Team notebook

Jalapeños

July 10, 2021

Contents

1 dp

1.1 Coin change

```
//Usar value[x] para saber el numero de monedas minimos para tener el
    cambio.
//Las monedas dadas, se pueden repetir.

value[0] = 0;
for (int x = 1; x <= n; x++) {
   value[x] = INF;
   for (auto c : coins) {
      if (x-c >= 0) {
         value[x] = min(value[x], value[x-c]+1);
      }
   }
}

//To construct the solution we can add an array first.
value[0] = 0;
for (int x = 1; x <= n; x++) {
   value[x] = INF;
   for (auto c : coins) {
      if (x-c >= 0 && value[x-c]+1 < value[x]) {
         value[x] = value[x-c]+1;
         first[x] = c;
      }
   }
}

//And used this to print.
while (n > 0) {
```

```
    cout << first[n] << "\n";
    n -= first[n];
}

//Count the number of ways.
count[0] = 1;
for (int x = 1; x <= n; x++) {
    for (auto c : coins) {
        if (x-c >= 0) {
            count[x] += count[x-c];
            //With big numbers, add this line.
            //count[x] %= m;
        }
    }
}

//Other form to solve the previous problem.
static long getNumberOfWays(long N, long[] Coins) {
    long[] ways = new long[(int)N + 1];
    ways[0] = 1;
    for (int i = 0; i < Coins.length; i++) {
        for (int j = 0; j < ways.length; j++) {
            if (Coins[i] <= j) {
                ways[j] += ways[(int)(j - Coins[i])];
            }
        }
    }
    return ways[(int)N];
}
```

1.2 EditDistance

```
public static int levenshteinDistance( String s1, String s2 ) {
    return dist( s1.toCharArray(), s2.toCharArray() );
}

public static int dist( char[] s1, char[] s2 ) {

    // memoize only previous line of distance matrix
    int[] prev = new int[ s2.length + 1 ];
    for( int j = 0; j < s2.length + 1; j++ ) {
        prev[ j ] = j;
    }

    for( int i = 1; i < s1.length + 1; i++ ) {
        // calculate current line of distance matrix
        int[] curr = new int[ s2.length + 1 ];
        curr[0] = i;

        for( int j = 1; j < s2.length + 1; j++ ) {
            int d1 = prev[ j ] + 1;
            int d2 = curr[ j - 1 ] + 1;
            int d3 = prev[ j - 1 ];
            if ( s1[ i - 1 ] != s2[ j - 1 ] ) {
                d3 += 1;
            }
            curr[ j ] = Math.min( Math.min( d1, d2 ), d3 );
        }
        // define current line of distance matrix as previous
        prev = curr;
    }
    return prev[ s2.length ];
}
```

1.3 Knapsack

```
static int max(int a, int b) {
    return Math.max(a, b);
}

static int knapSack(int W, int wt[], int val[], int n) {

    if (n == 0 || W == 0)
        return 0;

    if (wt[n - 1] > W)
        return knapSack(W, wt, val, n - 1);

    else
        return max(val[n - 1]
                       + knapSack(W - wt[n - 1], wt,
                val, n - 1),
                knapSack(W, wt, val, n - 1));
}

//Knacksack problem. To check if a number is possible with a sum of a
    subset of a 'w' array.
```

```java
//'m' is the limit of the numbers that will be checked.
possible[0][0] = true;
for (int k = 1; k <= n; k++) {
    for (int x = 0; x <= m; x++) {
        if (x-w[k] >= 0) {
            possible[x][k] |= possible[x-w[k]][k-1];
        }
        possible[x][k] |= possible[x][k-1];
    }
}

//The same but with a O(n) space complexity. The trick is starting from
    right to left in the second 'for' statement.
possible[0] = true;
for (int k = 1; k <= n; k++) {
    for (int x = m-w[k]; x >= 0; x--) {
        possible[x+w[k]] |= possible[x];
    }
}
```

1.4 Lis

```java
// This algorithm has O(n^2) complexity, there are another with
    O(nlog(n)).
private static int lis(int[] array) {
    int[] dp = new int[array.length];
    int max = 1;
    for (int i = 0; i < dp.length; i++) {
        int dpi = 1;
        for (int j = 0; j < i; j++) {
            if (array[j] < array[i] && dpi < 1 + dp[j]) {
                dpi = 1 + dp[j];
                max = Math.max(dpi, max);
            }
        }
        dp[i] = dpi;
    }
    return max;
}
```

1.5 LongestCommonSubsequence

```java
public int longestCommonSubsequenceLength(String str1, String str2) {
    int[][] memo = new int[str2.length() + 1][str1.length() + 1];
    for (int str2Row = 0; str2Row <= str2.length(); str2Row++) {
        for (int str1Col = 0; str1Col <= str1.length(); str1Col++) {
            if (str2Row == 0 || str1Col == 0) {
                memo[str2Row][str1Col] = 0;
            } else if (str2.charAt(str2Row - 1) == str1.charAt(str1Col -
                1)) {
                memo[str2Row][str1Col] = memo[str2Row - 1][str1Col - 1] +
                    1;
            } else {
                memo[str2Row][str1Col] = Math.max(
                        memo[str2Row - 1][str1Col], memo[str2Row][str1Col -
                            1]
                );
            }
        }
    }
    return memo[str2.length()][str1.length()];
}
```

1.6 Max Path in a Grid 2D

```java
// Calculate the max path in a Matrix only move right of down.
int[][] max = new int[n + 1][n + 1];
int[][] value = new int[n + 1][n + 1];

for (int y = 1; y <= n; y++) {
    for (int x = 1; x <= n; x++) {
        sum[y][x] = max(sum[y][x-1],sum[y-1][x])+value[y][x];
    }
}

// The answer is sum[n][n]
```

1.7 MaxSum1D

```java
public int maxSum1D(int[] nums) {
    int maxSoFar = nums[0];
    int maxEndingHere = nums[0];
```

```
    for (int i = 1; i < nums.length; i++) {
        maxEndingHere = Math.max(maxEndingHere + nums[i], nums[i]);
        maxSoFar = Math.max(maxSoFar, maxEndingHere);
    }
    return maxSoFar;
}
```

1.8 Maxsum2D

```
public static int maxSum2D(int[][] n) {
    int maxSum = Integer.MIN_VALUE;
    int currentSum = maxSum;
    for (int i = 0; i < n.length; i++) {
        int[] a = new int[n[0].length];
        for (int j = i; j < n.length; j++) {
            sum(a, n[j]);
            //maxSum is the MaxSum1D algorithm.
            currentSum = maxSum1D(a);
            maxSum = Math.max(currentSum, maxSum);
        }
    }
    return maxSum;
}

private static void sum(int[] a, int[] ints) {
    for (int i = 0; i < a.length; i++) {
        a[i] += ints[i];
    }
}
```

2 graph

2.1 Bellmanford

```
public void bellmanford(List<Edge> edgeList, int start) {
    int INF = Integer.MAX_VALUE;
    Map<Integer, Integer> distance = new HashMap<>();
    edgeList.forEach(x -> {
        distance.put(x.a, INF);
        distance.put(x.b, INF);
    });
    distance.put(start, 0);
    for (int i = 0; i < edgeList.size() - 1; i++) {
        for (Edge edge : edgeList) {
            int a = edge.a;
            int b = edge.b;
            int w = edge.w;
            int min = Math.min(distance.get(b), distance.get(a) + w);
            distance.put(b, min);
        }
    }
}

private class Edge {
    int a;
    int b;
    int w;
}
```

2.2 Bfs

```
public static boolean bfs(int first, int end, LinkedList<Integer>[]
     graph) {
    Queue<Integer> queue = new LinkedList<>();
    Set<Integer> visited = new HashSet<>();
    queue.add(first);
    visited.add(first);
    while (queue.size() > 0) {
        int element = queue.poll();
        if (element == end) {
            return true;
        }
        LinkedList<Integer> adj = graph[element];
        adj.forEach(a -> {
            if (!visited.contains(a)) {
                visited.add(a);
                queue.add(a);
            }
        });
    }
    return false;
}
```

2.3 DepthFirstSearch

```
static final int MAX = 100005;
static ArrayList<Integer>[] adyList = new ArrayList[MAX];
static boolean[] marked = new boolean[MAX];
static int nodes;

static void dfs(int n) {
    for(int i = 0; i <= nodes; i++) {
        adyList[i] = new ArrayList<>();
        marked[i] = false;
    }
    marked[n] = true;
    for (int v : adyList[n]) {
        if (!marked[v]) dfs(v);
    }
}
```

2.4 Dijkstra

```
static long INF = (1L << 62);
static final int MAX = 100005;
static ArrayList<edge>[] g = new ArrayList[MAX];
static boolean[] vis = new boolean[MAX];
static int[] pre = new int[MAX];
static long[] dist = new long[MAX];
static int N, M;


static class edge implements Comparable<edge>{
    int v;
    long w;

    edge(int _v, long _w){
        v = _v;
        w = _w;
    }

    @Override
    public int compareTo(edge o) {
        if(w > o.w)return 1;
        else return -1;
    }
}

static void dijkstra(int u) {
    PriorityQueue<edge> pq = new PriorityQueue<>();
    pq.add(new edge(u, 0));
    dist[u] = 0;

    while (!pq.isEmpty()) {
        u = pq.poll().v;
        if (!vis[u]) {
            vis[u] = true;
            for (edge nx : g[u]) {
                int v = nx.v;
                if(!vis[v] && dist[v] > dist[u] + nx.w) {
                    dist[v] = dist[u] + nx.w;
                    pre[v] = u;
                    pq.add(new edge(v, dist[v]));
                }
            }
        }
    }
}

static void init() {
    for(int i = 0; i <= N; i++) {
        g[i] = new ArrayList<>();
        dist[i] = INF;
        vis[i] = false;
    }
}
```

3 math

3.1 CatalanNumbers

```
//Guarda en el array Catalan Numbers los numeros de Catalan hasta MAX.
static int MAX = 30;
static long catalan[] = new long[MAX+1];
static void catalanNumbers(){
    catalan[0] = 1;
    for(int i = 1; i <= MAX; i++){
```

```
        catalan[i] = (long)(catalan[i-1]*((double)(2*((2 * i)- 1))/(i +
            1)));
    }
}
```

3.2 Combinatorics

```
//Calcula el coeficiente binomial nCr, entendido como el nmero de
    subconjuntos de k elementos escogidos de un conjunto con n elementos.
//Tambien se le conoce como (n)
//                      (r)
static long ncr(long n, long r) {
    if (r < 0 || n < r) return 0;
    r = Math.min(r, n - r);
    long ans = 1;
    for (int i = 1; i <= r; i++) {
        ans = ans * (n - i + 1) / i;
    }
    return ans;
}
```

3.3 EratosthenesSieve

```
//Guarda en primes los nmeros primos menores o iguales a MAX.
//Se puede usar la lista para recorrer los primos.
//O el marked para saber si un numero es primo.
static int MAX = 1000000;
static int SQRT = 1000;
static ArrayList<Integer> primes = new ArrayList<>();
static boolean marked[] = new boolean[MAX+1];

static void sieve() {
    marked[1] = true;
    int i = 2;
    for (; i <= SQRT; ++i) if (!marked[i]) {
        primes.add(i);
        for (int j = i*i; j <= MAX; j += i) marked[j] = true;
    }
    for (; i <= MAX; ++i) if (!marked[i]) primes.add(i);
}
```

3.4 GreatestCommonDivisor

```
//Calcula el mximo  comn  divisor entre a y b mediante el algoritmo de
    Euclides
public static int gcd(int a, int b) {
    if (b == 0) {
        return a;
    }
    return gcd(b, a % b);
}
```

3.5 LowestCommonMultiple

```
//Calculo del mnimo  comn  mltiplo  usando el mximo  comn  divisor.
public static int lcm(int a, int b) {
    return a * b / gcd(a, b);
}
```

3.6 ModularExponation

```
//Realiza la operacin (a ^ b) % mod.
static long modpow( long a, long b, long mod) {
    if (b == 0) return 1;
    if (b % 2 == 0) {
        long temp = modpow(a, b/2, mod);
        return (temp * temp) % mod;
    } else {
        long temp = modpow(a, b-1, mod);
        return (temp * a) % mod;
    }
}

//This is a shorter one.
//(x^n) % mod
int modpow(int x, int n, int m) {
    if (n == 0) return 1%m;
    long u = modpow(x,n/2,m);
    u = (u*u)%m;
    if (n%2 == 1) u = (u*x)%m;
    return u;
}
```

4 searching

4.1 BinarySearch

```
//Devuelve posicion, si no hay entonces -1
static int binary_search(int array[], int x){
    int l = 0, r = arr.length-1;
    while (l <= r) {
        int m = (l+r)/2;
        if(array[m] < x) l = m+1;
        else if (array[m] > x) r = m-1;
        else return m;
    }
    return -1;
}
```

5 sorting

5.1 MergeSort

```
private static void mergesort(int[] array, int[] temp, int start, int
     end) {
    if (start >= end)
        return;
    int mid = (start + end) / 2;
    mergesort(array, temp, start, mid);
    mergesort(array, temp,mid + 1, end);
    merge(array, temp, start, end);
}

private static void merge(int[] array, int[]temp, int leftStart, int
     rightEnd) {
    int leftEnd = (leftStart + rightEnd) / 2;
    int rightStart = leftEnd + 1;
    int size = rightEnd - leftStart + 1;

    int left = leftStart;
    int right = rightStart;
    int index = leftStart;
    while (left <= leftEnd && right <= rightEnd) {
        if (array[left] <= array[right]) {
            temp[index] = array[left++];
        } else {
            temp[index] = array[right++];
        }
        index++;
    }
    System.arraycopy(array, left, temp, index, leftEnd - left + 1);
    System.arraycopy(array, right, temp, index, rightEnd - right + 1);
    System.arraycopy(temp, leftStart, array, leftStart, size);
}
```