ডাটা স্ট্রাকচার – অ্যালগরিদম কনসেটপ রিভিশন –second Week (7 – 14 march)

- **Stack:**

  "stack" data structure হচ্ছে এমন এক জিনিস, যেখানে পরে আসলে আগে বের করে দেওয়া হয়।

  স্ট্যাক ডেটা স্ট্রাকচার বলতে এমনভাবে ডেটা সাজানোকে বুঝায়, যেখানে ডেটাগুলোকে স্তুপ আকারে একটার উপর একটা রাখা হয়।এবং নেওয়ার সময় প্রথমে উপর থেকে এক এক করে নেওয়া হয়। অর্থাৎ যে ডেটা পরে রাখা হবে সেটা প্রথমে বের করা হয়, যাকে বলে (LIFO) পদ্ধতি।

- **Stack er Example :** kitchen e plate sajiye rakha.

- **Code:**

```
var loveBike = [ ];

loveBike.push("Toma"); // Data insert style.
loveBike.push("Sraboni");

document.write(loveBike);

loveBike.pop(); // Data remove from top.
document.write(loveBike);
```

- **Queue:**

  আগে আসলে আগে পাবেন ফরমুলার ভিত্তিতে যে সব কাজ করা হয় তাকেই ”কিউ” বলে।
  যেমনঃ ট্রেন, বাস বা ক্রিকেট খেলার টিকিটের লাইন বা ব্যাংকে টাকা জমা দেয়ার লাইন অথবা কাস্টমার কেয়ারে সার্ভিস পাওয়া এসবই কিউ ডাটা স্ট্রাকচার এর বাস্তব উদাহরন।

- **Code:**

```
var shinnirLine = [ ];

shinnirLine.push("Jubayer");    //Data insert method
shinnirLine.push("Rahim");
shinnirLine.push("Forhad");
shinnirLine.push("Rayhan");
shinnirLine.push("Amit");
shinnirLine.push("Ekbal");

document.write(shinnirLine);
```

```
shinnirLine.shift();                //Data remove from start position.
document.write(shinnirLine);
```

- **Key_Value**:

  Dictionary Data Structure এ ডাটা store করার ক্ষেত্রে একটি Unique id থাকে ঐ id এর Under এ ডাটা insert করা এবং Data retrive করা হয়। এখানে যে Unique id ব্যবহার করা হয় তাকে Programing এর ভাষায় Key বলে এবং এর মানকে Value বলে।

- **Hash_Table:**

  Hash function ব্যবহার করে key বানিয়ে Dictionary তে key - value রাখার পর যে টেবিল তৈরি হয় তাকে Hash Table বলে।
  এই পদ্বতিকে বলা হয় HashTable Data Structure।

- **Hash_Function_from_google:**

  A hash function is any function that can be used to map data of arbitrary size to data of fixed size. The values returned by a hash function are called hash values, hash codes, digests, or simply hashes. One use is a data structure called a hash table, widely used in computer software for rapid data lookup.

- **Hash function and Dictionary topic problem:**

  স্যার, Dictionary Data Structer এবং হ্যাশ ডাটা স্ট্রাকচার এর মধ্যে পার্থক্য তেমন কি?
  আমি যতটা বুঝেছি তা হলো হ্যাশ ফাংশন যে ডাটা স্ট্রাকচারে কাজ করে সেটার নামই হলো ডিকশনারি ডাটা স্ট্রাকচার । যদি আমার বুঝটা ভুল না হয় তাহলে হ্যাশ ডাটা স্ট্রাকচার এবং ডিকশনারি ডাটা স্ট্রাকচার এর প্রশ্ন আলাদা কেন?

  আর আজকের টপিক আমি নিজে কনফিডেন্টলি বুঝেছি তা বলবো না স্যার, একটু প্রব্লেম হচ্ছে নিজের মধ্যে । 🙂 100% ক্লিয়ার মনে হয় আমার হয় নাই ।

  Thanks in Advance sir.

  Like · Reply · Mar 11, 2017 9:25pm · Edited

  Nasim Akash ·
  Studying BSc in Computer Science & Engineering_"CSE" at Jessore University of Science and Technology

  Jhankar Mahbub ·
  Sr. Web Developer at The Nielsen Company
  হোন্ডা বলতে আমরা মোটর সাইকেলকে বুঝি। আসলে– হোন্ডা হচ্ছে একটা কোম্পানির নাম। যারা মোটর সাইকেল বানায়। কিন্তু আমাদের দেশে হোন্ডা বলতে যেটা বুঝি সেটা হচ্ছে মোটরসাইকেল।

  এইবার আসি ডিকশনারি, হ্যাশ টেবিল। এই দুইটা হচ্ছে key-value এর দুইটা ব্র্যান্ড। আসল জিনিস

হচ্ছে একটা key থাকবে সেটা দিয়ে একটা value পাওয়া যাবে। যেমন হচ্ছে ক্লাসের রোল নং। সেই রোল নং দিয়ে, একজন স্টুডেন্টের নাম বের করা যায়। তারমানে রোল নং হচ্ছে key আর নামটা হচ্ছে value।

এখন এই key-valueকেই সহজ করে ডিকশনারি বলে। আর এই ডিকশনারির একটা ব্র্যান্ড হচ্ছে হ্যাশ টেবিল।

তবে মনে রাখার জন্য একটু সহজ করে মানে রাখতে পারো– হ্যাশ টেবিলে হ্যাশ নামে একটা ফাংশন থাকে যেটার কাজ হচ্ছে key-value সিস্টেমে রাখার সময় key টা কি হবে। যেমন ধর, কোন কোন ভার্সিটিতে রোল নং টা অনেক বড় হয়। যেমন ২০১৭–০৩–০১৬ যেখানে ২০১৭ সালের, ০৩ নম্বর ডিপার্টমেন্টের, ০১৬ নম্বর স্টুডেন্ট। এখন তুই যদি শুধু ওই ক্লাসের জন্য একটা key-ভ্যালু বানাতে চাস। শুধু সেই ক্লাসের জন্য। তাহলে রোল নং এর প্রথম দুই অংশ লাগবে না। সেজন্য তুই একটা ফাংশন লিখলি যেটার কাজ হবে পুরা রোল নং ইনপুট হিসেবে নেয়া। তারপর আউটপুট হিসেবে শুধু রোল নংয়ের শেষের অংশটা দিয়ে দেয়া। তারপর সেই অউটপুটকে key হিসেবে ব্যবহার করা। এই যে হ্যাশ নামে একটা ফাংশন ব্যবহার করা হইছে তাই এঁকে হ্যাশ টেবিল বলা হয়।

উপরের লেখাটা দুইবার পর। তারপরেও ক্লিয়ার না হইলে, সামনে আগাইতে থাক।

- **Algorithm:**

কোন একটা কাজ সমাধান করার জন্য ধাপে ধাপে যে নির্দেশ দরকার হয় তার সমষ্টি হচ্ছে অ্যালগরিদম।যেকোন রান্না করার জন্য যেমন কিছু রেসিপি থাকে,তেমন ই প্রব্লেম সলভ করার জন্যও অ্যালগরিদম থাকে।
কোনো একটি সমস্যা সমাধানের ধাপ গুলো কে স্টেপ বাই স্টেপ লেখাকে প্রোগ্রামিং এর ভাষায় এলগোরিদম বলা হয়।

- **Search Algorithm:**

অনেক গুলো অবজেক্ট বা বস্তু এর মধ্যে থেকে কোনো একটা নির্দিষ্ট অবজেক্ট বা বস্তু কে খুঁজে পাওয়ার জন্য যে পর্যায় ক্রমিক ধাপ সমূহ অবলম্বন করা হয় , তাকে search এলগোরিদম বলে।

- **Type of Searching:**

  এটি ২ ধরনের হতে পারে।
  ১লিনিয়ার সার্চ।.
  ২বায়নারি সার্চ।.

- **Code: Using indexOf();**

```
var books = [ "Bangla", "English", "ICT", "Physics", "Mathematics" ];
var found = books.indexOf("ICT");

if(found > -1) {
document.write("Yes, there have 'ICT' book.");
}
```

```
else {
document.write("Not found");
}
```

Another Code:

```
var books = [ "Bangla", "English", "ICT", "Physics", "Mathematics" ];

for(var i = 0; i < books.length; i++) {
var foundBook = books[i];
if(foundBook == "ICT") {
document.write("Hurrah ... ha... ha we find out 'ICT' in our book list.");
}
}
```

- **Sorting :**

  ডাটা সমূহকে ছোট থেকে বড়ো বা বড়ো থেকে ছোট ইত্যাদি বিভিন্ন অনুক্রম অনুযায়ী সাজানোর পদ্ধতিকে সর্টিং বলে।

- **Selection Sort Algorithm :**

  The **selection sort** improves on the bubble sort by making only one exchange for every pass through the list. In order to do this, a selection sort looks for the largest value as it makes a pass and, after completing the pass, places it in the proper location. As with a bubble sort, after the first pass, the largest item is in the correct place. After the second pass, the next largest is in place. This process continues and requires $n-1$ n−1 passes to sort *n* items, since the final item must be in place after the $(n-1)$ (n−1) st pass.

- **Code:**

  # free_Sorting_formula Code :

  ```
  var result = [82, 94, 88, 81, 92];
  document.write(result.sort());
  ```

  # custom_Sorting_formula:

  ```
  function fiveSubResult (result) {
          var temp;
          var len = result.length;
          var minIdx;
          for(var i = 0; i < len; i++) {
                  minIdx = i;
                  for(var j = i + 1; j < len; j++) {
                          if(result[j] < result[minIdx]) {
  ```

```
                                        minIdx = j;
                                    }
                                }
                                temp = result[i];
                                result[i] = result[minIdx];
                                result[minIdx] = temp;
                        }
                        return result;
                }

                document.write(fiveSubResult([94, 81, 88, 82, 84]));
```

- **Insertion Sort Algorithm :**

  The **insertion sort**, although still $O(n^2)$O(n2), works in a slightly different way. It always maintains a sorted sublist in the lower positions of the list. Each new item is then “inserted” back into the previous sublist such that the sorted sublist is one item larger.

  We begin by assuming that a list with one item (position $0$0) is already sorted. On each pass, one for each item 1 through $n-1$n−1, the current item is checked against those in the already sorted sublist. As we look back into the already sorted sublist, we shift those items that are greater to the right. When we reach a smaller item or the end of the sublist, the current item can be inserted.

 # Code_for_Insertion_Sort:

```
function insertionSort(cards){
        var i;
        var j;
        var temp;
        var len = cards.length;
        for(i = 0; i < len; i++){
                j = i;
                while(j > 0 && cards[j] < cards[j - 1]) {
                        temp = cards[j];
                        cards[j] = cards[j - 1];
                        cards[j - 1] = temp;
                        j--;
                }
        }
        return cards;
}
document.write(insertionSort([5, 2, 4, 10, 7]));
```