LEARN DSA WITH C++

WEEK :: 15

DYNAMIC PROGRAMING I

LEARN DSA WITH C++

COURSE INSTRUCTOR BY
ROHIT NEGI

MADE BY-
PRADUM SINGHA

Check Profile

My profile

LEARN DSA WITH C++

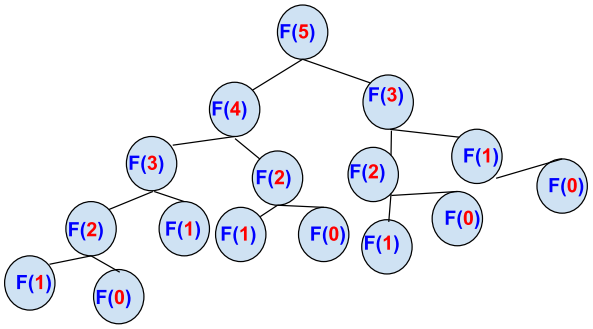
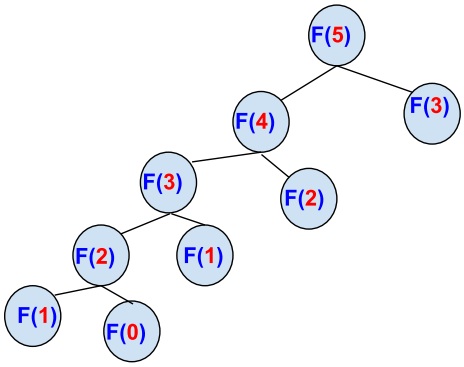
WEEK :: 15 **DAY**: 01 **DATE**: 01-08-2023

DYNAMIC PROGRAMING

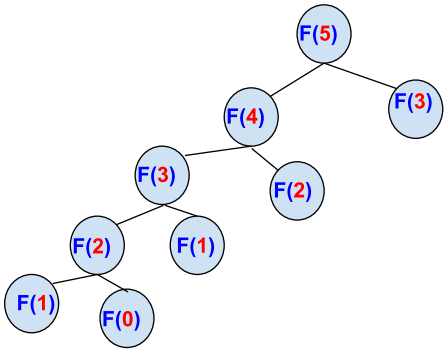
Dynamic programming is an optimization approach that transforms a complex problem into a sequence of simpler problems.

Dynamic programming	
Top down approach + Memoization	Tabulation + Bottom Up approach

Dynamic programming : No history repeat

Main Tree	Optimize Tree
	

How to Work



F(0) = 0;
F(1) = 1;

F(2) = F(1) + F(0) = 1+0=1
F(3) = F(2) + F(1) = 1+1=2
F(4) = F(3) + F(2) = 2+1=3
F(5) = F(4) + F(3) = 3+2=5

0	1	2	3	4	5
0	1	-1	-1	-1	-1
0	1	1	2	3	5

Nth Fibonacci Number

Top down approach

```
class Solution {
  public:
  int find(int n, vector<int> &dp)
```

Bottom Up approach

```
class Solution {
  public:
    int nthFibonacci(int n){
```

```
  {
      // base case
      if(n<=1)
      return n;

      // Already calculated result return
      if(dp[n] != -1)
      return dp[n];

      dp[n] = find(n-1, dp) + find(n-2,
dp);
      return dp[n];
  }
    int nthFibonacci(int n){
        // code here
        vector<int>dp(n+1, -1);
        return find(n, dp);

    }
};
```

```
        // code here
        vector<int>dp(n+1);
        dp[0]=0;
        dp[1]=1;

        for(int i=2; i<=n; i++)
        dp[i] = dp[i-1] + dp[i-2];

        return dp[n];
    }
};
```

Nth Fibonacci Number << GeeksforGeeks >

```
class Solution {
  public:
    int nthFibonacci(int n){
        // code here
        if(n<=1)
        return n;
        long long int first = 0;
        long long int second =1;
        long long int third;

        for(int i=2; i<=n; i++)
        {
            third = (first+second)%1000000007;
            first = second;
            second = third;
        }
        third %= 1000000007;
        return third;
    }
};
```

Climbing Stairs << LeetCode >>

Top Up Approach

```
class Solution
{
public:
int step(int stair, int n, vector<int> &dp)
{
    if(stair == n)
    return 1;
```

```
    if(stair>n)
    return 0;

    // Already calculated result return
    if(dp[stair] != -1)
    return dp[stair];

    dp[stair] = step(stair+1, n, dp) + step(stair+2, n, dp);
    return dp[stair];
}
    int climbStairs(int n)
    {
        vector<int>dp(n+2, -1);
        dp[n] =1, dp[n+1]=0;
        return step(0, n, dp);
    }
};
```

Bottom down approach

```
class Solution
{
public:
int step(int stair, int n, vector<int> &dp)
{
    if(stair == n)
    return 1;

    if(stair>n)
    return 0;

    // Already calculated result return
    if(dp[stair] != -1)
    return dp[stair];

    dp[stair] = step(stair+1, n, dp) + step(stair+2, n, dp);
    return dp[stair];
}
    int climbStairs(int n)
    {
        vector<int>dp(n+2);
        dp[n] =1,
        dp[n+1]=0;

        for(int i=n-1; i>=0; i--)
        dp[i]=dp[i+1] + dp[i+2];
        return dp[0];
    }
};
```

LEARN DSA WITH C++

WEEK :: 15 **DAY**: 02 **DATE**: 02-08-2023

DP : HOUSE ROBBER

House Robber << LeetCode >>

Top Down Approach

```
class Solution {
public:
int find(int index, vector<int> &nums, int n, vector<int> &dp)
{
    // Base condition
    if(index >= n)
    return 0;

    if(dp[index] != -1)
    return dp[index];

    return dp[index] = max(nums[index] + find(index+2, nums, n, dp), find(index+1, nums,
n, dp));
}
    int rob(vector<int>& nums) {
        int n= nums.size();
        vector<int>dp(n+2, -1);
        return find(0, nums, n, dp);
    }
};
```

Bottom Up Approach

```
class Solution {
public:

    int rob(vector<int>& nums) {
        int n= nums.size();
        vector<int>dp(n+2, -1);
        dp[n]=dp[n+1]=0;
        for(int i=n-1; i>-1; i--)
        dp[i] = max(nums[i] + dp[i+2], dp[i+1]);

        return dp[0];
    }
};
```

Final answer

```
class Solution {
public:

    int rob(vector<int>& nums) {
        int n= nums.size();
        int ans, first=0, second=0;
        for(int i=n-1; i>-1; i--)
```

```
        {
            ans = max(nums[i] + second, first);
            second = first;
            first = ans;
        }

        return ans;
    }
};
```

House Robber II

<< LeetCode >>

```
class Solution {
public:
int find(int index, int n, vector<int> &nums, vector<int> &dp)
{
    // Base condition
    if(index >= n)
    return 0;

    if(dp[index] != -1)
    return dp[index];

    return dp[index] = max(nums[index] + find(index+2, n, nums, dp), find(index+1, n,
nums, dp));
}
    int rob(vector<int>& nums) {
        int n= nums.size();
        if(n==1)
        return nums[0];
        vector<int>dp1(n+2, -1);
        vector<int>dp2(n+2, -1);
        return max(find(0, n-1, nums, dp1), find(1, n, nums, dp2));
    }
};
```

Bottom Up Approach

```
class Solution {
public:
    int rob(vector<int>& nums) {
        int n= nums.size();
        if(n==1)
        return nums[0];
        vector<int>dp1(n+2, -1);
        vector<int>dp2(n+2, -1);
        dp1[n-1]=dp1[n]=dp2[n]=dp2[n+1]=0;

        for(int i=n-2; i>-1; i--)
        dp1[i] = max(nums[i] + dp1[i+2], dp1[i+1]);

        for(int i=n-1; i>0; i--)
        dp2[i] = max(nums[i] + dp2[i+2], dp2[i+1]);
        return max(dp1[0],dp2[1]);
    }
};
```

Coin Change II << LeetCode >>

```
class Solution {
public:
int find(int index, int amount, vector<int> &coins, vector<vector<int>> &dp)
{
    // Base case
    if(amount == 0)
    return 1;

    if(index<0)
    return 0;

    if(dp[index][amount] != -1)
    return dp[index][amount];

    if(coins[index]>amount)
    return dp[index][amount] = find(index-1, amount, coins, dp);

    else
    return dp[index][amount] = find(index, amount-coins[index], coins, dp) +
find(index-1, amount, coins, dp);
}
    int change(int amount, vector<int>& coins) {
        int n= coins.size();
        vector<vector<int>>dp(n+1, vector<int>(amount+1, -1));
        return find(n-1, amount, coins, dp);
    }
};
```

Bottom Up Approach

```
class Solution {
public:
    int change(int amount, vector<int>& coins) {
        int n= coins.size();
        vector<vector<int>>dp(n+1, vector<int>(amount+1, 0));
        for(int i=0; i<=n; i++)
        dp[i][0] = 1;

        for(int i=1; i<= n; i++)
        for(int j=1; j<= amount; j++)
        {
            if(coins[i-1]>j)
            dp[i][j] = dp[i-1][j];
            else
            dp[i][j] = dp[i][j-coins[i-1]] + dp[i-1][j];
        };
        return dp[n][amount];
    }
};
```

Optimization Code

```
class Solution {
public:
    int change(int amount, vector<int>& coins) {
        int n= coins.size();
```

```
        vector<int>dp(amount+1, 0);
        dp[0]=1;

        for( int i=1; i<=n; i++)
        for(int j=1; j<=amount; j++)
        {
            if(coins[i-1] <= j)
            dp[j] += dp[j -coins[i-1]];
        }
        return dp[amount];
    }
};
```

LEARN DSA WITH C++

WEEK :: 15 **DAY**: 03 **DATE**: 03-08-2023

DP : KNAPSACK

Count ways to N'th Stair(Order does not matter) << GeeksforGeeks >>

Recursion { Need Optimize }

```
class Solution{
        public:
        int find(int index, int n, int step[])
        {
           // base condition
           if(n==0)
           return 1;

           if(index == 0)
           return 0;

           if(step[index-1]>n)
           return find(index-1, n, step);
           else
           return find(index, n-step[index-1], step)+find(index-1,n,step);
        }
                int nthStair(int n){
                   //  Code here
                   int step[2] = {1, 2};
                   return find(2, n, step);
```

```
        }
};
```

To Down Approach

```
class Solution{
      public:
      int find(int index, int n, int step[], vector<vector<int>> &dp)
      {
          // base condition
          if(n==0)
          return 1;

          if(index == 0)
          return 0;

          if(dp[index][n] != -1)
          return dp[index][n];

          if(step[index-1]>n)
          return dp[index][n] = find(index-1, n, step, dp);
          else
          return dp[index][n]=find(index, n-step[index-1], step,
dp)+find(index-1,n,step, dp);
      }
            int nthStair(int n){
                //  Code here
                int step[2] = {1, 2};
                vector<vector<int>>dp(3, vector<int>(n+1, -1));
                return find(2, n, step, dp);
            }
};
```

Bottom Up Approach

```
class Solution{
      public:
            int nthStair(int n){
                //  Code here
                int step[2] = {1, 2};
                vector<vector<int>>dp(3, vector<int>(n+1, 0));
                for(int i=0; i<3; i++)
                dp[i][0]=1;

                for(int i=1;i<=2; i++)
                for(int j=1; j<=n; j++)
                {
                    if(step[i-1]>j)
                    dp[i][j]= dp[i-1][j];
                    dp[i][j]=dp[i][j-step[i-1]]+ dp[i-1][j];
                }
                return dp[2][n];
            }
};
```

0 - 1 Knapsack Problem << GeeksforGeeks >>

Recursion { Need Optimize }

```
class Solution
{
   public:
   int find(int index, int W, int wt[], int val[])
   {
      // base condition
      if(W==0)
      return 0;

      if(index==0)
      return 0;

      if(wt[index-1]>W)
      return find(index-1, W, wt, val);

      else
      return max(val[index-1] + find(index-1, W-wt[index-1], wt, val), find(index-1, W, wt,val));
   }
   //Function to return max value that can be put in the knapsack of capacity W.
   int knapSack(int W, int wt[], int val[], int n)
   {
     // Your code here
     return find(n, W, wt, val);
   }
};
```

To Down Approach

```
class Solution
{
    public:
    int find(int index, int W, int wt[], int val[], vector<vector<int>> &dp)
    {
        // base condition
        if(W==0)
        return 0;

        if(index==0)
        return 0;

        if(dp[index][W] != -1)
        return dp[index][W];

        if(wt[index-1]>W)
        return dp[index][W]= find(index-1, W, wt, val, dp);
```

```
        else
        return dp[index][W]= max(val[index-1] + find(index-1, W-wt[index-1], wt,
val, dp), find(index-1, W, wt,val, dp));
    }
    //Function to return max value that can be put in the knapsack of capacity W.
    int knapSack(int W, int wt[], int val[], int n)
    {
       // Your code here
       vector<vector<int>>dp(n+1, vector<int>(W+1, -1));
       return find(n, W, wt, val, dp);
    }
};
```

Bottom Up Approach

```
class Solution
{
    public:
    //Function to return max value that can be put in the knapsack of capacity W.
    int knapSack(int W, int wt[], int val[], int n)
    {
       // Your code here
       vector<vector<int>>dp(n+1, vector<int>(W+1, 0));
       for(int i=1; i<=n; i++)
       for(int j=1; j<=W; j++)
       {
           if(wt[i-1]>j)
           dp[i][j]=dp[i-1][j];
           else
           dp[i][j]=max(val[i-1] + dp[i-1][j-wt[i-1]],dp[i-1][j]);
       }
       return dp[n][W];
    }
};
```

Optimize code

```
class Solution
{
    public:
    //Function to return max value that can be put in the knapsack of capacity W.
    int knapSack(int W, int wt[], int val[], int n)
    {
       // Your code here
       vector<int>dp(W+1, 0);
       for(int i=1; i<=n; i++)
       for(int j=W; j>0; j--)
       {
           if(wt[i-1]<=j)
           dp[j] = max(val[i-1]+dp[j-wt[i-1]], dp[j]);
       }
       return dp[W];
    }
};
```

LEARN DSA WITH C++

WEEK :: 15 **DAY**: 04 **DATE**: 04-08-2023

DP : LCS || PALINDROME

Coin Change << GeeksforGeeks >>

Recursion

```
class Solution {
public:
   int find(int N, int sum, int coins[])
   {
      // Base condition
      if (sum == 0)
         return 1;

      if (N == 0)
         return 0;

      // if coins is greater than sum, skip that coin
      if (coins[N - 1] > sum)
         return find(N - 1, sum, coins);
      else
         return find(N, sum - coins[N - 1], coins) + find(N - 1, sum, coins);
   }

   long long int count(int coins[], int N, int sum) {
      return find(N, sum, coins);
   }
};
```

To Down Approach

```
class Solution {
public:
    long long int find(int N, int sum, int coins[], vector<vector<long long int>>&
dp)
    {
        // Base condition
        if (sum == 0)
            return 1;

        if (N == 0)
            return 0;

        if (dp[N][sum] != -1)
            return dp[N][sum];
```

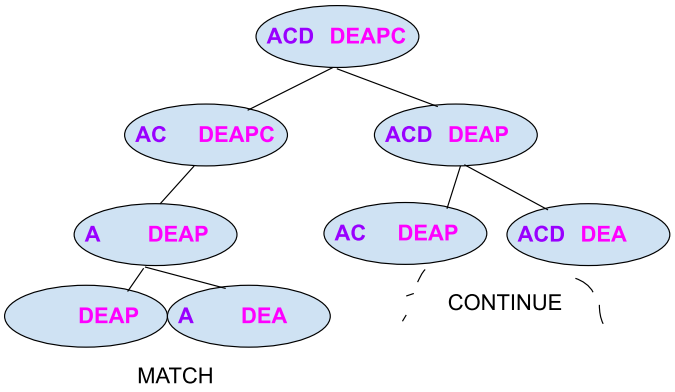
```
        // if coins is greater than sum, skip that coin
        if (coins[N - 1] > sum)
            return dp[N][sum] = find(N - 1, sum, coins, dp);
        else
            return dp[N][sum] = find(N, sum - coins[N - 1], coins, dp) + find(N - 1,
sum, coins, dp);
    }

    long long int count(int coins[], int N, int sum) {
        vector<vector<long long int>> dp(N + 1, vector<long long int>(sum + 1, -1));
        return find(N, sum, coins, dp);
    }
};
```

Bottom Up Approach

```
class Solution
{
public:
    long long int count(int coins[], int N, int sum)
    {
        vector<vector<long long int>> dp(N + 1, vector<long long int>(sum + 1, 0));

        for (int i = 0; i <= N; i++)
            dp[i][0] = 1;

        for (int i = 1; i <= N; i++) // Change 1 to N
            for (int j = 1; j <= sum; j++)
            {
                if (coins[i - 1] > j)
                    dp[i][j] = dp[i - 1][j];
                else
                    dp[i][j] = dp[i][j - coins[i - 1]] + dp[i - 1][j];
            }

        return dp[N][sum];
    }
};
```

Longest Common Subsequence LCS << GeeksforGeeks >>

str1 :- A B C D G H Commun point : A D H = 3 str2 :- A E D F H R Get continuous way Subsequence : ADH	str1 :- A C D Commun point : D =1 str2 :- D E A P C : AC =2 Get continuous way Subsequence : AC



Mach all one by one from end.
Remove which not match

```
class Solution                                  // Need optimization
{
   public:
   int find(int n,int m, string &s1, string &s2)
   {
      if(n==0 || m==0)
      return 0;

      if(s1[n-1] == s2[m-1])
      return 1+find(n-1, m-1, s1, s2);

      else
      return max(find(n-1, m, s1, s2), find(n, m-1, s1, s2));
   }
   //Function to find the length of longest common subsequence in two strings.
   int lcs(int n, int m, string s1, string s2)
   {
      // your code here
      return find(n, m, s1, s2);
   }
};
```

To Down Approach

```
class Solution
{
public:
    int find(int n, int m, string &s1, string &s2, vector<vector<int>> &dp)
    {
        if (n == 0 || m == 0)
            return 0;

        if (dp[n][m] != -1)
            return dp[n][m];

        if (s1[n - 1] == s2[m - 1])
            return dp[n][m] = 1 + find(n - 1, m - 1, s1, s2, dp);

        else
```

```
            return dp[n][m] = max(find(n - 1, m, s1, s2, dp), find(n, m - 1, s1, s2,
dp));
    }
    // Function to find the length of the longest common subsequence in two strings.
    int lcs(int n, int m, string s1, string s2)
    {
        // Initialize dp matrix with -1
        vector<vector<int>> dp(n + 1, vector<int>(m + 1, -1));
        return find(n, m, s1, s2, dp);
    }
};
```

Bottom Up Approach

```
class Solution
{
public:
    // Function to find the length of longest common subsequence in two strings.
    int lcs(int n, int m, string s1, string s2)
    {
        // Initialize dp matrix with 0
        vector<vector<int>> dp(n + 1, vector<int>(m + 1, 0));
        for(int i=1; i<=n; i++)
        for(int j=1; j<=m; j++)
        {
            if(s1[i-1] == s2[j-1])
            dp[i][j] = 1+dp[i-1][j-1];
            else
            dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
        }
```

```
class Solution
{
public:
    // Function to find the length of longest common subsequence in two strings.
    int lcs(int n, int m, string s1, string s2)
    {
        vector<int>dp(m+1, 0);
        int curr, prev;
        for(int i=1; i<=n; i++)
        {
            curr=0,prev =0;
            for(int j=1; j<=m; j++)
            {
                prev = curr;
                curr = dp[j];
                if(s1[i-1] == s2[j-1])
                dp[j] = 1+prev;
                else
                dp[j] = max(dp[j], dp[j-1]);
            }
       }

        return dp[m];
    }
};
```

Longest Palindromic Subsequence << GeeksForGeeks >>

```
class Solution{
  public:                                          // need optimization
  int find(int n, int m, string &s1, string &s2)
  {
     if(n==0 || m==0)
     return 0;

     if(s1[n-1] == s2[m-1])
     return 1+find(n-1, m-1, s1, s2);

     else
     return max(find(n-1, m, s1, s2), find(n, m-1, s1, s2));
  }

   int longestPalinSubseq(string A) {
      //code here
      string B=A;
      reverse(B.begin(), B.end());

      return find(A.size(), B.size(), A,B);
   }
};
```

Create To Down Approach Bottom Up Approach

Longest Repeating Subsequence << GeeksforGeeks >>

```
class Solution {
        public:
        int find(int n, int m, string S)
        {
           if(n==0 || m==0)
           return 0;

           if(n==m || S[n-1] != S[m-1])
           return max(find(n-1, m, S), find(n, m-1, S));

           else
           return 1+find(n-1, m-1, S);

        }
```

```
            int LongestRepeatingSubsequence(string str){
               // Code here
               int n= str.size();
               return find(n, n, str);
            }
};
```

To Down Approach

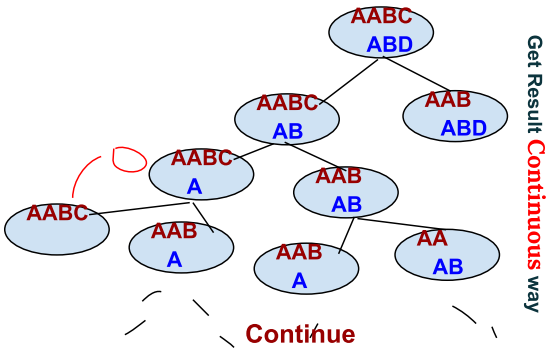
```
class Solution {
public:
    int find(int n, int m, string &S, vector<vector<int>> &dp) {
        if (n == 0 || m == 0)
            return 0;

        if (dp[n][m] != -1)
            return dp[n][m];

        if (n == m || S[n - 1] != S[m - 1])
            return dp[n][m] = max(find(n - 1, m, S, dp), find(n, m - 1, S, dp));

        else
            return dp[n][m] = 1 + find(n - 1, m - 1, S, dp);
    }

    int LongestRepeatingSubsequence(string str) {
        int n = str.size();
        vector<vector<int>> dp(n + 1, vector<int>(n + 1, -1));
        return find(n, n, str, dp);
    }
};
```

LEARN DSA WITH C++

WEEK :: 15 **DAY**: 05 **DATE**: 05-08-2023

DP : LIS || LCS || LAS

Longest Common Substring << GeeksforGeeks >>



Match one by one
Remove when no match
If match return 1 where one 0

```
//Recursion
class Solution{
  public:
  int find(int n, int m, string &s1, string &s2, int &ans)
  {
     // Base condition
     if(n==0 || m==0)
     return 0;

     int len =0;

     if(s1[n-1] == s2[m-1])
     {
       len = 1+find(n-1, m-1, s1,s2,ans);
       ans = max(ans, len);
     }
     find(n-1,m,s1,s2,ans);
     find(n,m-1,s1,s2,ans);

     return len;
  }

  int longestCommonSubstr (string S1, string S2, int n, int m)
  {
     // your code here
```

```
    int ans = 0;
    find(n, m,S1, S2, ans);

    return ans;
  }
};
```

To Down Approach

```
class Solution{
    public:
    int find(int n, int m, string &s1, string &s2, int &ans, vector<vector<int>>
&dp)
    {
        // Base condition
        if(n==0 || m==0)
        return 0;

        if(dp[n][m] != -1)
        return dp[n][m];

        int len =0;

        if(s1[n-1] == s2[m-1])
        {
           len = 1+find(n-1, m-1, s1,s2,ans, dp);
           ans = max(ans, len);
        }
        find(n-1,m,s1,s2,ans, dp);
        find(n,m-1,s1,s2,ans, dp);

        return dp[n][m] = len;
    }

    int longestCommonSubstr (string S1, string S2, int n, int m)
    {
        // your code here
        int ans = 0;
        vector<vector<int>>dp(n+1, vector<int>(m+1, -1));
        find(n, m,S1, S2, ans, dp);

        return ans;
    }
};
```

Bottom Up Approach

```
class Solution{
    public:
    int longestCommonSubstr (string S1, string S2, int n, int m)
    {
        // your code here
        int ans = 0;
        vector<vector<int>>dp(n+1, vector<int>(m+1, 0));

        for(int i=1; i<=n; i++)
        for(int j=1; j<=m; j++)
```

```
        {
            if(S1[i-1] == S2[j-1])
            {
                dp[i][j] = 1+dp[i-1][j-1];
                ans = max(dp[i][j], ans);
            }
        }

        return ans;
    }
};
```

Longest Increasing Subsequence

<< GeeksforGeeks >>

0,8,4,12,2,10,6,14,1,9,5,13,3,11,7,15

Length

1 → 0 compare with small elem

2 → 2 8 4 2 1

3 → 12 10 6 5 3

4 → 14 9 7

5 → 13 11

6 → 15

0,8,4,12,2,10,6,14,1,9,5,13,3,11,7,15

Store only small value using binary search

0	1	2	3	4	5	6	7
	0	1	3	7	11	15	

Elem = equal = put same index

= 1 lower = see index

```
class Solution
{
    public:
    //Function to find length of longest increasing subsequence.
    int longestSubsequence(int n, int a[])
    {
       // your code here
       int size =0, start, end, mid, index;
       vector<int>LIS(n);
       // 0 based indexing
       LIS[0] = a[0];
       for(int i=1; i<n; i++)
       {
           start=0, end=size;
           index = size+1;

           while(start<=end)
           {
               mid = start+(end-start)/2;

               if(LIS[mid]<a[i])
               start=mid+1;
               else if(LIS[mid] == a[i])
               {
                   index=mid;
                   break;
               }
               else
               {
                   index = mid;
                   end = mid-1;
               }
```

```
            }
            LIS[index]=a[i];
            size = max(size,index);
        }
        return size+1;
    }
};
```

Index	0	1	2	3	4	5
Elem	1	7	10	13	14	19
		(6,2)	(3,2)	(3,3)	(1,2)	(5,2)
			(9,2)	(6,3)	(4,2)	(9,3)
				(12,2)	(7,2)	(9,3)
(diff,length)					(13,2)	(12,3)
						(18,2)

```
class Solution{                                    // need optimization
public:
    int lengthOfLongestAP(int A[], int n) {
        // code here
        if(n<=2)
        return n;

        unordered_map<int,int>m[n];
        int d;
        int ans =2;
        for(int i=1; i<n; i++)
        for(int j=i-1; j>=0; j--)
        {
            d = A[i]-A[j];
            if(m[j].count(d))
            {
                m[i][d] = max(m[i][d], 1+m[j][d]);
                ans = max(ans, m[i][d]);
            }
            else
            {
                if(!m[i].count(d))
                {
                    m[i][d]=2;
                }
            }
        }
        return ans;
    }
};
```