Assign Cookies

19 October 2024 12:33

```
class Solution {
    public int findContentChildren(int[] g, int[] s)
    {
        PriorityQueue<Integer> pq1 = new PriorityQueue<>();//MIN HEAP
        PriorityQueue<Integer> pq2 = new PriorityQueue<>();//MIN HEAP
        int n=g.length;
        for(int i=0;i<n;i++)
        {
            pq1.add(g[i]);
        }
        int m=s.length;
        for(int i=0;i<m;i++)
        {
            pq2.add(s[i]);
        }int cnt=0;
        while(pq2.size()>0&&pq1.size()>0)
        {
            int a=pq1.peek();
            int b=pq2.peek();
            if(a<=b)
            {
                cnt++;
                pq1.poll();
                pq2.poll();
            }
            else
            {
                while(pq2.size()>0&&pq1.peek()>pq2.peek())
                {
                    pq2.poll();
                }
            }
        }
        return cnt;
    }
}
```

Fractional Knapsack Problem

19 October 2024 12:33

```
class Solution {
  // Function to get the maximum total value in the knapsack.

  class pair{
    int a;
    int b;

    pair(int a,int b)
    {
      this.a=a;
      this.b=b;
    }
  }

  double fractionalKnapsack(List<Integer> values, List<Integer> weight, int w)
  {


    int n=values.size();
    if(n<=0)
    {
      return 0;
    }
    pair[]arr=new pair[n];

    for(int i=0;i<n;i++)
    {
      arr[i]=(new pair(values.get(i),weight.get(i)));
    }

    //Sort the pair of array

    Arrays.sort(arr,new Comparator<pair>(){     -----> very imp
      @Override
      public int compare(pair p1,pair p2)
      {
        double r1=(double)p1.a/p1.b;
        double r2=(double)p2.a/p2.b;
        return Double.compare(r2,r1);
      }
    });

    // for(int i=0;i<n;i++)
    // {
    //     System.out.print(arr[i].a);
    //     System.out.println(arr[i].b);
    // }

    double ans=0;
    int k=n-1;
    //int w
```

```
        for(int i=0;i<n;i++)
        {
            if(arr[i].b<=w)
            {
                ans+=arr[i].a;
                w=w-arr[i].b;
                //System.out.println(w);
            }
            else
            {
                ans+=arr[i].a*((double)w/arr[i].b);
                //System.out.println(w);
                break;
            }
        }
        return ans;


    }
}
```

Greedy algorithm to find minimum number of coins***

19 October 2024 12:36

#MY CODE

```
class Solution {

    public int find(int coins[],int M,int sum,int[][]dp)
    {
        if(sum==0)
        {
            return 0;
        }
        if(M==0)
        {
            return 0;
        }
        if(dp[M][sum]!=-1)
        {
            return dp[M][sum];
        }
        if(coins[M-1]<=sum)
        {
            return dp[M][sum]=Math.min(1+find(coins,M,sum-coins[M-1],dp),find(coins,M-1,sum,dp));
        }
        else
        {
            return dp[M][sum]=find(coins,M-1,sum,dp);
        }


    }

    public int minCoins(int coins[], int M, int sum)
    {

        int n=coins.length;
        //Base condition
        if(M==0||sum==0)
        {
            return 0;
        }

        //DP initialization
        int[][]dp=new int[M+1][sum+1];
        for(int i=0;i<=M;i++)
        {
            for(int j=0;j<=sum;j++)
            {
                dp[i][j]=-1;
            }
        }

        int a=find(coins,M,sum,dp);
```

```
        //return (a==Integer.MAX_VALUE-1)?-1:a;

        return dp[M-1][sum-1];


    }
}
```

#CORRECT CODE

```
class Solution {

    // Helper function to find the minimum coins recursively
    public int find(int coins[], int M, int sum, int[][] dp) {
        // Base case when sum is 0, no coins are needed
        if (sum == 0) {
            return 0;
        }
        // If no coins are left but the sum is not zero, return a large value
        if (M == 0) {
            return Integer.MAX_VALUE - 1; // To prevent overflow in further calculations
        }
        // Check if the subproblem is already solved
        if (dp[M][sum] != -1) {
            return dp[M][sum];
        }
        // If the current coin can be included
        if (coins[M - 1] <= sum) {
            // Either include the coin or exclude it, take the minimum of both
            dp[M][sum] = Math.min(
                1 + find(coins, M, sum - coins[M - 1], dp),  // Include the current coin
                find(coins, M - 1, sum, dp)                  // Exclude the current coin
            );
        } else {
            // Exclude the coin if it can't be included
            dp[M][sum] = find(coins, M - 1, sum, dp);
        }
        return dp[M][sum];
    }

    // Main function to find minimum coins
    public int minCoins(int coins[], int M, int sum) {
        // Initialize a DP table to store intermediate results
        int[][] dp = new int[M + 1][sum + 1];
        for (int i = 0; i <= M; i++) {
            for (int j = 0; j <= sum; j++) {
                dp[i][j] = -1;
            }
        }

        // Call the helper function to solve the problem
        int result = find(coins, M, sum, dp);

        // If result is greater than sum, return -1 (no solution)
        return (result >= Integer.MAX_VALUE - 1) ? -1 : result;
```

```
    }
}
```

Lemonade Change

19 October 2024 12:36

#YOU ARE DOING GREAT BROTHER

```
class Solution {
    public boolean lemonadeChange(int[] bills)
    {
        int a=0;
        int b=0;
        int c=0;
        for(int i=0;i<bills.length;i++)
        {
            if(bills[i]==5)
            {
                a++;
            }
            else if(bills[i]==10)
            {
                if(a>0)
                {
                    a--;
                    b++;
                }
                else
                {
                    return false;
                }
            }
            else
            {
                if(b>0&&a>0)
                {
                    b--;
                    a--;
                }
                else if(b==0&&a>=3)
                {
                    a=a-3;
                }
                else
                {
                    return false;
                }
            }
        }
        return true;

    }
}
```

Valid Paranthesis Checker

19 October 2024 12:36

"(((((()*)(*)*))())())(()())())))((**)))))(()())()"
Output
true
Expected
false

N meetings in one room

19 October 2024 12:37

Jump Game

19 October 2024 12:37

Jump Game 2

19 October 2024 12:37

Minimum number of platforms required for a railway

19 October 2024 12:37

```
import java.util.Arrays;

class Solution {

    // Function to find the minimum number of platforms required
    static int findPlatform(int start[], int end[]) {

        int n = start.length;
        if (n == 0) {
            return 0;
        }

        // Sort the start and end times separately
        Arrays.sort(start);
        Arrays.sort(end);

        int platformsNeeded = 1;  // Tracks platforms required at any time
        int maxPlatforms = 1;     // Tracks the maximum number of platforms needed

        int i = 1;  // Pointer for arrival times (start[])
        int j = 0;  // Pointer for departure times (end[])

        while (i < n && j < n) {
            // If the next train arrives before the previous one departs, we need an additional platform
            if (start[i] <= end[j]) {
                platformsNeeded++;
                i++;
            }
            // If the next train departs before the next one arrives, we free a platform
            else {
                platformsNeeded--;
                j++;
            }

            // Update the maximum number of platforms needed at any point in time
            maxPlatforms = Math.max(maxPlatforms, platformsNeeded);
        }

        return maxPlatforms;
    }

    public static void main(String[] args) {
        // Example test case
        int start[] = {900, 940, 950, 1100, 1500, 1800};
        int end[] = {910, 1200, 1120, 1130, 1900, 2000};

        System.out.println("Minimum platforms required: " + findPlatform(start, end));
    }
}
```

Job sequencing Problem

19 October 2024 12:37

Candy

19 October 2024 12:37

Program for Shortest Job First (or SJF) CPU Scheduling

19 October 2024 12:37

Program for Least Recently Used (LRU) Page Replacement Algorithm

19 October 2024 12:38

Insert Interval

19 October 2024 12:38

Merge Intervals

19 October 2024 12:38

Non-overlapping Intervals

19 October 2024 12:38