Day-3 09-09-2021

Level-Easy

Qno-1 longest common prefix, if there is no common prefix return empty string.

Input: strs = ["flower", "flow", "flight"]
output: "fl"

```
// check str = 0
// Create a string variable to store first word
   like "flower".
// run while loop for other word { flow, flight }
=> string lCPC Vector<string>& strs) {
    int m = INT_MAX;
    if (strs.size()==0)
        return "";
    string c = strs[0] // store flower
  for (int i=1; i<strs.size(); i++)
    {
      int j=0, k=0, ans=0;
      while (j<c.size() and k<strs[i].size())
      {
          if (c[j] == strs[i][k])
              ans++;
          else break;
              j++;
              k++;
      }
      m = min (m, ans);
    }
    return c.substr(0, m);
  }
};
```

level- medium

Q1 longest substring without repeating characters

Input: s = "abcabcbb"
output : 3

Brute force
// check each substring one by one
// by using for loop
// complexity $O(n^3)$

optimal soln

// Create a vector string
// two pointer approach - left, right
// move right pointer till duplicate not occur.
// find length of substring by sliding window (r-l+1).
// increase right by 1
// return len.

```
int lengthOfLongestSubstring(string s) {
vector<int>mpp(256,-1);
int left=0, right=0;
int n=s.size();
int len=0;
while(right<n)
{
if (mpp[s[right]]!=-1)
left = max(mpp[s[right]]+1, left);
mpp[s[right]]=right;
len = max(len, right-left+1);
right++;
}
```

Imp

Level - medium
Que-2 : Word Break

input : s = "leetcode", wordDict = ["leet", "code"]
output : true.

```
// using dynamic programming
// create a dp array
// unordered set string to store the value.
// use for each loop
// use for loop from right.

bool wordBreak (string s, vector <string> & wordDict) {
    int n = s.size();
    int dp [n+1];
    memset (dp, 0, sizeof dp);      // memset is used to
    dp[n] = 1;                      convert the value to unsigned
unordered_set<string>dic;           char and copies it into each
for (string &word : wordDict)       of first n char of obj.
  ~~dic.string~~
    dic.insert (word);
for (int i=n-1; i>=0; i--)
  string word;
  for (int j=i ; j<=n; j++)
  {
      word.push_back (s[j]);
  if (dic.find(word) != dic.end()) {

      if (dp[j+1])
        dp[i] = 1;
  }
}
}
  return dp[0];
}
};
```