

**Nitin Chaudhary**

**9921103163**

**F7**

### **APS Lab Week 10**

//According to teacher's feedback- I have high ability to solve the problem which I have solve previously but I am not good at facing the new problems. For this they suggested the solution that I should try solving every problem for multiple duration of time, only then you will see that your problem solving ability is improving.

//Also according to my parents- Doing both good(study) and bad works simultaneously will never allow you to reach to the success.

Hence, I have understood this concept and from now will only do study priority wise and will always try for every problem first and then do it by own and will prefer the solution only when I am not able to solve it even by putting so much effort in it.

**Q1. Consider that you have an integer array. Present a greedy method to the minimum product possible within the elements present in the array (subset).**

### **Using dynamic programming-**

```
#include <bits/stdc++.h>
```

```
using namespace std;
```

```
//Below is the given function which will return the minimum product subarray.
```

```
int minProduct(vector<int>& nums) {
```

```
    //Initializing the required variables
```

```
    int n = nums.size();
```

```
    int minProduct = 1;
```

```
    int maxProduct = 1;
```

```
    int ans = INT_MAX;
```

```
    //Loop for traversing the entire array
```

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

//If number is negative then we will swap these numbers as after the multiplication of the negative number, maxProduct will become minProduct and vice versa.

```
        if(nums[i] < 0){
            swap(maxProduct, minProduct);
        }
        minProduct = min(minProduct*nums[i], nums[i]);
        maxProduct = max(maxProduct*nums[i], nums[i]);
        //our answer will be the min of both.
        ans = min(ans,minProduct);
    }
    return ans;
}
int main()
{
    vector<int>ans={-1,0,1,2,3,-4};
    cout<<minProduct(ans);
    return 0;
}
```

## Greedy approach-

```
#include <bits/stdc++.h>
using namespace std;
```

```
int minProductSubset(int a[], int n)
{
    if (n == 1)
        return a[0];
    int max_neg = INT_MIN, min_pos = INT_MAX, count_neg = 0,
        count_zero = 0, prod = 1;
    for (int i = 0; i < n; i++) {
        if (a[i] == 0) {
            count_zero++;
            continue;
        }
        if (a[i] < 0) {
            count_neg++;
            max_neg = max(max_neg, a[i]);
        }
    }
}
```

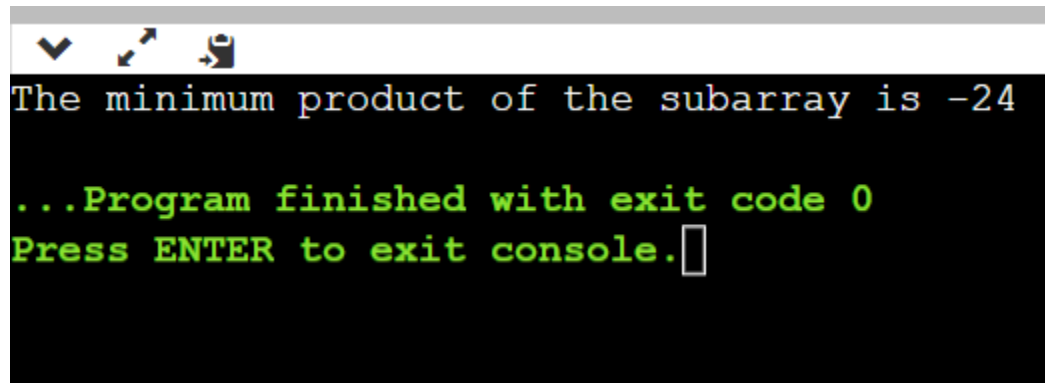
```

        if (a[i] > 0)
            min_pos = min(min_pos, a[i]);
        prod = prod * a[i];
    }
    if (count_zero == n || (count_neg == 0 && count_zero > 0))
        return 0;
    if (count_neg == 0)
        return min_pos;
    if (!(count_neg & 1) && count_neg != 0)
        prod = prod / max_neg;
    return prod;
}

int main()
{
    int a[] = { -1, -1, -2, 4, 3 };
    int n = sizeof(a) / sizeof(a[0]);
    cout << "The minimum product of the subarray is "<<minProductSubset(a, n);
    return 0;
}

```

## Output-



```

The minimum product of the subarray is -24

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

```

**Q2. A student can do 'n' activities in a week. Each activity takes a particular number of hours (mention the start time and the end time). Write a program to take 'n' such inputs from user so that a student can schedule maximum number of activities.**

### **Code-**

//I am very happy because i have implemented this code by myself without taking help from internet for the same question.

//I just understood the approach and then I implemented the approach in my way using priority\_queue.

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    priority_queue<pair<int,int>,vector<pair<int,int>>,greater<pair<int,int>>> pq;
    int start;
    int end;
    for(int i=0;i<6;i++){
        cout<<"Enter the start and end time of activity "<<i+1<<": ";
        cin>>start;
        cin>>end;
        pq.push(make_pair(end,start));
    }
    int en=pq.top().first;
    cout<<"The following activities must be selected which has start time and end time as
given below:"<<endl;
    cout<<pq.top().second<<" "<<pq.top().first<<endl;
    pq.pop();
    int count=1;
    while(!pq.empty()){
        int st=pq.top().second;
        if(st>=en){
            cout<<pq.top().second<<" "<<pq.top().first<<endl;
            count++;
            en=pq.top().first;
        }
        pq.pop();
    }
    cout<<"The number of activities which can be done are "<<count<<endl;
```

```
    return 0;
}
```

## Output-

```
Enter the start and end time of activity 1: 1 2
Enter the start and end time of activity 2: 3 4
Enter the start and end time of activity 3: 0 6
Enter the start and end time of activity 4: 5 7
Enter the start and end time of activity 5: 8 9
Enter the start and end time of activity 6: 5 9
The following activities must be selected which has start time and end time as given below:
1 2
3 4
5 7
8 9
The number of activities which can be done are 4

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

**Q3. Implement Huffman tree for variable-length encoding. Apply for the following information.**

**Symbol Frequency**

**Q 3**

**p 23**

**T 30**

**a 12**

**d 18**

The below code is written by the help of GFG as this was very hard for me to implement without any help-

```
#include <cstdlib>
#include <iostream>
using namespace std;
#define MAX_TREE_HT 100
struct MinHeapNode {
    char data;
    unsigned freq;
    struct MinHeapNode *left, *right;
};
struct MinHeap {
    unsigned size;
    unsigned capacity;
    struct MinHeapNode** array;
```

```

};
struct MinHeapNode* newNode(char data, unsigned freq)
{
    struct MinHeapNode* temp = (struct MinHeapNode*)malloc(sizeof(struct
MinHeapNode));

    temp->left = temp->right = NULL;
    temp->data = data;
    temp->freq = freq;

    return temp;
}
struct MinHeap* createMinHeap(unsigned capacity)
{
    struct MinHeap* minHeap= (struct MinHeap*)malloc(sizeof(struct MinHeap));
    minHeap->size = 0;

    minHeap->capacity = capacity;

    minHeap->array = (struct MinHeapNode**)malloc(
        minHeap->capacity * sizeof(struct MinHeapNode*));
    return minHeap;
}
void swapMinHeapNode(struct MinHeapNode** a,
                     struct MinHeapNode** b)

{
    struct MinHeapNode* t = *a;
    *a = *b;
    *b = t;
}
void minHeapify(struct MinHeap* minHeap, int idx)

{
    int smallest = idx;
    int left = 2 * idx + 1;
    int right = 2 * idx + 2;

    if (left < minHeap->size
        && minHeap->array[left]->freq
            < minHeap->array[smallest]->freq)
        smallest = left;

```

```

        if (right < minHeap->size
            && minHeap->array[right]->freq
                < minHeap->array[smallest]->freq)
            smallest = right;

        if (smallest != idx) {
            swapMinHeapNode(&minHeap->array[smallest],
                           &minHeap->array[idx]);
            minHeapify(minHeap, smallest);
        }
    }

int isSizeOne(struct MinHeap* minHeap)
{
    return (minHeap->size == 1);
}

struct MinHeapNode* extractMin(struct MinHeap* minHeap)
{
    struct MinHeapNode* temp = minHeap->array[0];
    minHeap->array[0] = minHeap->array[minHeap->size - 1];

    --minHeap->size;
    minHeapify(minHeap, 0);

    return temp;
}

void insertMinHeap(struct MinHeap* minHeap,
                  struct MinHeapNode* minHeapNode)
{
    ++minHeap->size;
    int i = minHeap->size - 1;

    while (i
        && minHeapNode->freq
            < minHeap->array[(i - 1) / 2]->freq) {

        minHeap->array[i] = minHeap->array[(i - 1) / 2];
    }

```

```

        i = (i - 1) / 2;
    }

    minHeap->array[i] = minHeapNode;
}
void buildMinHeap(struct MinHeap* minHeap)
{
    int n = minHeap->size - 1;
    int i;

    for (i = (n - 1) / 2; i >= 0; --i)
        minHeapify(minHeap, i);
}
void printArr(int arr[], int n)
{
    int i;
    for (i = 0; i < n; ++i)
        cout << arr[i];

    cout << "\n";
}
int isLeaf(struct MinHeapNode* root)
{
    return !(root->left) && !(root->right);
}
struct MinHeap* createAndBuildMinHeap(char data[], int freq[], int size)
{
    struct MinHeap* minHeap = createMinHeap(size);

    for (int i = 0; i < size; ++i)
        minHeap->array[i] = newNode(data[i], freq[i]);

    minHeap->size = size;
    buildMinHeap(minHeap);

    return minHeap;
}
struct MinHeapNode* buildHuffmanTree(char data[], int freq[], int size)
{
    struct MinHeapNode *left, *right, *top;

```



```

    struct MinHeap* minHeap= createAndBuildMinHeap(data, freq, size);
    while (!isSizeOne(minHeap)) {
        left = extractMin(minHeap);
        right = extractMin(minHeap);
        top = newNode('$', left->freq + right->freq);

        top->left = left;
        top->right = right;

        insertMinHeap(minHeap, top);
    }
    return extractMin(minHeap);
}

void printCodes(struct MinHeapNode* root, int arr[],int top)
{
    if (root->left) {

        arr[top] = 0;
        printCodes(root->left, arr, top + 1);
    }
    if (root->right) {

        arr[top] = 1;
        printCodes(root->right, arr, top + 1);
    }
    if (isLeaf(root)) {

        cout << root->data << ": ";
        printArr(arr, top);
    }
}

void HuffmanCodes(char data[], int freq[], int size)
{
    struct MinHeapNode* root= buildHuffmanTree(data, freq, size);
    int arr[MAX_TREE_HT], top = 0;
    cout<<"The huffman codes of the characters will be as follows:"<<endl;
    printCodes(root, arr, top);
}

int main()
{

    char arr[] = { 'Q', 'p', 'T', 'a', 'd'};
    int freq[] = { 3,23,30,12,18 };
    int size = sizeof(arr) / sizeof(arr[0]);

```

```
HuffmanCodes(arr, freq, size);  
return 0;  
}
```

## Output-

```
The huffman codes of the characters will be as follows:  
Q: 000  
a: 001  
d: 01  
p: 10  
T: 11  
  
...Program finished with exit code 0  
Press ENTER to exit console. 
```

**Q4. A mobile selling outlet sells various types of mobiles. However her shop is very small and can contain only 40 items, in total.**

**Mobile Count Profit**

**Redme Note 9 10 20000**

**Samsung Galaxy M12 5 10500**

**OnePlus 5 26 89000**

**Realme Narzo 20 Pro 8 4000**

**Xiaomi Poco M3 12 23000**

**Present a greedy approach to solve this.**

//I am happy as I have solved this problem by own even there is no solution available for this on internet.

```
#include<bits/stdc++.h>
```

using namespace std;

//this problem can be solved using fractional knapsack greedy approach.

```
int main(){
    priority_queue<pair<int,int>,vector<pair<int,int>>>pq;
    int count;
    int profit;
    cout<<"Enter the value of count and profit:"<<endl;
    for(int i=0;i<5;i++){
        cin>>count;
        cin>>profit;
        pq.push(make_pair(profit,count));
    }
    int capacity=40;
    cout<<"Number of Items included "<<"Profit value"<<endl;
    while(capacity&&!pq.empty()){
        int cnt=pq.top().second;
        if(capacity>=cnt){
            capacity-=cnt;
            cout<<cnt<<"          "<<pq.top().first<<endl;
        }
        else{
            cout<<capacity<<"          "<<pq.top().first<<endl;
            capacity=0;
        }
        pq.pop();
    }
    return 0;
}
```

## Output-

```
Enter the value of count and profit:
10 20000
5 10500
26 89000
8 4000
12 23000
Number of Items included Profit value
26 89000
12 23000
2 20000
...Program finished with exit code 0
Press ENTER to exit console.
```

**Q5.** Each of the states is divided into many districts for administrative reasons. These districts have many neighbors that share their borders. If we want to colour each of the districts in such a manner that adjacent districts (districts that share border) cannot be coloured with the same colour, perform for the following map.

### Code-

```
#include <iostream>
#include <list>
using namespace std;
class Graph
{
    int V;
    list<int> *adj;
public:
```

```

Graph(int V) { this->V = V; adj = new list<int>[V]; }
~Graph()     { delete [] adj; }
void addEdge(int v, int w);
void greedyColoring();
};

```

```

void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w);
    adj[w].push_back(v);
}
void Graph::greedyColoring()
{
    int result[V];
    result[0] = 0;
    for (int u = 1; u < V; u++)
        result[u] = -1;
    bool available[V];
    for (int cr = 0; cr < V; cr++)
        available[cr] = false;
    for (int u = 1; u < V; u++)
    {
        list<int>::iterator i;
        for (i = adj[u].begin(); i != adj[u].end(); ++i)
            if (result[*i] != -1)
                available[result[*i]] = true;
        int cr;
        for (cr = 0; cr < V; cr++)
            if (available[cr] == false)
                break;

        result[u] = cr;
        for (i = adj[u].begin(); i != adj[u].end(); ++i)
            if (result[*i] != -1)
                available[result[*i]] = false;
    }
    for (int u = 0; u < V; u++)
        cout << "Vertex " << u << " ----> Color "
            << result[u] << endl;
}

```

```
int main()
{
    Graph g1(5);
    g1.addEdge(0, 1);
    g1.addEdge(0, 2);
    g1.addEdge(1, 2);
    g1.addEdge(1, 3);
    g1.addEdge(2, 3);
    g1.addEdge(3, 4);
    cout << "Coloring of graph 1 \n";
    g1.greedyColoring();
    return 0;
}
```

### Output-

```
Coloring of graph 1
Vertex 0 ---> Color 0
Vertex 1 ---> Color 1
Vertex 2 ---> Color 2
Vertex 3 ---> Color 0
Vertex 4 ---> Color 1

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