DAA LAB ASSIGNMENT 2

NAME->Chandranshu Bhardwaj

ROLL NO.->102203797

Q1.) Activity selection?

Ans->

Run Debug Stop Share Save { } Beautify

main.cpp

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

struct Activity {
    int start;
    int finish;
};

bool compareActivities(const Activity& a, const Activity& b) {
    return a.finish < b.finish;
}

void activitySelection(vector<Activity>& activities) {
    sort(activities.begin(), activities.end(), compareActivities);

    cout << "Selected Activities: ";
    cout << "(" << activities[0].start << ", " << activities[0].finish << ") ";

    int lastSelected = 0;

    for (int i = 1; i < activities.size(); ++i) {
        if (activities[i].start >= activities[lastSelected].finish) {
            cout << "(" << activities[i].start << ", " << activities[i].finish << ") ";
            lastSelected = i;
        }
    }
}

int main() {
    vector<Activity> activities = {{1, 2}, {3, 4}, {0, 6}, {5, 7}, {8, 9}, {5, 9}};
    activitySelection(activities);
    return 0;
}
```

```
Selected Activities: (1, 2) (3, 4) (5, 7) (8, 9)

...Program finished with exit code 0
Press ENTER to exit console.
```

Q2.) Job Sequence?

Ans->

Run Debug Stop Share Save { } Beautify

main.cpp

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

struct Job {
    char id;
    int deadline;
    int profit;
};

bool compareJobs(const Job& a, const Job& b) {
    return a.profit > b.profit;
}

void jobSequencing(vector<Job>& jobs) {
    sort(jobs.begin(), jobs.end(), compareJobs);

    int maxDeadline = 0;
    for (const Job& job : jobs) {
        maxDeadline = max(maxDeadline, job.deadline);
    }

    vector<bool> slot(maxDeadline + 1, false);
    vector<char> result;

    for (const Job& job : jobs) {
        for (int i = min(maxDeadline, job.deadline); i >= 1; --i) {
            if (!slot[i]) {
                slot[i] = true;
                result.push_back(job.id);
                break;
            }
        }
    }

    cout << "Job Sequence: ";
    for (char jobId : result) {
        cout << jobId << " ";
    }
```

```
Job Sequence: c a


...Program finished with exit code 0
Press ENTER to exit console.
```

Q3.)Fractional knapsack?

Ans->

Run Debug Stop Share Save { } Beautify

main.cpp

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

struct Item {
    int weight;
    int value;
};

bool compareItems(const Item& a, const Item& b) {
    double ratioA = (double)a.value / a.weight;
    double ratioB = (double)b.value / b.weight;
    return ratioA > ratioB;
}

double fractionalKnapsack(int capacity, vector<Item>& items) {
    sort(items.begin(), items.end(), compareItems);

    double totalValue = 0.0;
    int currentWeight = 0;

    for (const Item& item : items) {
        if (currentWeight + item.weight <= capacity) {
            totalValue += item.value;
            currentWeight += item.weight;
        } else {

            int remainingCapacity = capacity - currentWeight;
            totalValue += (double)item.value * remainingCapacity / item.weight;

        }
    }

    return totalValue;
}

int main() {
    vector<Item> items = {{10, 60}, {20, 100}, {30, 120}};
    int capacity = 50;
```

```
Maximum value in knapsack: 240


...Program finished with exit code 0
Press ENTER to exit console.
```

Q3.)Huffman Coding?

Ans->

main.cpp

```cpp
#include <iostream>
#include <queue>
#include <map>
#include <string>

using namespace std;

struct Node {
    char data;
    int frequency;
    Node *left, *right;

    Node(char data, int frequency) {
        this->data = data;
        this->frequency = frequency;
        left = right = nullptr;
    }
};


struct compare {
    bool operator()(Node* l, Node* r) {
        return l->frequency > r->frequency;
    }
};


void encode(Node* root, string str, map<char, string>& huffmanCode) {
    if (root == nullptr) return;

    if (!root->left && !root->right) {
        huffmanCode[root->data] = str;
    }

    encode(root->left, str + "0", huffmanCode);


    encode(root->right, str + "1", huffmanCode);
}
```

input

```
Huffman Codes:
  : 011
. : 111110
H : 111111
a : 010
c : 11001
d : 10111
e : 00100
f : 0011
g : 11011
h : 110000
i : 1110
l : 00101
```

Run Debug Stop Share Save { } Beautify

main.cpp

```cpp

map<char, string> buildHuffmanTree(string text) {

    map<char, int> freq;
    for (char c : text) {
        freq[c]++;
    }


    priority_queue<Node*, vector<Node*>, compare> pq;
    for (auto pair : freq) {
        pq.push(new Node(pair.first, pair.second));
    }

    while (pq.size() != 1) {
        Node* left = pq.top(); pq.pop();
        Node* right = pq.top(); pq.pop();

        Node* newNode = new Node('$', left->frequency + right->frequency);
        newNode->left = left;
        newNode->right = right;

        pq.push(newNode);
    }

    Node* root = pq.top();
    map<char, string> huffmanCode;
    encode(root, "", huffmanCode);

    return huffmanCode;
}

int main() {
    string text = "Huffman coding is a data compression algorithm.";
    map<char, string> huffmanCode = buildHuffmanTree(text);

    cout << "Huffman Codes:\n";
    for (auto pair : huffmanCode) {
        cout << pair.first << " : " << pair.second << endl;
    }

```

input

```
i : 1110
l : 00101
m : 1001
n : 1000
o : 000
p : 10110
r : 11010
s : 1010
t : 11110
u : 110001
```