fractional Knapsack

#include <stdio.h>

#include <stdlib.h>

// Define a structure for items with weight (w) and value (v)

typedef struct {

int w, v;

} Item;

// Comparator function to sort items by value-to-weight ratio in descending order

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

double r1 = ((Item \*)a)->v / (double)((Item \*)a)->w;

double r2 = ((Item \*)b)->v / (double)((Item \*)b)->w;

return r2 - r1 > 0 ? 1 : -1; // Higher ratio first

}

// Function to compute the maximum value in the fractional knapsack

double fractionalKnapsack(Item a[], int n, int W) {

qsort(a, n, sizeof(Item), cmp); // Sort items by ratio

double val = 0.0; // Total value collected

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

If (a[i].w <= W) { // If whole item can be taken

val += a[i].v;

W -= a[i].w;

} else { // Take fractional part of item

val += a[i].v \* ((double)W / a[i].w);

break; // Knapsack is full

}

}

return val; // Return maximum value

}

int main() {

// Example items: {weight, value}

Item a[] = {{10, 60}, {20, 100}, {30, 120}};

int W = 50; // Total capacity of knapsack

// Call function and print result

printf("Max value: %.2f\n", fractionalKnapsack(a, 3, W));

return 0;

}

0/1 Knapsack

#include <stdio.h>

// Function to return max of two numbers

int max(int a, int b) {

return a > b ? a : b;

}

// Function to solve 0/1 Knapsack

int knapsack(int W, int wt[], int val[], int n) {

int dp[n + 1][W + 1];

// Build table dp[][] in bottom-up manner

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

for (int w = 0; w <= W; w++) {

if (i == 0 || w == 0)

dp[i][w] = 0; // Base case

else if (wt[i - 1] <= w)

dp[i][w] = max(val[i - 1] + dp[i - 1][w - wt[i - 1]], dp[i - 1][w]);

else

dp[i][w] = dp[i - 1][w]; // Cannot include item

}

}

return dp[n][W]; // Maximum value in knapsack

}

int main() {

int val[] = {60, 100, 120};

int wt[] = {10, 20, 30};

int W = 50;

int n = sizeof(val) / sizeof(val[0]);

printf("Max value: %d\n", knapsack(W, wt, val, n));

return 0;

}

jobSequencing

#include <stdio.h>

#include <stdlib.h>

typedef struct {

int id, deadline, profit;

} Job;

// Compare jobs by profit (descending)

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

return ((Job \*)b)->profit - ((Job \*)a)->profit;

}

// Find maximum deadline

int findMaxDeadline(Job jobs[], int n) {

int max = jobs[0].deadline;

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

if (jobs[i].deadline > max)

max = jobs[i].deadline;

return max;

}

// Job sequencing function

void jobSequencing(Job jobs[], int n) {

qsort(jobs, n, sizeof(Job), cmp); // Sort by profit

int maxDeadline = findMaxDeadline(jobs, n);

int slot[maxDeadline + 1]; // Time slots

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

slot[i] = -1; // -1 means free

int totalProfit = 0, count = 0;

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

for (int j = jobs[i].deadline; j > 0; j--) {

if (slot[j] == -1) { // Found free slot

slot[j] = i;

totalProfit += jobs[i].profit;

count++;

break;

}

}

}

printf("Jobs done: %d\n Total Profit: %d\n Sequence: ", count, totalProfit);

for (int i = 1; i <= maxDeadline; i++)

if (slot[i] != -1)

printf("J%d", jobs[slot[i]].id);

printf("\n");

}

int main() {

Job jobs[] = {{1, 2, 100}, {2, 1, 19}, {3, 2, 27}, {4, 1, 25}, {5, 3, 15}};

int n = sizeof(jobs) / sizeof(jobs[0]);

jobSequencing(jobs, n);

return 0;

}

Jobs done: 3

Total Profit: 142

Sequence: J3 J1 J5

Optimal Merge pattern

#include <stdio.h>

#include <stdlib.h>

// Compare function for qsort

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

return \*(int \*)a - \*(int \*)b;

}

// Function to find minimum total merge cost

int optimalMerge(int files[], int n) {

int totalCost = 0;

// Repeat until only one file is left

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

qsort(files, n - i, sizeof(int), cmp); // Sort remaining files

// Merge two smallest files

int merged = files[0] + files[1];

totalCost += merged;

// Replace them with the merged file

files[0] = merged;

files[1] = files[n - i - 1]; // Move last to second place

}

return totalCost;

}

int main() {

int files[] = {20, 30, 10, 5};

int n = sizeof(files) / sizeof(files[0]);

int minCost = optimalMerge(files, n);

printf("Minimum total merge cost: %d\n", minCost);

return 0;

}

BFS

#include <stdio.h>

#define MAX 100

int adj[MAX][MAX], visited[MAX], queue[MAX], front = 0, rear = 0;

void BFS(int start,int n) {

visited[start] = 1;

queue[rear++] = start;

printf("BFS: ");

while (front < rear) {

int v = queue[front++];

printf("%d ", v);

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

if (adj[v][i] && !visited[i]) {

visited[i] = 1;

queue[rear++] = i;

}

}

}

printf("\n");

}

int main() {

int n = 6; // Number of vertices

adj[0][1] = adj[1][0] = 1;

adj[0][2] = adj[2][0] = 1;

adj[1][3] = adj[3][1] = 1;

adj[1][4] = adj[4][1] = 1;

adj[2][5] = adj[5][2] = 1;

BFS(0, n);

return 0;

}

BFS: 0 1 2 3 4 5

DFS

#include <stdio.h>

#define MAX 100

int adj[MAX][MAX], visited[MAX];

void DFS(int v,int n) {

printf("%d ", v);

visited[v] = 1;

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

if (adj[v][i] && !visited[i])

DFS(i, n);

}

}

int main() {

int n = 6; // Number of vertices

adj[0][1] = adj[1][0] = 1;

adj[0][2] = adj[2][0] = 1;

adj[1][3] = adj[3][1] = 1;

adj[1][4] = adj[4][1] = 1;

adj[2][5] = adj[5][2] = 1;

printf("DFS: ");

DFS(0, n);

printf("\n");

return 0;

}

DFS: 0 1 3 4 2 5

Krushkal:

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

typedef struct {

int u, v, w;

} Edge;

int parent[MAX],n, e;

Edge edges[MAX];

int find(int x) {

return parent[x] == x ? x : (parent[x] = find(parent[x]));

}

void unite(int a, int b) {

parent[find(a)] = find(b);

}

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

return ((Edge\*)a)->w - ((Edge\*)b)->w;

}

int main() {

printf("Enter no. of vertices and edges: ");

scanf("%d %d", &n, &e);

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

printf("Edge %d (u v w): ", i + 1);

scanf("%d %d %d", &edges[i].u, &edges[i].v, &edges[i].w);

}

for (int i = 0; i < n; i++) parent[i] = i;

qsort(edges, e, sizeof(Edge), cmp);

int cost = 0;

printf("\nEdges in MST:\n");

for (int i = 0, count = 0; i < e && count < n - 1; i++) {

int u = edges[i].u, v = edges[i].v;

if (find(u) != find(v)) {

unite(u, v);

cost += edges[i].w;

printf("%d - %d : %d\n", u, v, edges[i].w);

count++;

}

}

printf("Total Cost: %d\n", cost);

return 0;

}

Enter no. of vertices and edges: 4 5

Edge 1 (u v w): 0 1 10

Edge 2 (u v w): 0 2 6

Edge 3 (u v w): 0 3 5

Edge 4 (u v w): 1 3 15

Edge 5 (u v w): 2 3 4

Edges in MST:

2 - 3 : 4

0 - 3 : 5

0 - 1 : 10

Total Cost: 19

Hashing:

#include <stdio.h>

#include <stdlib.h>

int main() {

int size, i, ind, num, ch, key;

int a[100];

printf("Enter the size of the hash table:\n");

scanf("%d", &size);

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

a[i] = -1;

while (1) {

printf("\nMenu:\n");

printf("1. Insert data\n");

printf("2. Display table\n");

printf("3. Search key\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &ch);

switch (ch) {

case 1:

printf("Enter the data to be inserted: ");

scanf("%d", &num);

ind = num % size;

a[ind] = num;

break;

case 2:

printf("\nHash table contents:\n");

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

if (a[i] != -1)

printf("Index %d: %d\n", i, a[i]);

else

printf("Index %d: EMPTY\n", i);

}

break;

case 3:

printf("Enter key to search: ");

scanf("%d", &key);

ind = key % size;

if (a[ind] == key)

printf("Element found at index %d\n", ind);

else

printf("Element not found\n");

break;

case 4:

exit(0);

default:

printf("Invalid choice! Please try again.\n");

}

}

return 0;

}

Heap sort

#include <stdio.h>

void heapify(int a[], int n, int i) {

int largest = i, l = 2\*i + 1, r = 2\*i + 2;

if (l < n && a[l] > a[largest]) largest = l;

if (r < n && a[r] > a[largest]) largest = r;

if (largest != i) {

int t = a[i]; a[i] = a[largest]; a[largest] = t;

heapify(a, n, largest);

}

}

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

for (int i = n/2 - 1; i >= 0; i--) heapify(a, n, i);

for (int i = n - 1; i > 0; i--) {

int t = a[0]; a[0] = a[i]; a[i] = t;

heapify(a, i, 0);

}

}

int main() {

int a[] = {5, 3, 8, 4, 1, 2};

int n = sizeof(a)/sizeof(a[0]);

heapsort(a, n);

for (int i = 0; i < n; i++) printf("%d ", a[i]);

return 0;

}