BCSE204P - Design and Analysis of Algorithms Lab

Cycle sheet - 1

Name: Dhruv Rajeshkumar Shah
Reg no: 21BCE0611

Q1. Write the programs for the following algorithms -

a) Maximum subarray sum
b) Karatsuba's fast multiplication
c) Huffman coding
d) Fractional Knapsack

Ans.

a) Maximum subarray sum

CODE

```
// Dhruv Rajeshkumar Shah
// 21BCE0611

// Maximum sum of the subarray

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


// Function to find the maximum of two numbers
int max(int a,int b)
{
   return a>b?a:b;
}

// Function to find cross sum of the array
int crosssum(int a[],int l,int mid,int h)
{
   int s=0; int left=INT_MIN;
   for(int i=mid;i>=l;i--)
   {
       s+=a[i];
       if(s>left)
         left=s;
   }
   s=0;int right=INT_MIN;
   for(int i=mid+1;i<=h;i++)
   {
       s+=a[i];
```

```
        if(s>right)
          right=s;
    }
    return(left+right);
}

// Main function to find the maximum sum of the subarray
int maxsum(int a[], int l,int h)
{
    if(l==h)
      return a[l];
    else
    {
        int mid=(l+h)/2;
        int temp=max(maxsum(a,l,mid),maxsum(a,mid+1,h));
        return max(temp,crosssum(a,l,mid,h));
    }
}

// Main function
int main()
{
    // Taking input from the user
    int n;
    cout<<"Enter the number of elements in the array: ";
    cin>>n;
    int arr[n];

    cout<<"Enter the elements of the array: ";
    for(int i=0;i<n;i++)
      cin>>arr[i];

    // Printing the maximum sum of the subarray
    cout<<"The maximum sum of the subarray is: ";
    cout<<maxsum(arr,0,7)<<endl;
}
```

Screenshot

Q1 > max_subarr_sum.cpp > main()

```cpp
// Dhruv Rajeshkumar Shah
// 21BCE0611

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


// Function to find the maximum of two numbers
int max(int a,int b)
{
    return a>b?a:b;
}

// Function to find cross sum of the array
int crosssum(int a[],int l,int mid,int h)
{
    int s=0; int left=INT_MIN;
    for(int i=mid;i>=l;i--)
    {
        s+=a[i];
        if(s>left)
          left=s;
    }
    s=0;int right=INT_MIN;
    for(int i=mid+1;i<=h;i++)
    {
        s+=a[i];
        if(s>right)
          right=s;
    }
    return(left+right);
}

// Main function to find the maximum sum of the subarray
int maxsum(int a[], int l,int h)
{
    if(l==h)
      return a[l];
    else
    {
        int mid=(l+h)/2;
        int temp=max(maxsum(a,l,mid),maxsum(a,mid+1,h));
        return max(temp,crosssum(a,l,mid,h));
    }
}

// Main function
```

```cpp
int main()
{
    // Taking input from the user
    int n;
    cout<<"Enter the number of elements in the array: ";
    cin>>n;
    int arr[n];

    cout<<"Enter the elements of the array: ";
    for(int i=0;i<n;i++)
      cin>>arr[i];

    // Printing the maximum sum of the subarray
    cout<<"The maximum sum of the subarray is: ";
    cout<<maxsum(arr,0,7)<<endl;
}
```

OUTPUT

```
~/Coding/DAA/cyclic_1 g++ ./Q1/max_subarr_sum.cpp -o ./out/max_subarr_sum && ./out/max_subarr_sum
Enter the number of elements in the array: 8
Enter the elements of the array: -2 -5 6 -2 -3 1 5 -6
The maximum sum of the subarray is: 7
```

b) Karatsuba's fast multiplication

CODE

```cpp
// Dhruv Rajeshkumar Shah
// 21BCE0611

// Karatsuba algorithm

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

// Main karatsuba function
int karat(int a,int b)
{
   // Converting the numbers to string
   string as=to_string(a);
   string bs=to_string(b);
```

```
    // Finding the length of the numbers
    int alen=as.length();
    int blen=bs.length();

    // Base case
    if (alen==1 && blen==1)
      return a*b;
    else
    {
        // Padding the smaller number with 0s
        int n=alen>blen?alen:blen;
        while(alen!=n)
        {
            as="0"+as;
            alen=as.length();
        }
        while(blen!=n)
        {
            bs="0"+bs;
            blen=bs.length();
        }

        // Dividing the numbers into two halves
        int aL=stoi(as.substr(0,n/2));
        int aR=stoi(as.substr(n/2,n-(n/2)));
        int bL=stoi(bs.substr(0,n/2));
        int bR=stoi(bs.substr(n/2,n-(n/2)));

        // Recursively calling the function
        int x1=karat(aL,bL);
        int x2=karat(aL+aR,bL+bR);
        int x3=karat(aR,bR);

        int m=ceil(n/2.0);

        // Calculating and returning the final answer
        return(x1*pow(10,m*2)+(x2-x1-x3)*pow(10,m)+x3);
    }
}
```

```
int main()
{
   // Taking input from the user
   int a,b;
   cout<<"Enter the first number: ";
   cin>>a;
   cout<<"Enter the second number: ";
   cin>>b;

   // Printing the answer
   cout<<"The product of the two numbers is: ";
   cout<<karat(a,b)<<endl;
}
```

Screenshot

Q1 > C++ karatsuba.cpp > ...

```cpp
// Dhruv Rajeshkumar Shah
// 21BCE0611

// Karatsuba algorithm

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

// Main karatsuba function
int karat(int a,int b)
{
    // Converting the numbers to string
    string as=to_string(a);
    string bs=to_string(b);

    // Finding the length of the numbers
    int alen=as.length();
    int blen=bs.length();

    // Base case
    if (alen==1 && blen==1)
      return a*b;
    else
    {
        // Padding the smaller number with 0s
        int n=alen>blen?alen:blen;
        while(alen!=n)
        {
            as="0"+as;
            alen=as.length();
        }
        while(blen!=n)
        {
            bs="0"+bs;
            blen=bs.length();
        }

        // Dividing the numbers into two halves
        int aL=stoi(as.substr(0,n/2));
        int aR=stoi(as.substr(n/2,n-(n/2)));
        int bL=stoi(bs.substr(0,n/2));
        int bR=stoi(bs.substr(n/2,n-(n/2)));

        // Recursively calling the function
        int x1=karat(aL,bL);
        int x2=karat(aL+aR,bL+bR);
        int x3=karat(aR,bR);
```

```
Q1 > C++ karatsuba.cpp > ...
49
50          int m=ceil(n/2.0);
51
52          // Calculating and returning the final answer
53          return(x1*pow(10,m*2)+(x2-x1-x3)*pow(10,m)+x3);
54      }
55  }
56  int main()
57  {
58      // Taking input from the user
59      int a,b;
60      cout<<"Enter the first number: ";
61      cin>>a;
62      cout<<"Enter the second number: ";
63      cin>>b;
64
65      // Printing the answer
66      cout<<"The product of the two numbers is: ";
67      cout<<karat(a,b)<<endl;
68  }
69
```

OUTPUT

```
~/Coding/DAA/cyclic_1/Q1  cd "/home/dhruv/Coding/DAA/cyclic_1/Q1/" && g++ karatsuba.
Enter the first number: 1980
Enter the second number: 2315
The product of the two numbers is: 4583700
```

c) Huffman Coding

CODE

```
// Dhruv Rajeshkumar Shah
// 21BCE0611

// Huffman coding

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


// Queue node
struct Node {
   char letter;
   int freq;
```

```
    struct Node *left, *right;
    Node (char letter, int freq)
    {
        left = right = NULL;
        this->letter = letter;
        this->freq = freq;
    };
};

// Function to compare two nodes
struct compare {
    bool operator()(Node* l, Node* r)
    {
        return (l->freq > r->freq);
    }
};


// Function to print the huffman code
void printCode(struct Node* root, string str)
{
    if (!root)
        return;
    if (root->letter != '*')
        cout << root->letter << ": " << str << "\n";
    printCode(root->left, str + "0");
    printCode(root->right, str + "1");
}

// Hoofman coding function
void huff(char a[], int f[], int n){
    // Create a min heap
    priority_queue<Node*, vector<Node*>, compare> minHeap;
    // Insert all the characters
    for (int i = 0; i < n; ++i)
        minHeap.push(new Node(a[i], f[i]));

    struct Node *left, *right, *top;

    // Iterate while size of heap doesn't become 1
```

```
    while (minHeap.size() != 1) {
        left = minHeap.top();
        minHeap.pop();
        right = minHeap.top();
        minHeap.pop();
        top = new Node('*', left->freq + right->freq);
        top->left = left;
        top->right = right;
        minHeap.push(top);
    }

    // Print the huffman code
    printCode(minHeap.top(), "");
}


// Main function
int main()
{
    // Taking input from the user
    int n;
    cout<<"Enter the number of characters: ";
    cin>>n;
    char a[n];
    int f[n];
    cout<<"Enter the characters: ";
    for(int i=0;i<n;i++)
        cin>>a[i];
    cout<<"Enter the frequencies: ";
    for(int i=0;i<n;i++)
        cin>>f[i];


    // Printing the huffman code
    huff(a, f, n);
    return 0;
}
```

Screenshot

Q1 > C++ huff_code.cpp > ...

```cpp
// Dhruv Rajeshkumar Shah
// 21BCE0611

// Huffman coding

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


// Queue node
struct Node {
    char letter;
    int freq;
    struct Node *left, *right;
    Node (char letter, int freq)
    {
        left = right = NULL;
        this->letter = letter;
        this->freq = freq;
    };
};

// Function to compare two nodes
struct compare {
    bool operator()(Node* l, Node* r)
    {
        return (l->freq > r->freq);
    }
};


// Function to print the huffman code
void printCode(struct Node* root, string str)
{
    if (!root)
        return;
    if (root->letter != '*')
        cout << root->letter << ": " << str << "\n";
    printCode(root->left, str + "0");
    printCode(root->right, str + "1");
}

// Hoofman coding function
void huff(char a[], int f[], int n){
    // Create a min heap
    priority_queue<Node*, vector<Node*>, compare> minHeap;
    // Insert all the characters
```

Q1 > C++ huff_code.cpp > ...

```cpp
    for (int i = 0; i < n; ++i)
        minHeap.push(new Node(a[i], f[i]));

    struct Node *left, *right, *top;

    // Iterate while size of heap doesn't become 1
    while (minHeap.size() != 1) {
        left = minHeap.top();
        minHeap.pop();
        right = minHeap.top();
        minHeap.pop();
        top = new Node('*', left->freq + right->freq);
        top->left = left;
        top->right = right;
        minHeap.push(top);
    }

    // Print the huffman code
    printCode(minHeap.top(), "");
}


// Main function
int main()
{
    // Taking input from the user
    int n;
    cout<<"Enter the number of characters: ";
    cin>>n;
    char a[n];
    int f[n];
    cout<<"Enter the characters: ";
    for(int i=0;i<n;i++)
        cin>>a[i];
    cout<<"Enter the frequencies: ";
    for(int i=0;i<n;i++)
        cin>>f[i];


    // Printing the huffman code
    huff(a, f, n);
    return 0;
}

```

OUTPUT

```
~/Coding/DAA/cyclic_1/Q1 cd "/home/dhruv/Coding/DAA/cyclic_1/Q1/" && g++ huff_code.cpp -o ../out/huff_code
Enter the number of characters: 6
Enter the characters: a b c d e f
Enter the frequencies: 5 9 12 13 16 15
a: 000
b: 001
f: 01
e: 10
c: 110
d: 111

~/Coding/DAA/cyclic_1/Q1
```

d) Fractional Knapsack

CODE

```
// Dhruv Rajeshkumar Shah
// 21BCE0611

// Fractional Knapsack Problem

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


// Defining node
typedef struct node {
   int w;
   int v;
   float r;
} Item;


// Function to initialize the nodes
void init(Item a[], int n){
   // Taking weight and value of each item as input
   // and calculating the ratio
   for (int i = 0; i < n; i++){
       cout << "Enter weight and value of item " << i + 1 << ": ";
       cin >> a[i].w >> a[i].v;
       a[i].r = (float)a[i].v / a[i].w;
   }

   // Sorting the items in descending order of ratio
   sort(a, a + n, [](Item a, Item b){
       return a.r > b.r;
   });
}


// Function to calculate fractional knapsack
void fractKnap(Item a[], int n, int cap){
   float amt = 0;
```

```
    for (int i = 0; i < n; i++){
        if (a[i].w <= cap){
            cap -= a[i].w;
            amt += a[i].v;
        }
        else {
            amt += a[i].r * cap;
            cap = 0;
            break;
        }
    }

    // Printing the maximum amount
    cout << "Maximum amount: " << amt << endl;
}


// Main function
int main(){
    // Taking number of items and capacity of knapsack as input
    int n, cap;
    cout << "Enter number of items: ";
    cin >> n;
    cout << "Enter capacity of knapsack: ";
    cin >> cap;

    // Initializing the nodes
    Item a[n];
    init(a, n);

    // Calculating the maximum amount
    fractKnap(a, n, cap);

    return 0;
}
```

Screenshot

Q1 > fract_knap.cpp > ...

```cpp
// Dhruv Rajeshkumar Shah
// 21BCE0611

// Fractional Knapsack Problem

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


// Defining node
typedef struct node {
    int w;
    int v;
    float r;
} Item;


// Function to initialize the nodes
void init(Item a[], int n){
    // Taking weight and value of each item as input
    // and calculating the ratio
    for (int i = 0; i < n; i++){
        cout << "Enter weight and value of item " << i + 1 << ": ";
        cin >> a[i].w >> a[i].v;
        a[i].r = (float)a[i].v / a[i].w;
    }

    // Sorting the items in descending order of ratio
    sort(a, a + n, [](Item a, Item b){
        return a.r > b.r;
    });
}


// Function to calculate fractional knapsack
void fractKnap(Item a[], int n, int cap){
    float amt = 0;
    for (int i = 0; i < n; i++){
        if (a[i].w <= cap){
            cap -= a[i].w;
            amt += a[i].v;
        }
        else {
            amt += a[i].r * cap;
            cap = 0;
            break;
        }
```

```
Q1 > C++ fract_knap.cpp > ...
49      }
50
51      // Printing the maximum amount
52      cout << "Maximum amount: " << amt << endl;
53  }
54
55
56  // Main function
57  int main(){
58      // Taking number of items and capacity of knapsack as input
59      int n, cap;
60      cout << "Enter number of items: ";
61      cin >> n;
62      cout << "Enter capacity of knapsack: ";
63      cin >> cap;
64
65      // Initializing the nodes
66      Item a[n];
67      init(a, n);
68
69      // Calculating the maximum amount
70      fractKnap(a, n, cap);
71
72      return 0;
73  }
```

OUTPUT

```
~/Coding/DAA/cyclic_1/Q1 cd "/home/dhruv/Coding/DAA/cyclic_1/Q1/" && g++ fract_knap.cpp
Enter number of items: 5
Enter capacity of knapsack: 60
Enter weight and value of item 1: 5 30
Enter weight and value of item 2: 10 40
Enter weight and value of item 3: 15 45
Enter weight and value of item 4: 22 77
Enter weight and value of item 5: 25 90
Maximum amount: 230
```

Q2. Write the programs for the following algorithms -

a) 0-1 Knapsack
b) Matrix chain multiplication
c) Longest common subsequence
d) Subset sum
e) Assembly line scheduling
f) N-Queen problem
g) Graph coloring

Ans.

a) 0-1 Knapsack

CODE

```
// Dhruv Rajeshkumar Shah
// 21BCE0611

// A dynamic programming based 0-1 knapsack

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

// Function to return maximum of two integers
int max(int a, int b) { return (a > b) ? a : b; }
// Returns the maximum value that can be put in a knapsack of capacity W
int knapSack(int W, int wt[], int val[], int n)
{
   int i, w;

   vector<vector<int> > dp(n + 1, vector<int>(W + 1));

   for (i = 0; i <= n; i++) {
       for (w = 0; w <= W; w++) {
           if (i == 0 || w == 0)
               dp[i][w] = 0;
           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];
            }
        }
        return dp[n][W];
}

// Main function
int main()
{
    // Taking input from user
    int n, W;
    cout << "Enter the number of items: ";
    cin >> n;
    cout << "Enter the capacity of the knapsack: ";
    cin >> W;
    int weight[n], profit[n];
    cout << "Enter the weights of the items: ";
    for (int i = 0; i < n; i++)
        cin >> weight[i];
    cout << "Enter the profits of the items: ";
    for (int i = 0; i < n; i++)
        cin >> profit[i];

    // Printing the maximum profit
    cout << "The maximum profit is: ";
    cout << knapSack(W, weight, profit, n) << endl;


    return 0;
}
```

Screenshots

Q2 > C++ 01_knap.cpp > ...

```cpp
// Dhruv Rajeshkumar Shah
// 21BCE0611

// A dynamic programming based

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


// Function to return maximum of two integers
int max(int a, int b) { return (a > b) ? a : b; }

// Returns the maximum value that can be put in a knapsack of capacity W
int knapSack(int W, int wt[], int val[], int n)
{
    int i, w;

    vector<vector<int> > dp(n + 1, vector<int>(W + 1));

    for (i = 0; i <= n; i++) {
        for (w = 0; w <= W; w++) {
            if (i == 0 || w == 0)
                dp[i][w] = 0;
            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];
        }
    }
    return dp[n][W];
}


// Main function
int main()
{
    // Taking input from user
    int n, W;
    cout << "Enter the number of items: ";
    cin >> n;
    cout << "Enter the capacity of the knapsack: ";
    cin >> W;
    int weight[n], profit[n];
    cout << "Enter the weights of the items: ";
    for (int i = 0; i < n; i++)
        cin >> weight[i];
    cout << "Enter the profits of the items: ";
    for (int i = 0; i < n; i++)
        cin >> profit[i];

    // Printing the maximum profit
    cout << "The maximum profit is: ";
    cout << knapSack(W, weight, profit, n) << endl;


    return 0;
}
```

OUTPUT

```
~/Coding/DAA/cyclic_1/Q2  cd "/home/dhruv/Coding/DAA/cyclic_1/Q2/" && g++ 01_kna
Enter the number of items: 3
Enter the capacity of the knapsack: 50
Enter the weights of the items: 10 20 30
Enter the profits of the items: 60 100 120
The maximum profit is: 220

~/Coding/DAA/cyclic_1/Q2  cd "/home/dhruv/Coding/DAA/cyclic_1/Q2/" && g++ 01_kna
Enter the number of items: 4
Enter the capacity of the knapsack: 40
Enter the weights of the items: 30 10 40 20
Enter the profits of the items: 10 20 30 40
The maximum profit is: 60

~/Coding/DAA/cyclic_1/Q2
```

b) Matrix chain multiplication

CODE

```cpp
// Dhruv Rajeshkumar Shah
// 21BCE0611

// Matrix Multiplication

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

struct order
{
   int r1, c1, r2, c2;
};


int main()
{
   // taking input from user
   int n;
   cout << "Enter the number of matrices: ";
   cin >> n;
   int rank[n + 1];
   cout << "Enter the rank of the matrices: ";
```

```
   for (int i = 0; i <= n; i++)
       cin >> rank[i];
   order o[n][n];

   // Dynamic programming
   int c[n][n] = {0};
   int i, j, k, diff, q;
   for (i = 0; i < n; i++)
   {
       c[i][i] = 0;
       o[i][i].r1 = i;
       o[i][i].c1 = i;
       o[i][i].r2 = i;
       o[i][i].c2 = i;
   }
   for (diff = 2; diff <= n; diff++)
   {
       for (i = 0; i < n - diff + 1; i++)
       {
           j = i + diff - 1;
           int min = 999999;
           for (k = i; k <= j - 1; k++)
           {
               q = c[i][k] + c[k + 1][j] + rank[i] * rank[k + 1] * rank[j
+ 1];
               if (q < min)
               {
                   min = q;
                   o[i][j].r1 = i;
                   o[i][j].c1 = k;
                   o[i][j].r2 = k + 1;
                   o[i][j].c2 = j;
               }
           }
           c[i][j] = min;
       }
   }

   // Printing the dp table
   for (i = 0; i < n; i++)
```

```
    {
        for (j = 0; j < n; j++)
        {
            printf("%d ", c[i][j]);
        }
        printf("\n");
    }

    // Printing the cost (number of multiplications)
    printf("The total cost is %d \n", c[0][n - 1]);

    // Printing the order
    printf("The order is \n");
    i = 0;
    j = n - 1;
    if (fabs(o[i][j].r1 - o[i][j].c1) <= 1)
        printf("(%d,%d)", o[i][j].r1, o[i][j].c1);
    while (fabs(o[i][j].r1 - o[i][j].c1) > 1)
    {
        printf("(");
        i = o[i][j].r1;
        j = o[i][j].c1;
        printf("(%d,%d),(%d,%d)", o[i][j].r1, o[i][j].c1, o[i][j].r2,
o[i][j].c2);
    }
    printf(")");
    i = 0;
    j = n - 1;
    if (fabs(o[i][j].r2 - o[i][j].c2) <= 1)
        printf("(%d,%d)", o[i][j].r2, o[i][j].c2);
    while (fabs(o[i][j].r2 - o[i][j].c2) > 1)
    {
        printf("(");
        i = o[i][j].r2;
        j = o[i][j].c2;
        printf("(%d,%d),(%d,%d)", o[i][j].r1, o[i][j].c1, o[i][j].r2,
o[i][j].c2);
    }
    printf(")\n");
}
```

Screenshot

Q2 > matrix_mult.cpp > ...

```cpp
// Dhruv Rajeshkumar Shah
// 21BCE0611

// Matrix Multiplication

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

struct order
{
    int r1, c1, r2, c2;
};


int main()
{
    // taking input from user
    int n;
    cout << "Enter the number of matrices: ";
    cin >> n;
    int rank[n + 1];
    cout << "Enter the rank of the matrices: ";
    for (int i = 0; i <= n; i++)
        cin >> rank[i];
    order o[n][n];

    // Dynamic programming
    int c[n][n] = {0};
    int i, j, k, diff, q;
    for (i = 0; i < n; i++)
    {
        c[i][i] = 0;
        o[i][i].r1 = i;
        o[i][i].c1 = i;
        o[i][i].r2 = i;
        o[i][i].c2 = i;
    }
    for (diff = 2; diff <= n; diff++)
    {
        for (i = 0; i < n - diff + 1; i++)
        {
            j = i + diff - 1;
            int min = 999999;
            for (k = i; k <= j - 1; k++)
            {
                q = c[i][k] + c[k + 1][j] + rank[i] * rank[k + 1] * rank[j + 1];
                if (q < min)
```

Q2 > C++ matrix_mult.cpp > ...

```
                    {
                        min = q;
                        o[i][j].r1 = i;
                        o[i][j].c1 = k;
                        o[i][j].r2 = k + 1;
                        o[i][j].c2 = j;
                    }
                }
                c[i][j] = min;
            }
        }

        // Printing the dp table
        for (i = 0; i < n; i++)
        {
            for (j = 0; j < n; j++)
            {
                printf("%d ", c[i][j]);
            }
            printf("\n");
        }

        // Printing the cost (number of multiplications)
        printf("The total cost is %d \n", c[0][n - 1]);

        // Printing the order
        printf("The order is \n");
        i = 0;
        j = n - 1;
        if (fabs(o[i][j].r1 - o[i][j].c1) <= 1)
            printf("(%d,%d)", o[i][j].r1, o[i][j].c1);
        while (fabs(o[i][j].r1 - o[i][j].c1) > 1)
        {
            printf("(");
            i = o[i][j].r1;
            j = o[i][j].c1;
            printf("(%d,%d),(%d,%d)", o[i][j].r1, o[i][j].c1, o[i][j].r2, o[i][j].c2);
        }
        printf(")");
        i = 0;
        j = n - 1;
        if (fabs(o[i][j].r2 - o[i][j].c2) <= 1)
            printf("(%d,%d)", o[i][j].r2, o[i][j].c2);
        while (fabs(o[i][j].r2 - o[i][j].c2) > 1)
        {
            printf("(");
            i = o[i][j].r2;
            j = o[i][j].c2;
            printf("(%d,%d),(%d,%d)", o[i][j].r1, o[i][j].c1, o[i][j].r2, o[i][j].c2);
        }
        printf(")\n");
    }
```

OUTPUT

```
~/Coding/DAA/cyclic_1/Q2 cd "/home/dhruv/Coding/DAA/cyclic_1/Q2/" && g++
Enter the number of matrices: 4
Enter the rank of the matrices: 40 20 30 10 30
0 24000 14000 26000
0 0 6000 12000
0 0 0 9000
0 0 0 0
The total cost is 26000
The order is
((0,0),(1,2))(3,3))
```

c) Longest common subsequence

CODE

```cpp
// Dhruv Rajeshkumar Shah
// 21BCE0611

// A dynamic programming based longest common subsequence

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


// Main function
int main()
{
   // Taking input from user
   string s1, s2;

   cout << "Enter the first string: ";
   cin >> s1;

   cout << "Enter the second string: ";
   cin >> s2;

   // Getting the length of the strings
   int m = s1.size();
```

```
int n = s2.size();

int dp[m + 1][n + 1];
int seq[m + 1][n + 1];
for (int i = 0; i < m + 1; i++)
    for (int j = 0; j < n + 1; j++)
    {
        dp[i][j] = 0;
        seq[i][j] = 0;
    }
for (int i = 1; i <= m; i++)
{
    for (int j = 1; j <= n; j++)
    {
        if (s1.at(i - 1) == s2.at(j - 1))
        {
            dp[i][j] = 1 + dp[i - 1][j - 1];
            // To denote diagonal
            seq[i][j] = 3;
        }
        else
        {
            int first = dp[i][j - 1];
            int second = dp[i - 1][j];
            // To denote left and top
            dp[i][j] = first > second ? first : second;
            seq[i][j] = first > second ? 1 : 2;
        }
    }
}
for (int i = 0; i <= m; i++)
{
    for (int j = 0; j <= n; j++)
    {
        cout << dp[i][j] << " ";
    }
    cout << "\n";
}
cout << "seq\n";
for (int i = 0; i <= m; i++)
```

```
    {
        for (int j = 0; j <= n; j++)
        {
            cout << seq[i][j] << " ";
        }
        cout << "\n";
    }
    int last = seq[m][n];
    string substring = "";
    int i = m, j = n;
    while (i != 0 && j != 0)
    {
        if (last == 3)
        {
            substring.append(s1.substr(i - 1, 1));
            i--;
            j--;
        }
        else if (last == 1)
            j--;
        else if (last == 2)
            i--;
        last = seq[i][j];
    }
    cout << "The common subsequence before reversing is    " << substring
<< endl;
    reverse(substring.begin(), substring.end());
    cout << "The common subsequence after reversing is    " << substring <<
endl;
}
```

Screenshot

Q2 > lcs.cpp > ...

```cpp
// Dhruv Rajeshkumar Shah
// 21BCE0611

// A dynamic programming based

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


// Main function
int main()
{
    // Taking input from user
    string s1, s2;

    cout << "Enter the first string: ";
    cin >> s1;

    cout << "Enter the second string: ";
    cin >> s2;

    // Getting the length of the strings
    int m = s1.size();
    int n = s2.size();

    int dp[m + 1][n + 1];
    int seq[m + 1][n + 1];
    for (int i = 0; i < m + 1; i++)
        for (int j = 0; j < n + 1; j++)
        {
            dp[i][j] = 0;
            seq[i][j] = 0;
        }
    for (int i = 1; i <= m; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            if (s1.at(i - 1) == s2.at(j - 1))
            {
                dp[i][j] = 1 + dp[i - 1][j - 1];
                // To denote diagonal
                seq[i][j] = 3;
            }
            else
            {
                int first = dp[i][j - 1];
                int second = dp[i - 1][j];
```

Q2 > C++ lcs.cpp > ...

```cpp
                // To denote left and top
                dp[i][j] = first > second ? first : second;
                seq[i][j] = first > second ? 1 : 2;
            }
        }
    }
    for (int i = 0; i <= m; i++)
    {
        for (int j = 0; j <= n; j++)
        {
            cout << dp[i][j] << " ";
        }
        cout << "\n";
    }
    cout << "seq\n";
    for (int i = 0; i <= m; i++)
    {
        for (int j = 0; j <= n; j++)
        {
            cout << seq[i][j] << " ";
        }
        cout << "\n";
    }
    int last = seq[m][n];
    string substring = "";
    int i = m, j = n;
    while (i != 0 && j != 0)
    {
        if (last == 3)
        {
            substring.append(s1.substr(i - 1, 1));
            i--;
            j--;
        }
        else if (last == 1)
            j--;
        else if (last == 2)
            i--;
        last = seq[i][j];
    }
    cout << "The common subsequence before reversing is    " << substring << endl;
    reverse(substring.begin(), substring.end());
    cout << "The common subsequence after reversing is    " << substring << endl;
}
```

OUTPUT

```
~/Coding/DAA/cyclic_1/Q2  cd "/home/dhruv/Coding/DAA/cyclic_1/
Enter the first string: steady
Enter the second string: speedy
0 0 0 0 0 0 0
0 1 1 1 1 1 1
0 1 1 1 1 1 1
0 1 1 2 2 2 2
0 1 1 2 2 2 2
0 1 1 2 2 3 3
0 1 1 2 2 3 4
seq
0 0 0 0 0 0 0
0 3 1 1 1 1 1
0 2 2 2 2 2 2
0 2 2 3 3 1 1
0 2 2 2 2 2 2
0 2 2 2 2 3 1
0 2 2 2 2 2 3
The common subsequence before reversing is    ydes
The common subsequence after reversing is    sedy
```

d) Subset sum

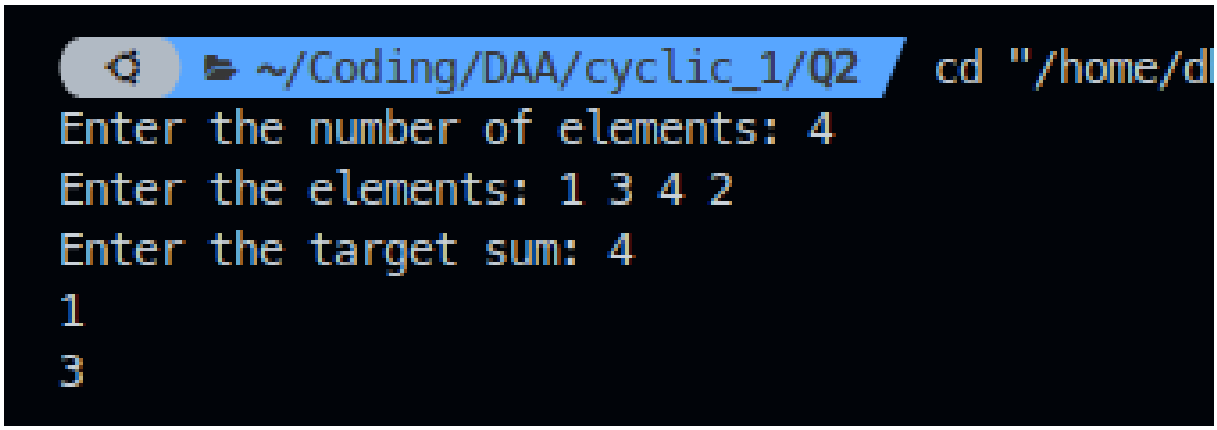
CODE

```cpp
// Dhruv Rajeshkumar Shah
// 21BCE0611

// Subset sum problem

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

stack<int> stck;
bool found = false;

// Function to print the stack
void print()
```

```
{
    stack<int> temp;
    while (!stck.empty())
    {
        temp.push(stck.top());
        stck.pop();
    }
    while (!temp.empty())
    {
        cout << temp.top() << "\n";
        stck.push(temp.top());
        temp.pop();
    }
}

// Function to solve the problem
void solve(int curr, int ind, int a[], int n, int tar)
{
    if (curr > tar)
        return ;
    if (curr == tar)
    {
        found = true;
        print();
        return ;
    }
    for (int i = ind; i < n; i++)
    {
        stck.push(a[i]);
        solve(curr + a[i], i + 1, a, n, tar);
        stck.pop();
    }
    return;
}

int main()
{
    // Taking input from user
    int n, tar;
    int a[n];
```

```
    cout << "Enter the number of elements: ";
    cin >> n;
    cout << "Enter the elements: ";
    for (int i = 0; i < n; i++)
        cin >> a[i];
    cout << "Enter the target sum: ";
    cin >> tar;


    // Printing the solution
    solve(0, 0, a, n, tar);
    if (found == false)
        cout << "No solution";
    return 0;
}
```

OUTPUT



e) Assembly line scheduling

CODE

```
// Dhruv Rajeshkumar Shah
// 21BCE0611

// Assembly line scheduling

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

int main()
{
    // int a[2][6] = {{7, 9, 3, 4, 8, 4}, {8, 5, 6, 4, 5, 7}};
    // int t[2][5] = {{2, 3, 1, 3, 4}, {2, 1, 2, 2, 1}};
    // int e[2] = {2, 4};
    // int x[2] = {3, 2};
    // int n = 6;

    // Taking input from user
    int n;
    cout << "Enter the number of stations: ";
    cin >> n;
    int a[2][n], t[2][n - 1], e[2], x[2];
    cout << "Enter the time taken to enter the station: ";
    cin >> e[0] >> e[1];
    cout << "Enter the time taken to exit the station: ";
    cin >> x[0] >> x[1];
    cout << "Enter the time taken to enter the station: ";
    for (int i = 0; i < 2; i++)
        for (int j = 0; j < n; j++)
            cin >> a[i][j];
    cout << "Enter the time taken to transfer from one station to another:
";
    for (int i = 0; i < 2; i++)
        for (int j = 0; j < n - 1; j++)
            cin >> t[i][j];

    int path[2][n];
    int dp[n][n];
```

```
// time taken to reach station 0 in assembly line 0
dp[0][0] = e[0] + a[0][0];

// time taken to reach station 0 in assembly line 1
dp[1][0] = e[1] + a[1][0];

for (int j = 1; j < n; j++)
{
    int first = dp[0][j - 1] + a[0][j];
    int second = dp[1][j - 1] + t[1][j - 1] + a[0][j];
    if (first <= second)
    {
        dp[0][j] = first;
        path[0][j] = 0;
    }
    else
    {
        dp[0][j] = second;
        path[0][j] = 1;
    }
    first = dp[0][j - 1] + t[0][j - 1] + a[1][j];
    second = dp[1][j - 1] + a[1][j];
    if (first <= second)
    {
        dp[1][j] = first;
        path[1][j] = 0;
    }
    else
    {
        dp[1][j] = second;
        path[1][j] = 1;
    }
}

int last;
if (dp[0][n - 1] + x[0] < dp[1][n - 1] + x[1])
    last = 0;
else
    last = 1;
```

```
    cout << min(dp[0][n - 1] + x[0], dp[1][n - 1] + x[1]) << "\n";
    int i = last;
    cout << "exit should be reached from assembly line " << i << "\n";
    for (int j = n - 1; j > 0; j--)
    {
        i = path[i][j];
        cout << "station " << j << " should be reached from assembly line "
<< i << "\n";
    }
    return 0;
}
```

OUTPUT

```
~/Coding/DAA/cyclic_1/Q2 cd "/home/dhruv/Coding/DAA/cyclic_1/Q2/" && g++ a
Enter the number of stations: 6
Enter the time taken to enter the station: 2 4
Enter the time taken to exit the station: 3 2
Enter the time taken to enter the station: 7 9 3 4 8 4
8 5 6 4 5 7
Enter the time taken to transfer from one station to another: 2 3 1 3 4
2 1 2 2 1
38
exit should be reached from assembly line 0
station 5 should be reached from assembly line 1
station 4 should be reached from assembly line 1
station 3 should be reached from assembly line 0
station 2 should be reached from assembly line 1
station 1 should be reached from assembly line 0
```

f) N-queens problem

CODE

```
// Dhruv Rajeshkumar Shah
// 21BCE0611

// N Queens Problem

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

void solve(int col);
bool issafe(int row, int col);
void print();
int n;
int chess[20][20];
bool found = false;
int main(){
   cout<<"Enter number of Queens: ";
   cin>>n;
   solve(0); // start filling from the 0th column
   if(!found)
       cout<<"No Solution \n";
   return 0;
}

void print(){
   for(int i=0; i<n; i++){
       for(int j=0; j<n; j++){
           if(chess[i][j] == 1)
               cout<<"Q ";
           else
               cout<<"_ ";
       }
       cout<<endl;
   }
   cout<<endl;
}

void solve(int col){
```

```
    if(col == n){  // if col==n , it means that all n queens are placed
        found = true;
        print();
        return;
    }

    //loop through the row so as to try to place the queen right from 0th
row onwards
    for(int i=0; i<n; i++){
        //checking if the current cell is safe to place a queen
        if(issafe(i,col)){
            //set current value to 1 to mean that u have placed a queen
            chess[i][col] = 1;
            //trying to place the next queen in the next column
            solve(col+1);
            //backtrack - reset to 0 to mean that the queen is removed
            chess[i][col] = 0;
        }
    }
}
//to check whether the current position is safe or not
bool issafe(int row, int col){
    //Check the same row but only upto the current cell as the subsequent
columns
    // would not have been filled so far
    //if the current cell =row, col, then the cells in the same row will
have
    //indexes as (row,0),(row,1),(row,2) ....etc Hence (i--) in the loop
    for(int i=col; i>=0; i--){
        if(chess[row][i] == 1)
            return false;
    }
    //check the top diagonal
    // if the current cell=row,col, then the cells in the top diagonals
will have
    //indexes as (row-1,col-1),(row-2,col-2) etc.... Hence,(i--,j--) in the
loop
    for(int i=row, j=col; i>=0 && j>=0; i--,j--){
        if(chess[i][j] == 1)
            return false;
```

```
    }
    //check the bottom diagonal
     // if the current cell=row,col, then the cells in the bottom diagonals
will have
    //indexes as (row+1,col-1),(row+2,col-2) etc.... Hence,(i++,j--) in the
loop
    for(int i=row, j=col; i<n && j>=0; i++,j--){
        if(chess[i][j] == 1)
            return false;
    }

    //return true because it is safe
    return true;
}
```

OUTPUT

```
~/Coding/DAA/cyclic_1/Q2  cd "/home/dhruv/Coding/DAA/cyclic_1/Q2/" && g++
Enter number of Queens: 3
No Solution

~/Coding/DAA/cyclic_1/Q2  cd "/home/dhruv/Coding/DAA/cyclic_1/Q2/" && g++
Enter number of Queens: 4
_ _ Q _
Q _ _ _
_ _ _ Q
_ Q _ _

_ Q _ _
_ _ _ Q
Q _ _ _
_ _ Q _
```

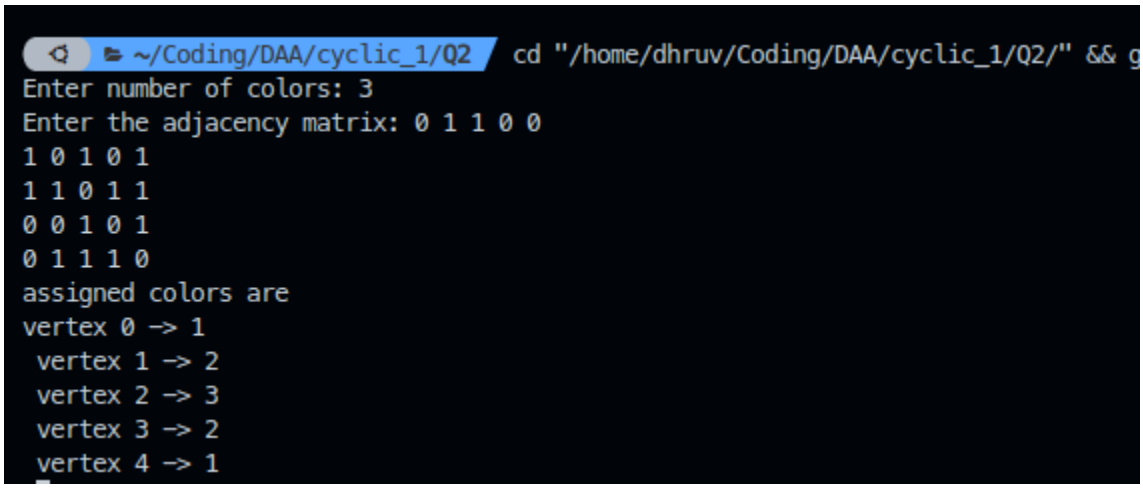
g) Graph coloring

CODE

```
// Dhruv Rajeshkumar Shah
// 21BCE0611

// N Queens Problem

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


bool isSafe(int v, int c, int adj[][5], int color[])
{
   for (int i = 0; i < 5; i++)
       if (adj[v][i] && c == color[i])
           return false;
   return true;
}

bool solve(int v, int m, int adj[][5], int color[])
{
     if (v ==5)
       return true;
   for (int i = 1; i <= m; i++)  //try coloring vertex v with different
colors
   {
       if (isSafe(v,i, adj, color)) {
           color[v] = i;
           if (solve( v + 1, m, adj, color) == true)
               return true;
           color[v] = 0; //backtrack
       }
   }
   return false;
}
 void print(int color[])
{
   cout << "assigned colors are \n";
   for (int i = 0; i < 5; i++)
```

```
        cout << "vertex "<<i <<" -> " <<color[i] << "\n ";
}
int main()
{
   // Take input from user
   int m;
   cout << "Enter number of colors: ";
   cin >> m;

   // Initialize the adjacency matrix
   int adj[5][5];
   cout << "Enter the adjacency matrix: ";
   for (int i = 0; i < 5; i++)
       for (int j = 0; j < 5; j++)
           cin >> adj[i][j];

   // Initialize the color array
   int color[5];

   for (int i = 0; i < 5; i++)
       color[i] = 0;
   if (solve(0, m, adj, color) == false)
       cout << "Solution does not exist";
   else
     print(color);
   return 0;
}
```

OUTPUT


~/Coding/DAA/cyclic_1/Q2
cd "/home/dhruv/Coding/DAA/cyclic_1/Q2/" && g
Enter number of colors: 3
Enter the adjacency matrix: 0 1 1 0 0
1 0 1 0 1
1 1 0 1 1
0 0 1 0 1
0 1 1 1 0
assigned colors are
vertex 0 -> 1
vertex 1 -> 2
vertex 2 -> 3
vertex 3 -> 2
vertex 4 -> 1

BCSE204P - Design and Analysis of Algorithms Lab

Cycle sheet - 2

Name: Dhruv Rajeshkumar Shah
Reg no: 21BCE0611

Q1. Write the programs for the following algorithms -

a) Naive pattern matching
b) KMP algorithm
c) Robin Karp algorithm

Ans.

a) Naive Pattern Matching

CODE

```cpp
// Dhruv Rajeshkumar Shah
// 21BCE0611

// Naive pattern matching

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

// Function to find the pattern
void findPattern(string text, string pattern)
{
    int n = text.length();
    int m = pattern.length();
    for (int i = 0; i <= n - m; i++)
    {
        int j;
        for (j = 0; j < m; j++)
            if (text[i + j] != pattern[j])
                break;
        if (j == m)
            cout << "Pattern found at index " << i << ".\n";
    }
}

// Main function
int main()
{
```

```
    // Taking input from user
    string text, pattern;
    cout << "Enter the text: ";
    cin >> text;
    cout << "Enter the pattern: ";
    cin >> pattern;

    findPattern(text, pattern);
    return 0;
}
```

OUTPUT

```
~/C/DAA/cyclic_2/Q1  main !2 ?4  cd "/Users/dhruvshah/Coding
/DAA/cyclic_2/Q1/" && g++-12 naive_pattern.cpp -o ../out/naive_pattern &&
"/Users/dhruvshah/Coding/DAA/cyclic_2/Q1/"/../out/naive_pattern
Enter the text: hellodhruvwhatsupdhruv
Enter the pattern: dhruv
Pattern found at index 5.
Pattern found at index 17.
```

b) KMP algorithm

CODE

```
// Dhruv Rajeshkumar Shah
// 21BCE0611

// KMP Algorithm

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

// Function to calculate the LPS table
void calcLPS(string p,int LPS[]){
    LPS[0] = 0;
    int i = 0,j=1;
    while (j<p.length()){
        if(p[i]==p[j])
```

```
        {
            LPS[j] = i+1;
            i++; j++;
        }
        else{
            if(i==0){
                LPS[j] = 0;
                j++;
            }
            else{
                i = LPS[i-1];
            }
        }
    }
}

// Main function
int main()
{
    string s, p;

    // Taking input from user
    cout << "Enter the string: ";
    cin >> s;
    cout << "Enter the pattern: ";
    cin >> p;

    int slen = s.length();
    int plen = p.length();
    int LPS[plen];
    calcLPS(p,LPS); // to build the LPS table
    int i=0,j=0;
    while(i<slen){
        if(p[j]==s[i]){i++;j++;} // If there is a match, proceed to check
the remaining characters
        if (j == plen) {
```

```
            cout<<i - plen <<' ';  // to print the index of the string
where the pattern matches
            j = LPS[j - 1];
        }
        else if (i < slen && p[j] != s[i]) {
            if (j == 0)
                i++;
            else
                j = LPS[j - 1];  //if there is a mismatch after a few
matched characters,
        }                          // proceed to check only from the LPS of
the previously
    }                             // matched character
    cout<<endl;
    return 0;
}
```

OUTPUT

```
 ~/Coding/DAA/cyclic_2/Q1  main !2 ?4  cd "/Users/dhruvshah/Coding/DAA
s/dhruvshah/Coding/DAA/cyclic_2/Q1/"../out/KMP
Enter the string: hellodhruvwhatsupdhruv
Enter the pattern: dhruv
5 17
```

c) Robin Karp Algorithm

CODE

```
// Dhruv Rajeshkumar Shah
// 21BCE0611

// Robin Karp Algorithm

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

// Modulus
```

```
#define prime 13

int main()
{
   string s,p;

   // Taking input from user
   cout<<"Enter the string: ";
   cin>>s;
   cout<<"Enter the pattern: ";
   cin>>p;



   int plen = p.length();
   int slen = s.length();
   int ph=0,sh=0,h=1,maxchar=10;
 // we assume that the string and the pattern will contain only 10
different
 //characters from A to J
   for(int i=0;i<plen-1;i++)
     h=(h*maxchar)%prime;  // to calculate pow(maxchar,plen-1)%prime
   for(int i=0;i<plen;i++)
   {
     ph=(ph*maxchar+p[i]-65+1)%prime;//to calculate hash value of the
pattern
     sh=(sh*maxchar+s[i]-65+1)%prime;//to calculate hash value of the
first window
   }
   cout<<"Hash value of the pattern "<<p<<" is "<<ph<<endl;
   cout<<"Hash value of the window "<<s.substr(0,plen)<<" is "<<sh<<endl;
   for(int i=0;i<=slen-plen;i++)
   {
     if(ph==sh) // only if the pattern's hash value & window's hash value
match, we check the actual characters
     {
       int flag=0,count=0;
```

```
      for(int j=0;j<plen;j++)
      {
      if(s[i+j]==p[j])
      {
    flag=1;
        count++;
      }
      else
        break;
      }
      if(count==plen)
      {
          cout<<endl<<"Pattern occurs at index: "<<i<<endl<<endl;
          //continue;
      }
    }
    //int nextchar=i+1;
    if(i<slen-plen) //if the string contains more characters to be
compared
    {
      sh=((sh-(s[i]-65+1)*h)*maxchar+(s[i+plen]-65+1))%prime;
      while(sh<0)
        sh+=prime;
      cout<<"Hash value of the window "<<s.substr(i+1,plen)<<" is
"<<sh<<endl;
    }
  }//end of for
  return 0;
}
```

OUTPUT

```
~/Coding/DAA main !2 ?4 cd "/Users/dhruvshah/Coding/DAA/cyclic_2/Q1
nnerFile && "/Users/dhruvshah/Coding/DAA/cyclic_2/Q1/"../out/tempCodeRunnerFile
Enter the string: hellodhruvwhatsupdhruv
Enter the pattern: dhruv
Hash value of the pattern dhruv is 10
Hash value of the window hello is 4
Hash value of the window ellod is 7
Hash value of the window llodh is 1
Hash value of the window lodhr is 1
Hash value of the window odhru is 4
Hash value of the window dhruv is 10

Pattern occurs at index: 5

Hash value of the window hruvw is 11
Hash value of the window ruvwh is 3
Hash value of the window uvwha is 6
Hash value of the window vwhat is 4
Hash value of the window whats is 5
Hash value of the window hatsu is 0
Hash value of the window atsup is 5
Hash value of the window tsupd is 6
Hash value of the window supdh is 9
Hash value of the window updhr is 1
Hash value of the window pdhru is 7
Hash value of the window dhruv is 10

Pattern occurs at index: 17
```

Q2. Write the programs for the following algorithms -

a) Bellman Ford
b) Edmund Karp
c) Floyd Warshall
d) Push relabel

Ans.

a) Bellman Ford

CODE

```
// Dhruv Rajeshkumar Shah
// 21BCE0611


// Bellman Ford Algorithm
```

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

struct edge
{
   int source,dest,wt;
};

// Main function
int main()
{
   int v,e,start;

   // Taking input from user
   cout<<"Enter the number of vertices: ";
   cin>>v;
   cout<<"Enter the number of edges: ";
   cin>>e;

   struct edge edges[e];
   float dist[v];
   cout<<"Enter the source,dest and weight of each edge: \n";
   for(int i=0;i<e;i++)
     cin>>edges[i].source>>edges[i].dest>>edges[i].wt;

   cout<<"Enter the start node: ";
   cin>>start;

   for(int i=0;i<v;i++)
     dist[i]=INT_MAX;
   dist[start-1]=0;
   for(int i=0;i<v-1;i++)
   {
       for(int j=0;j<e;j++)
```

```
        {
            int u=edges[j].source;
            int v=edges[j].dest;
            int w=edges[j].wt;
            if (dist[v-1]>dist[u-1]+w)
               dist[v-1]=dist[u-1]+w;
        }
    }
    int flag=0; // this loop is to check if the graph contains any negative
weight cycle
    for(int j=0;j<e;j++)
    {
            int u=edges[j].source;
            int v=edges[j].dest;
            int w=edges[j].wt;
            if (dist[v-1]>dist[u-1]+w)
            { cout<<"the graph contains negative weight cycle; so no
solution\n";
              flag=1;
              break;
            }
    }
    if(flag==0)
    cout<<"The shortest distance from the start node to all other nodes is:
\n";
    cout<<0<<" "<<0<<endl;
    for(int i=1;i<v;i++)
      cout<<i<<" "<<dist[i-1]<<endl;
    return 0;
}
```

OUTPUT

```
~/C/DAA/cyclic_2/Q2  main !2 ?4  cd "/Users/dhruvshah/Coding/DAA/cycl
ord && "/Users/dhruvshah/Coding/DAA/cyclic_2/Q2/"../out/bellman_ford
Enter the number of vertices: 6
Enter the number of edges: 9
Enter the source,dest and weight of each edge:
0 1 6
0 2 4
0 3 5
3 2 -2
2 1 -2
1 4 -1
3 5 -1
2 4 3
4 5 3
Enter the start node: 0
The shortest distance from the start node to all other nodes is:
0 0
1 1
2 3
3 5
4 0
5 3
```

b) Edmund Karp algorithm

CODE

```cpp
// Dhruv Rajeshkumar Shah
// 21BCE0611

// Edmond Karp Algorithm

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

int n;
typedef struct Node
{
 int id, state, prev;
} node;

int res[100][100];
```

```
// Function to find the path
bool DFS(node vert[], node source, node sink)
{
 node u;
 stack<node> s;
 for (int i = 0; i < n; i++)
 {
   vert[i].state = 0;
 }
 vert[source.id].state = 0;
 vert[source.id].prev = -1;

 s.push(source);

 while (!s.empty())
 {
   u = s.top();
   s.pop();

   for (int i = 0; i < n; i++)
   {
     if (res[u.id][i] && !vert[i].state)
     {
       s.push(vert[i]);
       vert[i].state = 1;
       vert[i].prev = u.id;
     }
   }
 }

 return (vert[sink.id].state == 1);
}

// Main function
int main()
```

```
{
 // Taking input from user
 cout << "Enter number of nodes: ";
 cin >> n;
 cout << "Enter the adjacency Matrix: "
      << endl;
 for (int i = 0; i < n; i++)
 {
   for (int j = 0; j < n; j++)
   {
     cin >> res[i][j];
   }
 }

 node vert[n], source, sink;

 for (int i = 0; i < n; i++)
 {
   vert[i].id = i;
 }

 source.id = 0;
 sink.id = n - 1;

 int maxFlow = 0;

 while (DFS(vert, source, sink))
 {
   int augFlow = INT_MAX;
   for (int v = sink.id; v != source.id; v = vert[v].prev)
   {
     int u = vert[v].prev;
     augFlow = augFlow < res[u][v] ? augFlow : res[u][v];
   }
   for (int v = sink.id; v != source.id; v = vert[v].prev)
   {
```

```
    int u = vert[v].prev;
    res[u][v] -= augFlow;
    res[v][u] += augFlow;
  }
  maxFlow += augFlow;
}

cout << "Maxflow : " << maxFlow << endl;

return 0;
}
```

OUTPUT

```
~/C/DAA/cyclic_2/Q2 main !2 ?4 cd "/Users/dhruvshah/Coding/DAA/cycl
p && "/Users/dhruvshah/Coding/DAA/cyclic_2/Q2/"../out/edmond_karp
Enter number of nodes: 6
Enter the adjacency Matrix:
0 10 0 10 0 0
0 0 4 2 8 0
0 0 0 0 0 10
0 0 0 0 9 0
0 0 6 0 0 10
0 0 0 0 0 0
Maxflow : 19
```

c) Floyd Warshall

CODE

```
// Dhruv Rajeshkumar Shah
// 21BCE0611

// Floyd Warshall Algorithm

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

#define MAX 20
```

```
struct path
{
   char a[MAX];
   int len;
} route[MAX][MAX]; // route matrix uses this structure
int cost[MAX][MAX];
int n;

// Main function
int main()
{
   int i, j, k, c, x, y;
   int max_edges, origin, destin;
   cout << "Enter number of nodes : ";
   cin >> n;
   for (i = 1; i <= n; i++)
   {
       for (j = 1; j <= n; j++)
       {
           route[i][j].len = 0;
           cost[i][j] = 999999; // initialize all entries in cost[][] to 0
       }
   }
   max_edges = n * (n - 1);
   for (i = 1; i <= max_edges; i++)
   {
       cout << "Enter edge ( 0 0 to quit ) : " << endl;
       cin >> origin >> destin;
       if ((origin == 0) && (destin == 0))
           break;
       if (origin > n || destin > n || origin <= 0 || destin <= 0)
       {
           cout << "Invalid edge!\n";
           i--;
       }
       else
```

```
        {
            cout << "\nEnter the cost: ";
            cin >> c;                   // only for a valid edge, read its
cost 'c' and store it in the appropriate
            cost[origin][destin] = c; // index of cost[ ][ ]
            route[origin][destin].a[route[origin][destin].len++] = origin +
48;
            route[origin][destin].a[route[origin][destin].len++] = destin +
48; // for storing origin node &
route[origin][destin].a[route[origin][destin].len]='\0';
//destination as characters in route[][]
        }
    }
    // display
    cout << "\nInitial Cost Matrix\n";
    for (i = 1; i <= n; i++)
    {
        for (j = 1; j <= n; j++)
            cout << cost[i][j] << " ";
        cout << "\n";
    }
    for (i = 1; i <= n; i++)      // 'i' index generates the intermediate
vertex
    {                             // 'j' index generates the row number of
the matrices
        for (j = 1; j <= n; j++) // 'j' index generates the row number of
the matrices
        {
            for (k = 1; k <= n; k++) // 'k' index generates the column
number of the matrices
            {
                if (cost[j][k] > cost[j][i] + cost[i][k]) // if the route
via vertex 'i' is shorter, update the cost of that route
                {
                    cost[j][k] = cost[j][i] + cost[i][k]; // in the cost
matrix.
```

```
                    for (x = 0; x < route[j][i].len; x++)
                        route[j][k].a[x] = route[j][i].a[x]; // first copy
contents of route[j][i] to route[j][k]
                    for (y = 1; y < route[i][k].len; y++)
                        route[j][k].a[x - 1 + y] = route[i][k].a[y]; //
next append contents of route[i][k] to route[j][k]
                    route[j][k].a[x - 1 + y] = '\0';
                    route[j][k].len = route[j][i].len + route[i][k].len -
1;
                }
            }
        }
    }
    cout << "\nFinal Cost Matrix\n";
    for (i = 1; i <= n; i++)
    {
        for (j = 1; j <= n; j++)
            cout << cost[i][j] << "   ";
        cout << "\n";
    }
    cout << "\nFinal Route Matrix\n";
    for (i = 1; i <= n; i++)
    {
        for (j = 1; j <= n; j++)
            cout << route[i][j].a << "  ";
        cout << "\n";
    }
    return 0;
}
```

OUTPUT

```
~/C/DAA/cyclic_2/Q2  main !2 ?4  cd "/Users/dhruvshah/Coding/DAA/cycl
arshall && "/Users/dhruvshah/Coding/DAA/cyclic_2/Q2/"../out/floyd_warshall
Enter number of nodes : 6
Enter edge ( 0 0 to quit ) :
1 2

Enter the cost: 6
Enter edge ( 0 0 to quit ) :
1 3

Enter the cost: 4
Enter edge ( 0 0 to quit ) :
1 4

Enter the cost: 5
Enter edge ( 0 0 to quit ) :
4 3

Enter the cost: -2
Enter edge ( 0 0 to quit ) :
3 2

Enter the cost: -2
Enter edge ( 0 0 to quit ) :
2 5

Enter the cost: -1
Enter edge ( 0 0 to quit ) :
4 6

Enter the cost: -1
Enter edge ( 0 0 to quit ) :
3 5

Enter the cost: 3
Enter edge ( 0 0 to quit ) :
5 6

Enter the cost: 3
Enter edge ( 0 0 to quit ) :
0 0

Initial Cost Matrix
999999 6 4 5 999999 999999
999999 999999 999999 999999 -1 999999
999999 -2 999999 999999 3 999999
999999 999999 -2 999999 999999 -1
999999 999999 999999 999999 999999 3
999999 999999 999999 999999 999999 999999
```

```
Final Cost Matrix
999999   1   3   5   0   3
999998   999994   999996   999998   -1   2
999996   -2   999994   999996   -3   0
999994   -4   -2   999994   -5   -2
999999   999995   999997   999999   999994   3
999999   999995   999997   999999   999994   999997

Final Route Matrix
  1432  143  14  14325  143256
25  252  25  25  25  256
325  32  325  325  325  3256
4325  432  43  4325  4325  43256
  32  3    325  56
  32  3    325  3256
```

d) Push relabel Algorithm

CODE

```
// Dhruv Rajeshkumar Shah
// 21BCE0611

// Push Relabel Algorithm

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

struct Vertex
{
   int h;
   int eflow;
};

int v, e;
int **cap, **flow;
struct Vertex *vert;

// Function to find the active node
int getactivenode(int source, int sink)
{
```

```
    for (int i = 1; i < v - 1; i++)
    {
        if (vert[i].eflow > 0)
        {
            for (int j = 0; j < v; j++)
            {
                if (cap[i][j] != 0 || flow[i][j] != 0)
                {
                    if (cap[i][j] != flow[i][j])
                        return i;
                }
            }
        }
    }
    return -1;
}

// Function to find the preflow
void preflow(int s)
{
    vert[s].h = v;
    for (int i = 0; i < v; i++)
    {
        if (i != s && cap[s][i] != 0)
        {
            flow[s][i] = cap[s][i];
            flow[i][s] = -cap[s][i];
            vert[i].eflow += cap[s][i];
        }
    }
}

// Function to push the flow
bool push(int u)
{
    for (int i = 0; i < v; i++)
```

```
    {
        if (cap[u][i] != 0 || flow[u][i] != 0)
        {
            if (flow[u][i] == cap[u][i])
                continue;
            if (vert[u].h > vert[i].h)
            { // minimum of residual capacity[u][i] and eflow(u)
                int Flow = cap[u][i] - flow[u][i] < vert[u].eflow ?
cap[u][i] - flow[u][i] : vert[u].eflow;
                vert[u].eflow -= Flow;
                vert[i].eflow += Flow;
                flow[u][i] += Flow;
                flow[i][u] -= Flow;
                return true;
            }
        }
    }
    return false;
}

// Function to relabel the nodes
void relabel(int u)
{
    int minh = INT_MAX;
    for (int i = 0; i < v; i++)
    {
        if (cap[u][i] != 0 || flow[u][i] != 0)
        {
            if (cap[u][i] == flow[u][i])
                continue;
            if (vert[i].h < minh)
            {
                minh = vert[i].h;
            }
        }
    }
```

```
    vert[u].h = minh + 1;
}

// Function to find the maximum flow
int maxFlow(int source, int sink)
{
    preflow(source);
    cout << "\nInitial capacity\n";
    for (int i = 0; i < v; i++)
    {
        for (int j = 0; j < v; j++)
            cout << cap[i][j] << " ";
        cout << "\n";
    }
    int u = getactivenode(source, sink);
    while (u != -1)
    {
        if (!push(u))
            relabel(u);
        u = getactivenode(source, sink);
    }
    return vert[sink].eflow;
}

// Main function
int main()
{
    // Taking input from user
    cout << "Enter no. of vertices : ";
    cin >> v;
    // v=6;
    cout << "Enter no. of edges : ";
    cin >> e;
    // e=10;

    vert = new Vertex[v];
```

```
    cap = new int *[v];
    for (int i = 0; i < v; i++)
        cap[i] = new int[v];
    flow = new int *[v];
    for (int i = 0; i < v; i++)
        flow[i] = new int[v];
    cout << "Enter edges and their capacity : \n";
    for (int i = 0; i < v; i++)
    {
        vert[i].h = 0;
        vert[i].eflow = 0;
        for (int j = 0; j < v; j++)
        {
            flow[i][j] = 0;
            cap[i][j] = 0;
        }
    }
    int x, y, c;
    for (int i = 0; i < e; i++)
    {
        cin >> x >> y >> c;
        cap[x][y] = c;
    }

    // Taking source and sink
    cout << "Enter source and sink : ";
    int source, sink;
    cin >> source >> sink;

    // Finding the maximum flow
    cout << "MaxFlow : " << maxFlow(source, sink)<<endl;
    cout << "Final flow\n";
    for (int i = 0; i < v; i++)
    {
        for (int j = 0; j < v; j++)
            cout << flow[i][j] << " ";
```

```
        cout << endl;
    }
    return 0;
}
```

OUTPUT

```
~/C/DAA/cyclic_2/Q2  main !2 ?4  cd "/Users/dhruvshah/Coding/DAA/cycl
bel && "/Users/dhruvshah/Coding/DAA/cyclic_2/Q2/"../out/push_relabel
Enter no. of vertices : 5
Enter no. of edges : 7
Enter edges and their capacity :
0 1 3
0 2 4
1 2 1
1 3 1
2 3 4
2 4 2
3 4 6
Enter source and sink : 0 4

Initial capacity
0 3 4 0 0
0 0 1 1 0
0 0 0 4 2
0 0 0 0 6
0 0 0 0 0
MaxFlow : 6
Final flow
0 2 4 0 0
-2 0 1 1 0
-4 -1 0 4 1
0 -1 -4 0 5
0 0 -1 -5 0
```

BCSE204P - Design and Analysis of Algorithms Lab

Cycle sheet - 3

Name: Dhruv Rajeshkumar Shah
Reg no: 21BCE0611

Q1. Write the programs for the following algorithms -

a) Line segment intersection algorithm

b) Jarvis March algorithm for Convex Hull

Ans.

a) Line segment intersection algorithm

CODE

```
// Dhruv Rajeshkumar Shah
// 21BCE0611

// Line segment intercept

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

typedef struct point{
    int x, y;
} point;

bool onLine(point p, point q, point r) {
    // Check if the point q lies on the line segment pr
    if (q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) && q.y <= max(p.y,
r.y) && q.y >= min(p.y, r.y)) {
        return true;
    }
    return false;
}

int dir(point p, point q, point r) {
    // Check the direction of the point r with respect to the line segment
pq
    int val = (q.y - p.y) * (r.x - q.x) - (q.x - p.x) * (r.y - q.y);
    if (val == 0) {
        return 0;
    }
```

```
    return (val > 0) ? 1 : 2;
}

bool intersect(point p1, point q1, point p2, point q2) {
    // Check if the line segment intersects with the line segment
    int d1 = dir(p1, q1, p2);
    int d2 = dir(p1, q1, q2);
    int d3 = dir(p2, q2, p1);
    int d4 = dir(p2, q2, q1);
    if (d1 != d2 && d3 != d4) {
        return true;
    }
    if (d1 == 0 && onLine(p1, p2, q1)) {
        return true;
    }
    if (d2 == 0 && onLine(p1, q2, q1)) {
        return true;
    }
    if (d3 == 0 && onLine(p2, p1, q2)) {
        return true;
    }
    if (d4 == 0 && onLine(p2, q1, q2)) {
        return true;
    }
    return false;
}


int main() {
    point p1, q1, p2, q2;
    cout << "Enter starting point of line 1: ";
    cin >> p1.x >> p1.y;
    cout << "Enter ending point of line 1: ";
    cin >> q1.x >> q1.y;
    cout << "Enter starting point of line 2: ";
    cin >> p2.x >> p2.y;
```

```
    cout << "Enter ending point of line 2: ";
    cin >> q2.x >> q2.y;

    if (intersect(p1, q1, p2, q2)) {
        cout << "Lines intersect"<<endl;
    } else {
        cout << "Lines do not intersect"<<endl;
    }

    return 0;
}
```

OUTPUT

```
 ~/C/DAA/cyclic_3/Q2  main !2 ?5  cd "/Users/dhruvsha
c && "/Users/dhruvshah/Coding/DAA/cyclic_3/Q1/"/../out/line_interc
Enter starting point of line 1: 0 0
Enter ending point of line 1: 5 5
Enter starting point of line 2: 0 5
Enter ending point of line 2: 5 0
Lines intersect

 ~/C/DAA/cyclic_3/Q1  main !2 ?5  cd "/Users/dhruvsha
c && "/Users/dhruvshah/Coding/DAA/cyclic_3/Q1/"/../out/line_interc
Enter starting point of line 1: 0 0
Enter ending point of line 1: 0 7
Enter starting point of line 2: 3 7
Enter ending point of line 2: 3 10
Lines do not intersect
```

b) Jarvis March algorithm for Convex Hull

CODE

```
// Dhruv Rajeshkumar Shah
// 21BCE0611

// Jarvis March algorithm for Convex Hull
```

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

// Point class
struct Point
{
	int x;
	int y;
};
// To find orientation of ordered triplet (p, q, r).
int direction(Point p, Point q, Point r)
{
   int val =  (r.x - p.x)*(q.y - p.y)  - (q.x - p.x) * (r.y - p.y);
   if (val == 0)
       return 0; // collinear
   return (val > 0) ? 1 : 2; // clock or counterclock wise
}
// Prints convex hull of a set of n points.
void convexHull(Point points[], int n)
{
   if (n < 3)
       return;
   bool included[n];
   for (int i = 0; i < n; i++)
       included[i] = false;
   int left = 0;
   for (int i = 1; i < n; i++)
       if (points[i].x < points[left].x)
           left = i;
   int prev = left, curr;
   do
   {
       // Find the point 'curr' such that direction(prev, curr, i) is
       // anti-clockwise for all points 'i'
       curr = (prev + 1) % n;
```

```
        for (int i = 0; i < n; i++)
            if (direction(points[prev], points[curr], points[i]) == 2)
                curr = i;
        included[prev] = true;
        prev = curr;

    }
    while (prev != left);
    for (int i = 0; i < n; i++)
    {
        if (included[i] != false)
            cout << "(" << points[i].x << ", " << points[i].y << ")\n";
    }
}
int main()
{
    // Taking input from user
    int n;
    cout<<"Enter the number of points : ";
    cin>>n;
    Point points[n];
    cout<<"Enter the points : ";
    for(int i=0;i<n;i++)
    {
        cin>>points[i].x>>points[i].y;
    }

    // Print the result
    cout << "The points in the convex hull are: \n";
    convexHull(points, n);
    return 0;
}
```

OUTPUT

```
~/C/DAA/cyclic_3/Q1 main !2 ?5 cd "/Users/dhruvsha
rch && "/Users/dhruvshah/Coding/DAA/cyclic_3/Q1/"/../out/jarvis_ma
Enter the number of points : 7
Enter the points :
0 3
2 2
1 1
2 1
3 0
0 0
3 3
The points in the convex hull are:
(0, 3)
(3, 0)
(0, 0)
(3, 3)
```

Q2. Write the programs for the following algorithms -

a) Randomized Quick sort
b) Randomized Hiring Algorithm

Ans.

a) Randomized Quick sort

CODE

```cpp
// Dhruv Rajeshkumar Shah
// 21BCE0611

// Randomized Quick Sort

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

// Function to sort array using randomized quick sort
void quick(int a[], int left, int right)
```

```
{
   int temp;
   if (left >= right)
       return;

   // Randomizing the pivot
   srand(time(NULL));
   int pivot = (rand() % (right - left + 1)) + left;

   temp = a[left];
   a[left] = a[pivot];
   a[pivot] = temp;
   pivot = left;
   int l = left;
   int r = right;
   while (l < r)
   {
       while (a[r] > a[pivot])
           r--;
       while (a[l] <= a[pivot])
           l++;
       if (l < r)
       {
           temp = a[l];
           a[l] = a[r];
           a[r] = temp;
       }
   }
   temp = a[pivot];
   a[pivot] = a[r];
   a[r] = temp;
   quick(a, left, r - 1);
   quick(a, r + 1, right);
}

// Main function
```

```cpp
int main()
{
   // Taking input from user
   int n, i;
   cout << "Enter the number of elements: ";
   cin >> n;
   int a[n];
   cout << "Enter the elements: ";
   for (i = 0; i < n; i++)
       cin >> a[i];

   // Calling quick sort function
   quick(a, 0, n - 1);

   // Printing the sorted array
   cout << "\nSorted array is \n";
   for (i = 0; i < n; i++)
       cout << a[i] << " ";
   cout<<endl;
   return 0;
}
```

OUTPUT

```
~/C/DAA/cyclic_3/Q2  main !2 ?5  cd "/Users/dhruvsha
pCodeRunnerFile && "/Users/dhruvshah/Coding/DAA/cyclic_3/Q2/"/../o
Enter the number of elements: 8
Enter the elements: 9 4 2 7 3 1 5 8

Sorted array is
1 2 3 4 5 7 8 9
```

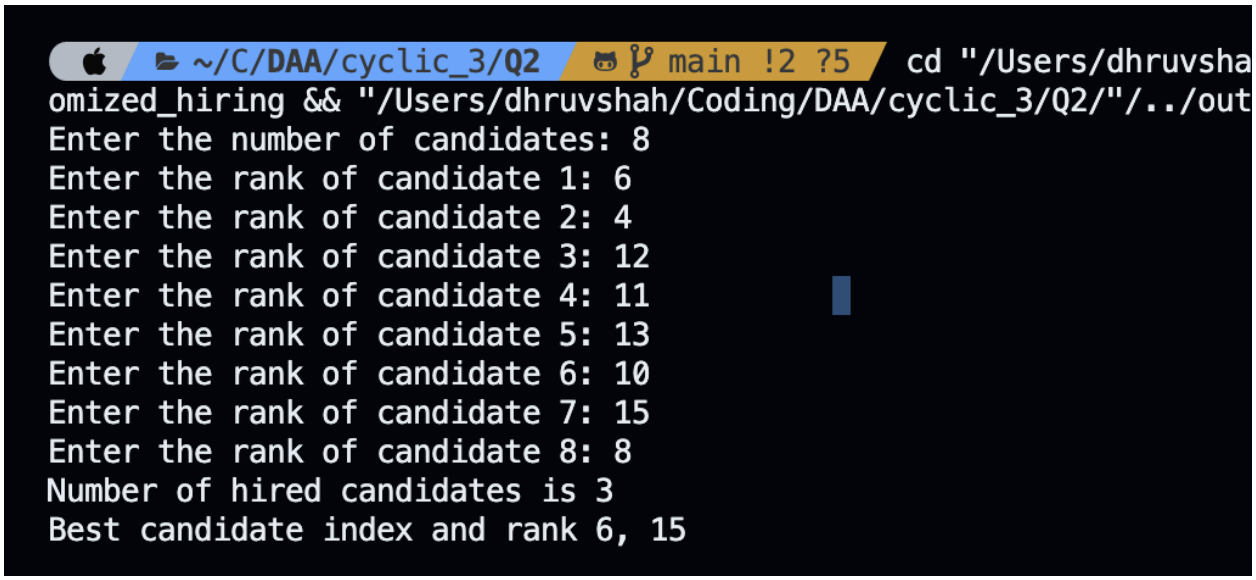
b) Randomized Hiring problem

CODE

```cpp
// Dhruv Rajeshkumar Shah
// 21BCE0611

// Randomized Hiring Algorithm

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

// Structure to store the rank and interview status of a candidate
struct candidate
{
   int rank, interview_status;
};

// Main function
int main()
{
   // Taking input from user
   int n, no_hired = 0;
   cout << "Enter the number of candidates: ";
   cin >> n;
   struct candidate cand[n];
   for (int i = 0; i < n; i++)
   {
       cout << "Enter the rank of candidate " << i + 1 << ": ";
       cin >> cand[i].rank;
       cand[i].interview_status = 0;
   }

   int best = -1;
   srand(time(0));
   int index;
   for (int i = 0; i < n; i++)
```

```
    {
        do
        {
            index = rand() % n; // since upper-lower=n-0=n
        } while (cand[index].interview_status == 1);
        // if the same number which was generated already is generated
again, call rand() again
        cand[index].interview_status = 1;
        if (best == -1)
        {
            best = index;
            no_hired += 1;
        }
        else if (cand[index].rank > cand[best].rank)
        {
            best = index;
            no_hired += 1;
        }
    }
    cout << "Number of hired candidates is " << no_hired << endl;
    cout << "Best candidate index and rank " << best << ", " <<
cand[best].rank << endl;
    return 0;
}
```

OUTPUT


~/C/DAA/cyclic_3/Q2
main !2 ?5
cd "/Users/dhruvsha
omized_hiring && "/Users/dhruvshah/Coding/DAA/cyclic_3/Q2/"../out
Enter the number of candidates: 8
Enter the rank of candidate 1: 6
Enter the rank of candidate 2: 4
Enter the rank of candidate 3: 12
Enter the rank of candidate 4: 11
Enter the rank of candidate 5: 13
Enter the rank of candidate 6: 10
Enter the rank of candidate 7: 15
Enter the rank of candidate 8: 8
Number of hired candidates is 3
Best candidate index and rank 6, 15