ((ax+b) mod p) mod n** | | Where  n : size of the array  P : prime number larger than n  a, b : are random numbers | |  |  1. Try this hash function on all elements 2. If (largest number of collisions at any cell > c\*n )   Generate new hash function and repeat   1. After obtaining perfect hash function for primary table   Start creating secondary tables for each cell | | 1. Loop over cells in the primary table   Generate for each cell an O(n2) perfect hash table using the elements that mapped to this cell location in primary hash function  Where number of collisions at each secondary table =0   * Generate hi,j(x) unversal hash function * Try out this hash function * If (number of collisions at an cell >0)   Try new hash function   * Repeat the same step for all cells in the primary table | |

|  |
| --- |
| **O(n) Space Solution – (Code Snippet)** |
| |  | | --- | | **public** LSPHashMap(Entry dataSet[]){    // hash function parameters  m=dataSet.length;  p = generateNextPrime(2 \* m);    // initialize hash table  primaryTable=**new** QSPHashMap[m];  givenDataSet=dataSet;    // construct hash table  obtainHashFunction();  storeEntries();  } |  |  | | --- | | **private** **void** storeEntries(){    @SuppressWarnings("unchecked")  LinkedList<Entry> tmpData[]=**new** LinkedList[m];  **for**(Entry element:givenDataSet){  **int** index=runHashFunction(element.getKey());  **if**(tmpData[index]==**null**){  tmpData[index]=**new** LinkedList<Entry>();  }  tmpData[index].addLast(element);  }  **for**(**int** i=1; i<=m; i++){  LinkedList<Entry> cell=tmpData[i-1];  **if**(cell==**null**){  **continue**;  }  Entry[] cellArray=cell.toArray(**new** Entry[cell.size()]);  primaryTable[i-1]=**new** QSPHashMap(cellArray);  }    } | |

|  |
| --- |
| **Performance Analysis** |
| |  |  |  | | --- | --- | --- | |  | Constrcution  Time | Lookup Time | | O(n2) Solution | O(1) | O(1) | | O(n) Solution |  | O(1) | |