NAME: J.Sravani

REG NO: 192372141

Question no – 1

GRAPH COLORING

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

int minColors(int n, int adj[][n], int color[]) {

int maxColor = 0;

for (int u = 0; u < n; u++) {

bool available[n];

for (int i = 0; i < n; i++)

available[i] = true;

for (int v = 0; v < n; v++) {

if (adj[u][v] && color[v] != -1)

available[color[v]] = false;

}

int cr;

for (cr = 0; cr < n; cr++) {

if (available[cr])

break;

}

color[u] = cr;

if (cr + 1 > maxColor)

maxColor = cr + 1;

}

return maxColor;

}

int maximizeYourRegions(int n, int adj[][n]) {

int color[n];

for (int i = 0; i < n; i++)

color[i] = -1;

int maxColor = minColors(n, adj, color);

int yourRegions = 0;

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

if (color[i] == 0) // Assume you start coloring with color 0

yourRegions++;

}

return yourRegions;

}

int main() {

int n;

printf("Enter the number of regions: ");

scanf("%d", &n);

int adj[n][n];

printf("Enter the adjacency matrix:\n");

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

for (int j = 0; j < n; j++) {

scanf("%d", &adj[i][j]);

}

}

int result = maximizeYourRegions(n, adj);

printf("Maximum number of regions you can color: %d\n", result);

return 0;

}

INPUT :

No.of regions : 4

No. of adjacency matrix :

1 1 0 1

0 1 1 0

1 1 0 1

0 1 1 0

OUTPUT:

Maximum no. of regions you can color : 2

Question no – 2

SORTING ARRAY

#include <stdio.h>

void findMinAndMax(int arr[], int n, int \*min, int \*max) {

\*min = arr[0];

\*max = arr[0];

for (int i = 1; i < n; i++) {

if (arr[i] < \*min) {

\*min = arr[i];

}

if (arr[i] > \*max) {

\*max = arr[i];

}

}

}

int main() {

int arr1[] = {2,4,6,8,10,12,13,18,};

int n1 = sizeof(arr1) / sizeof(arr1[0]);

int min1, max1;

findMinAndMax(arr1, n1, &min1, &max1);

printf("Test Case 1: Min = %d, Max = %d\n", min1, max1);

int arr2[] = {11,13,15,17,19,21,23,35,37};

int n2 = sizeof(arr2) / sizeof(arr2[0]);

int min2, max2;

findMinAndMax(arr2, n2, &min2, &max2);

printf("Test Case 2: Min = %d, Max = %d\n", min2, max2)

int arr3[] = {22,34,35,36,43,67,12,13,15,17};

int n3 = sizeof(arr3) / sizeof(arr3[0]);

int min3, max3;

findMinAndMax(arr3, n3, &min3, &max3);

printf("Test Case 3: Min = %d, Max = %d\n", min3, max3);

return 0;

}

Input:

arr1[] = {2,4,6,8,10,12,13,18,};

arr2[] = {11,13,15,17,19,21,23,35,37};

arr3[] = {22,34,35,36,43,67,12,13,15,17};

Output:

Min 1=2

Max1=18

Min 2=11

Max2=37

Min3=12

Max=67

Question no 3

HOUSE ROBBERY

#include <stdio.h>

int max(int a, int b) {

return a > b ? a : b;

}

int rob(int\* nums, int numsSize) {

if (numsSize == 0) {

return 0;

}

if (numsSize == 1) {

return nums[0];

}

int dp[numsSize];

dp[0] = nums[0];

dp[1] = max(nums[0], nums[1]);

for (int i = 2; i < numsSize - 1; i++) {

dp[i] = max(nums[i] + dp[i - 2], dp[i - 1]);

}

int maxMoney = dp[numsSize - 2];

dp[0] = 0;

dp[1] = nums[1];

for (int i = 2; i < numsSize; i++) {

dp[i] = max(nums[i] + dp[i - 2], dp[i - 1]);

}

maxMoney = max(maxMoney, dp[numsSize - 1]);

return maxMoney;

}

int main() {

int nums1[] = {2, 3, 2};

int nums2[] = {1, 2, 3, 1};

int result1 = rob(nums1, 3);

int result2 = rob(nums2, 4)

printf("Maximum money without alerting the police (nums1): %d\n", result1);

printf("Maximum money without alerting the police (nums2): %d\n", result2);

return 0;

}

Input:

nums1[] = {2, 3, 2};

nums2[] = {1, 2, 3, 1};

Output:

Maximum money without alerting the police (nums1): 3

Maximum money without alerting the police (nums2):4

Question No 4

SINGLE SOURCE SHORTEST PATH-(DIJIKSTRS ALGORITHM)

#include <stdio.h>

#define INF 9999

#define MAX 10

void DijkstraAlgorithm(int Graph[MAX][MAX], int src, int V) {

int visited[V];

int distance[V];

int u;

int i, j;

for (i = 0; i < V; i++) {

visited[i] = 0;

distance[i] = INF;

}

distance[src] = 0;

for (i = 0; i < V - 1; i++) {

u = -1;

for (j = 0; j < V; j++) {

if ((visited[j] == 0) && (u == -1 || distance[j] < distance[u]))

u = j;

}

visited[u] = 1;

for (j = 0; j < V; j++) {

if ((Graph[u][j] != 0) && (visited[j] == 0)) {

if (distance[u] + Graph[u][j] < distance[j])

distance[j] = distance[u] + Graph[u][j];

}

}

}

printf("Distance from the Source Node to %d: %d\n", src + 1, distance[src]);

for (i = 0; i < V; i++) {

printf("Distance from the Source Node to %d: %d\n", i + 1, distance[i]);

}

}

int main() {

int Graph[MAX][MAX] = {

{0, 9, 75, 0, 0},

{9, 0, 95, 19, 42},

{75, 95, 0, 51, 66},

{0, 19, 51, 0, 31},

{0, 42, 66, 31, 0}};

int V = 9;

int src = 0;

DijkstraAlgorithm(Graph, src, V);

return 0;

}

Input:

{0, 9, 75, 0, 0},

{9, 0, 95, 19, 42},

{75, 95, 0, 51, 66},

{0, 19, 51, 0, 31},

{0, 42, 66, 31, 0}

Output:

Distance from the Source Node to 1: 0

Distance from the Source Node to 1: 0

Distance from the Source Node to 2: 9

Distance from the Source Node to 3: 75

Distance from the Source Node to 4: 28

Distance from the Source Node to 5: 51

Qustion no 5

SELECTION SORT

#include <stdio.h>

void swap(int \*a, int \*b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void selectionSort(int array[], int size) {

for (int step = 0; step < size - 1; step++) {

int min\_idx = step;

for (int i = step + 1; i < size; i++) {

if (array[i] < array[min\_idx])

min\_idx = i;

}

swap(&array[min\_idx], &array[step]);

}

}

void printArray(int array[], int size) {

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

printf("%d ", array[i]);

}

printf("\n");

}

int main() {

int data[] = {20, 12, 10, 15, 2};

int size = sizeof(data) / sizeof(data[0]);

selectionSort(data, size);

printf("Sorted array in Acsending Order:\n");

printArray(data, size);

}

Input:

data[] = {20, 12, 10, 15, 2}

Output:

Sorted array in Acsending Order:

2 10 12 15 20

Question No 6

SEQUENTIAL SEARCH

#include <stdio.h>

int findKthPositive(int\* arr, int arrSize, int k) {

int missingCount = 0;

int current = 1;

int i = 0;

while (missingCount < k) {

if (i < arrSize && arr[i] == current) {

i++;

} else {

missingCount++;

}

if (missingCount == k) {

return current;

}

current++;

}

return -1;

}

int main() {

int arr1[] = {2, 3, 4, 7, 11};

int k1 = 5;

int arrSize1 = sizeof(arr1) / sizeof(arr1[0]);

int result1 = findKthPositive(arr1, arrSize1, k1);

if (result1 != -1) {

printf("Test Case 1: The %dth missing positive integer is: %d\n", k1, result1);

} else {

printf("Test Case 1: The %dth missing positive integer is not found.\n", k1);

}

int arr2[] = {1, 2, 3, 4};

int k2 = 2;

int arrSize2 = sizeof(arr2) / sizeof(arr2[0]);

int result2 = findKthPositive(arr2, arrSize2, k2);

if (result2 != -1) {

printf("Test Case 2: The %dth missing positive integer is: %d\n", k2, result2);

} else {

printf("Test Case 2: The %dth missing positive integer is not found.\n", k2);

}

return 0;

}

Input:

arr1[] = {2, 3, 4, 7, 11}

K1=5

arr2[] = {1, 2, 3, 4}

K2=2

Output:

The 5th missing positive integer is: 9

The 2th missing positive integer is: 6

Question No 7

BINARY SEARCH

#include <stdio.h>

int binarySearch(int arr[], int left, int right, int target) {

while (left <= right) {

int mid = left + (right - left) / 2;

if (arr[mid] == target)

return mid;

if (arr[mid] < target)

left = mid + 1;

else

right = mid - 1;

}

return -1;

}

int main() {

int arr1[] = {5,10,15,20,25,30,35,40,45};

int n1 = sizeof(arr1) / sizeof(arr1[0]);

int target1 = 10;

int result1 = binarySearch(arr1, 0, n1 - 1, target1);

if (result1 == -1)

printf("Test Case 1: Element not found\n");

else

printf("Test Case 1: Element found at index %d\n", result1);

int arr2[] = {10,20,30,40,50,60};

int n2 = sizeof(arr2) / sizeof(arr2[0]);

int target2 = 8;

int result2 = binarySearch(arr2, 0, n2 - 1, target2);

if (result2 == -1)

printf("Test Case 2: Element not found\n");

else

printf("Test Case 2: Element found at index %d\n", result2);

int arr3[] = {21,32,40,54,65,76,87};

int n3 = sizeof(arr3) / sizeof(arr3[0]);

int target3 = 3;

int result3 = binarySearch(arr3, 0, n3 - 1, target3);

if (result3 == -1)

printf("Test Case 3: Element not found\n");

else

printf("Test Case 3: Element found at index %d\n", result3);

return 0;

}

Input:

arr1[] = {5,10,15,20,25,30,35,40,45}

arr2[] = {10,20,30,40,50,60}

arr3[] = {21,32,40,54,65,76,87}

Output:

Element found at index 4

Element found at index 5

Element found at index 2

Question No 8

COMBINSATION SUM 1

#include <stdio.h>

#include <stdlib.h>

void backtrack(int\* candidates, int candidatesSize, int target, int start, int\* path, int pathLen, int\*\* res, int\* returnSize, int\* returnColumnSizes) {

if (target < 0) {

return;

}

if (target == 0) {

res[\*returnSize] = (int\*)malloc(pathLen \* sizeof(int));

returnColumnSizes[\*returnSize] = pathLen;

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

res[\*returnSize][i] = path[i];

}

(\*returnSize)++;

return;

}

for (int i = start; i < candidatesSize; i++) {

path[pathLen] = candidates[i];

backtrack(candidates, candidatesSize, target - candidates[i], i, path, pathLen + 1, res, returnSize, returnColumnSizes);

}

}

int\*\* combinationSum(int\* candidates, int candidatesSize, int target, int\* returnSize, int\*\* returnColumnSizes) {

int\*\* res = (int\*\*)malloc(150 \* sizeof(int\*));

\*returnSize = 0;

int\* path = (int\*)malloc(150 \* sizeof(int));

\*returnColumnSizes = (int\*)malloc(150 \* sizeof(int));

backtrack(candidates, candidatesSize, target, 0, path, 0, res, returnSize, \*returnColumnSizes);

return res;

}

void printResult(int\*\* result, int returnSize, int\* returnColumnSizes) {

printf("[");

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

printf("[");

for (int j = 0; j < returnColumnSizes[i]; j++) {

printf("%d", result[i][j]);

if (j < returnColumnSizes[i] - 1) {

printf(",");

}

}

printf("]");

if (i < returnSize - 1) {

printf(",");

}

}

printf("]\n");

}

int main() {

int candidates1[] = {2, 3, 6, 7};

int target1 = 7;

int returnSize1;

int\* returnColumnSizes1;

int\*\* result1 = combinationSum(candidates1, 4, target1, &returnSize1, &returnColumnSizes1);

printf("Test Case 1: ");

printResult(result1, returnSize1, returnColumnSizes1);

int candidates2[] = {2, 3, 5};

int target2 = 8;

int returnSize2;

int\* returnColumnSizes2;

int\*\* result2 = combinationSum(candidates2, 3, target2, &returnSize2, &returnColumnSizes2);

printf("Test Case 2: ");

printResult(result2, returnSize2, returnColumnSizes2);

int candidates3[] = {2, 3, 5};

int target3 = 10;

int returnSize3;

int\* returnColumnSizes3;

int\*\* result3 = combinationSum(candidates3, 3, target3, &returnSize3, &returnColumnSizes3);

printf("Test Case 3: ");

printResult(result3, returnSize3, returnColumnSizes3);

for (int i = 0; i < returnSize1; i++) {

free(result1[i]);

}

free(result1);

free(returnColumnSizes1);

for (int i = 0; i < returnSize2; i++) {

free(result2[i]);

}

free(result2);

free(returnColumnSizes2);

for (int i = 0; i < returnSize3; i++) {

free(result3[i]);

}

free(result3);

free(returnColumnSizes3);

return 0;

}

Input:

candidates1[] = {2, 3, 6, 7}

target1 = 7

candidates2[] = {2, 3, 5}

target2 = 8

candidates3[] = {2, 3, 5}

target3 = 10

Output:

[[2,2,3],[7]]

[[2,2,2,2],[2,3,3],[3,5]]

[[2,2,2,2,2],[2,2,3,3],[2,3,5],[5,5]]

Question No 9

MERGE SORT

#include <stdio.h>

void merge(int a[], int l, int m, int r) {

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

int L[n1], R[n2];

for (i = 0; i < n1; i++)

L[i] = a[l + i];

for (j = 0; j < n2; j++)

R[j] = a[m + 1 + j];

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2)

if (L[i] <= R[j])

a[k++] = L[i++];

else

a[k++] = R[j++];

while (i < n1)

a[k++] = L[i++];

while (j < n2)

a[k++] = R[j++];

}

void mergesort(int a[], int l, int r) {

if (l < r) {

int m = l + (r - l) / 2;

mergesort(a, l, m);

mergesort(a, m + 1, r);

merge(a, l, m, r);

}

}

void printArray(int a[], int n) {

for (int i = 0; i < n; i++)

printf("%d ", a[i]);

printf("\n");

}

int main() {

int a1[] = {31,23,35,27,11,21,15,28};

int n1 = sizeof(a1) / sizeof(a1[0]);

mergesort(a1, 0, n1 - 1);

printf("Sorted array for Test Case 1:\n");

printArray(a1, n1);

int a2[] = {22,34,25,36,43,67,52,13,65,17};

int n2 = sizeof(a2) / sizeof(a2[0]);

mergesort(a2, 0, n2 - 1);

printf("Sorted array for Test Case 2:\n");

printArray(a2, n2);

return 0;

}

Input:

a1[] = {31,23,35,27,11,21,15,28}

a2[] = {22,34,25,36,43,67,52,13,65,17}

Output:

11,15,21,23,27,28,31,35

13,17,22,25,34,36,43,52,65,67

Question No 10

CLOSEST PAIR OF POINTS(DIVIDE AND CONQUER)

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

struct Point {

int x, y;

double distance;

};

double calculateDistance(int x, int y) {

return sqrt(x \* x + y \* y);

}

int compare(const void\* a, const void\* b) {

struct Point\* point1 = (struct Point\*)a;

struct Point\* point2 = (struct Point\*)b;

return (point1->distance > point2->distance) - (point1->distance < point2->distance);

}

void kClosestPoints(int points[][2], int pointsSize, int k) {

struct Point\* pointArray = (struct Point\*)malloc(pointsSize \* sizeof(struct Point));

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

pointArray[i].x = points[i][0];

pointArray[i].y = points[i][1];

pointArray[i].distance = calculateDistance(points[i][0], points[i][1]);

}

qsort(pointArray, pointsSize, sizeof(struct Point), compare);

printf("[");

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

printf("[%d,%d]", pointArray[i].x, pointArray[i].y);

if (i < k - 1) {

printf(",");

}

}

printf("]\n");

free(pointArray);

}

int main() {

int points1[][2] = {{1, 3}, {-2, 2}, {5, 8}, {0, 1}};

int k1 = 2;

printf("Output for Test Case 1: ");

kClosestPoints(points1, sizeof(points1) / sizeof(points1[0]), k1);

int points2[][2] = {{1, 3}, {-2, 2}};

int k2 = 1;

printf("Output for Test Case 2: ");

kClosestPoints(points2, sizeof(points2) / sizeof(points2[0]), k2);

int points3[][2] = {{3, 3}, {5, -1}, {-2, 4}};

int k3 = 2;

printf("Output for Test Case 3: ");

kClosestPoints(points3, sizeof(points3) / sizeof(points3[0]), k3);

return 0;

}

Input:

points1[][2] = {{1, 3}, {-2, 2}, {5, 8}, {0, 1}}

points2[][2] = {{1, 3}, {-2, 2}}

points3[][2] = {{3, 3}, {5, -1}, {-2, 4}}

Output:

The Closest pair points for Test Case 1 is [[0,1],[-2,2]]

The Closest pair points for Test Case 2 is [[-2,2]]

The Closest pair points for Test Case 3 is [[3,3],[-2,4]]