1. Two sum (songs/ truck) （选出最大值pair的index，会有重复，return含有最大长度的那一组）

/\*

Given int array of song list

return a pair of song which total songs time will end exactly 30 minute before

given ride duration. if 2 pairs with same duration get the longest song

For example:

Ride Duration:90

Songs: (1,10,25,35,60,20,40)

Output:

[20,40] as 40 has longer song

\*/

class MusicPlay {

// time: O(n)

public List<Integer> getSongs(int[] songs, int ride) {

List<Integer> res = new ArrayList<>();

Map<Integer, Integer> map = new HashMap<>();

int songOne = -1;

int songTwo = -1;

int longSong = -1;

for (int i = 0; i < songs.length; i++) {

if (map.containsKey(ride - 30 - songs[i])) {

if (longSong < Math.max(songs[i], ride - 30 - songs[i])) {

songOne = i;

songTwo = map.get(ride - 30 - songs[i]);

longSong = Math.max(songs[i], ride - 30 - songs[i]);

}

}

map.put(songs[i], i);

}

res.add(songOne);

res.add(songTwo);

return res;

}

public static void main(String[] args) {

MusicPlay sol = new MusicPlay();

int ride = 90;

int[] songs = new int[] {1, 10, 25, 35, 60, 20, 40};

List<Integer> res = new ArrayList<>();

res = sol.getSongs(songs, ride);

for (Integer k : res) {

System.out.println(k);

}

}

}

1. Reorder log (leetcode 937)
2. 装零件 (merger stones, [1 1 2] -> [2 2] -> [4])

/\*

\* if needs to merger consecutive K stones every time, use this method

\* we have a 2D matrix dp, dp[i][j] means merge stone\_i to stone\_j into one pile;

\* m means the number of stones to merge, we first calculate from K stones, then to len stones,

\* then for dp[i][j] = Math.min(dp[i][mid] + dp[mid+1][j]), i <= mid < j

\*/

// time: O(n^3)

public int mergeStones(int[] stones, int K) {

if (stones == null || stones.length == 0) {

return 0;

}

int len = stones.length;

if ((len - 1) % (K - 1) > 0) {

return -1;

}

int[] prefix = new int[len + 1];

for (int i = 0; i < len; i++) {

prefix[i + 1] = prefix[i] + stones[i];

}

int[][] dp = new int[len][len];

for (int m = K; m <= len; m++) {

for (int i = 0; i + m <= len; i++) {

int j = i + m - 1;

dp[i][j] = Integer.MAX\_VALUE;

for (int mid = i; mid < j; mid += K - 1) {

dp[i][j] = Math.min(dp[i][j], dp[i][mid] + dp[mid + 1][j]);

}

if ((j - i) % (K - 1) == 0) {

dp[i][j] += prefix[j + 1] - prefix[i];

}

}

}

return dp[0][len - 1];

}

/\*

\* if just to choose any two of stones to merge, use minheap

\*/

public int merge(List<Integer> list) {

if (list == null || list.size() == 0) {

return 0;

}

if (list.size() == 1) {

return list.get(0);

}

int res = 0;

PriorityQueue<Integer> minHeap = new PriorityQueue<>();

for (Integer n : list) {

minHeap.offer(n);

}

while (minHeap.size() >= 2) {

int first = minHeap.poll();

int second = minHeap.poll();

res += (first + second);

minHeap.offer(first + second);

}

return res;

}

1. Two sum closest (foreApp and backApp / air route)

/\* two sum closest  
     \* 给两个int的list，一个capacity，从两个list中各选一个item把num加和，返回所有pair里小于capacity并且最大的id，有多个就返回多个  
     \* list是这个格式的：[[id1, num1], [id2, num2], [id3, num3]]  
     \* 返回的格式是：[[id, id], ...] \*/

public class Main {

public List<List<Integer>> utilization(int[][] fore, int[][] back, int capacity) {

TreeMap<Integer, List<Integer>> tree = new TreeMap<>();

for (int[] pair : back) {

List<Integer> list = tree.getOrDefault(pair[1], new ArrayList<>());

list.add(pair[0]);

tree.put(pair[1], list);

}

TreeMap<Integer, List<List<Integer>>> res = new TreeMap<>();

for (int[] pair : fore) {

Integer floorKey = tree.floorKey(capacity - pair[1]);

if (floorKey != null) {

int diff = capacity - pair[1] - floorKey;

List<List<Integer>> list = res.getOrDefault(diff, new ArrayList<>());

for (int id : tree.get(floorKey)) {

List<Integer> match = new ArrayList<>();

match.add(pair[0]);

match.add(id);

list.add(match);

}

res.put(diff, list);

}

}

return res.get(res.firstKey());

}

public static void main(String[] args) {

Main sol = new Main();

int capacity = 10000;

int[][] fore = new int[][] {{1,7000}, {2,5000}, {3,3000}, {4,10000}};

int[][] back = new int[][] {{1,2000}, {2,4000}, {3,3000}, {4,5000}};

List<List<Integer>> res = new ArrayList<>();

res = sol.utilization(fore, back, capacity);

for (List<Integer> li : res) {

System.out.println(li);

}

}

}

1. Shortest path, Maze (2d array) (Robert / remove obstacle)

/\*

[[1, 0, 0],  
  [1, 0, 0],  
  [1, 9 ,0]].-baidu 1point3acres  
outout: 3 [0, 0] -> [1, 0] -> [2, 0] -> [2, 1]

\*/

// time: O(mn)

public int minDistance(int numR, int numC, int[][] area) {

int[][] dirs = new int[][] {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};

int res = 0;

boolean[][] visited = new boolean[numR][numC];

Queue<int[]> queue = new LinkedList<>();

queue.offer(new int[] {0, 0});

visited[0][0] = true;

while (!queue.isEmpty()) {

int size = queue.size();

for (int i = 0; i < size; i++) {

int[] cur = queue.poll();

if (area[cur[0]][cur[1]] == 9) {

return res;

}

for (int[] dir : dirs) {

int x = cur[0] + dir[0];

int y = cur[1] + dir[1];

if (x >= 0 && x < numR && y >= 0 && y < numC && !visited[x][y]

&& area[x][y] != 0) {

queue.offer(new int[] {x, y});

visited[x][y] = true;

}

}

}

res++;

}

return -1;

}

public static void main(String[] args) {

Main sol = new Main();

int[][] area = new int[][] {{1, 0, 0}, {1, 0, 0}, {1, 9, 0}};

int res = sol.minDistance(3, 3, area);

System.out.println(res);

}

}

1. K closest points to origin (amzon fresh)

public class Main {

/\*

\* numDest: 3

\* allLocations: {{1,2}, {3,4}, {1,-1}}

\* numDeli: 2

\* return: {{1,2}, {1,-1}}

\*/

public List<List<Integer>> closestXDestination(int numDest, List<List<Integer>> allLocations, int numDeli) {

if (allLocations == null || allLocations.size() == 0 || allLocations.size() < numDeli) {

return new ArrayList<>();

}

sort(allLocations, 0, numDest - 1, numDeli);

return allLocations.subList(0, numDeli);

}

private void sort(List<List<Integer>> allLocations, int left, int right, int numDeli) {

if (left >= right) {

return;

}

int pivotPos = partition(allLocations, left, right);

int leftLen = pivotPos - left + 1;

if (numDeli < leftLen) {

sort(allLocations, left, pivotPos - 1, numDeli);

} else if (numDeli > leftLen) {

sort(allLocations, pivotPos + 1, right, numDeli - leftLen);

}

}

private int partition(List<List<Integer>> allLocations, int left, int right) {

int pivotIndex = left + (int)(Math.random() \* (right - left + 1));

int pivot = distance(allLocations.get(pivotIndex));

swap(pivotIndex, right, allLocations);

int i = left;

int j = right - 1;

while (i <= j) {

if (distance(allLocations.get(i)) < pivot) {

i++;

} else if (distance(allLocations.get(j)) >= pivot) {

j--;

} else {

swap(i++, j--, allLocations);

}

}

swap(i, right, allLocations);

return i;

}

private void swap(int i, int j, List<List<Integer>> allLocations) {

int t1 = allLocations.get(i).get(0);

int t2 = allLocations.get(i).get(1);

allLocations.get(i).set(0, allLocations.get(j).get(0));

allLocations.get(i).set(1, allLocations.get(j).get(1));

allLocations.get(j).set(0, t1);

allLocations.get(j).set(1, t2);

}

private int distance(List<Integer> deli) {

return (int)(Math.pow(deli.get(0), 2) + Math.pow(deli.get(1), 2));

}

}

1. MST

public int MST(int numCities, int numRoads, List<List<Integer>> oldRoads,

List<List<Integer>> newRoads) {

if (newRoads == null || newRoads.size() == 0) {

return 0;

}

PriorityQueue<List<Integer>> pq = new PriorityQueue<>(11, new Comparator<List<Integer>>() {

@Override

public int compare(List<Integer> l1, List<Integer> l2) {

return l1.get(2) - l2.get(2);

}

});

for (List<Integer> l : newRoads) {

pq.add(l);

}

DSU uf = new DSU(numCities);

Set<String> set = new HashSet<>();

for (List<Integer> r : newRoads) {

set.add(r.get(0) + "," + r.get(1));

}

int count = 0;

for (List<Integer> r : oldRoads) {

String key = r.get(0) + "," + r.get(1);

if (set.contains(key)) {

continue;

}

int index1 = r.get(0);

int index2 = r.get(1);

if (uf.union(index1, index2)) {

count++;

}

}

if (count >= numCities - 1) {

return 0;

}

int sum = 0;

while (!pq.isEmpty() && count < numCities - 1) {

List<Integer> cur = pq.poll();

int city1 = cur.get(0);

int city2 = cur.get(1);

if (uf.union(city1, city2)) {

count++;

sum += cur.get(2);

}

}

if (count < numCities - 1) {

return -1;

}

return sum;

}

class DSU {

int[] parent;

int[] rank;

public DSU(int size) {

parent = new int[size + 1];

for (int i = 0; i <= size; i++) {

parent[i] = i;

}

rank = new int[size + 1];

}

public int find(int x) {

if (parent[x] != x) {

parent[x] = find(parent[x]);

}

return parent[x];

}

public boolean union(int x, int y) {

int xr = find(x);

int yr = find(y);

if (xr == yr) {

return false;

} else if (rank[xr] < rank[yr]) {

parent[xr] = yr;

} else if (rank[xr] > rank[yr]) {

parent[yr] = xr;

} else {

parent[yr] = xr;

rank[xr]++;

}

return true;

}

}

1. Partition label

/\*

\* Input: S = "ababcbacadefegdehijhklij"

\* Output: [9,7,8]

\* Explanation:

\* The partition is "ababcbaca", "defegde", "hijhklij".

\* This is a partition so that each letter appears in at most one part.

\* A partition like "ababcbacadefegde", "hijhklij" is incorrect, because it splits S into less parts.

\*/

// time: O(n)

public List<Integer> partitionLabels(String S) {

List<Integer> res = new ArrayList<>();

int[] last = new int[26];

for (int i = 0; i < S.length(); i++) {

last[S.charAt(i) - 'a'] = i;

}

int start = 0;

int end = 0;

for (int i = 0; i < S.length(); i++) {

end = Math.max(end, last[S.charAt(i) - 'a']);

if (i == end) {

res.add(end - start + 1);

start = i + 1;

}

}

return res;

}