Assignment 3

**Question 1:** Write a program to implement the KMP string matching algorithm

**Algorithm**

Start

Algorithm for method main():

Start

Step 1: Input the pattern and the text to be searched and store them in string variables txt and pat

Step 2: search(pat, txt)

Stop

Algorithm for subroutine search(pat, txt):

Start

Step 1: M <-- pat.length(), N <-- txt.length()

Step 2: lps[ M ]

Step 3: calc(pat, M, lps)

Step 4: i = 0, j = 0

Step 5: while i < N

1. if pat[ j ] = txt[ i ]
   1. j++, i++
2. if j = M
   1. print(“Found pattern at index ” + (i – j))
   2. j <-- lps[ j-1 ]
3. else if i < N AND pat[j] != txt[i]
   1. if j != 0
      1. j <-- lps[ j-1 ]
   2. else
      1. i <-- i+1

Stop

Algorithm for subroutine calc(pat, M, lps)

Start

Step 1: len <-- 0, lps[0] <-- 0, i <-- 1

Step 2: while i < M

1. if pat[i] == pat[len]
   1. len <-- len + 1
   2. lps[i] <-- len
   3. i <-- i + 1
2. else
   1. if len != 0
      1. len <-- lps[len – 1]
   2. else
      1. lps[i] <-- 0
      2. i <-- i + 1

Stop

Stop

**Code:**

#include <iostream>

#include <vector>

#include <string>

void calc(const std::string &pat, int M, std::vector<int> &lps)

{

int len = 0;

lps[0] = 0;

int i = 1;

while (i < M)

{

if (pat[i] == pat[len])

{

len++;

lps[i] = len;

i++;

}

else

{

if (len != 0)

{

len = lps[len-1];

}

else

{

lps[i] = 0;

i++;

}

}

}

}

void search(const std::string &pat, const std::string &txt)

{

int M = pat.length();

int N = txt.length();

std::vector<int> lps(M);

calc(pat, M, lps);

int i = 0;

int j = 0;

while (i < N)

{

if (pat[j] == txt[i])

{

j++;

i++;

}

if (j == M)

{

std::cout<<"Found pattern at index "<<i-j<<'\n';

j = lps[j-1];

}

else if (i < N && pat[j] != txt[i])

{

if (j != 0)

j = lps[j-1];

else

i = i+1;

}

}

}

int main()

{

std::string txt, pat;

std::cout<<"Enter the text and the pattern to be searched\n";

std::cin>>txt>>pat;

search(pat, txt);

return 0;

}

**Input:** ABABDABACDABABCABAB

ABABCABAB

**Output:** Found pattern at index 10

**Question 2:** Given a value V and infinite supply of coins of m-denominations {C1=1 < C2<C3<…..<Cm}, we want to make change for Rs. V. Apply DP strategy to find out minimum number of coins to make the change?

**Algorithm:**

Start

Step 1: Accept the number of available coins and store it in an integer variable n

Step 2: Declare an integer array coins of length n and input elements in the arrat

Step 3: Input the value and store it in an integer variable val

Step 4: Initialise an integer arrat ans of length val + 1 and initialise all the elements with 0

Step 5: for i = 0 to n-1

1. ans[ coins[ i ] ] <-- 1

Step 6: for i = 2 to val

1. x <-- 1 + ans[ i - 1 ]
2. for c in coins
   1. if c < i
      1. x <-- MIN(x, 1 + ans[ i - c ])
3. if ans[i] != 0
   1. ans[i] <-- MIN(ans[i], x)
4. else
   1. ans[i] <-- x

Step 7: Print ans[val]

Stop

**Code:**

#include <iostream>

#include <vector>

int main()

{

std::cout<<"Enter the number of available coins\n";

int n; std::cin>>n;

std::vector<int> coins(n);

std::cout<<"Enter the coins\n";

for(auto& i : coins)

std::cin>>i;

std::cout<<"Enter the value\n";

int val; std::cin>>val;

std::vector<int> ans(val+1, 0);

for(const auto& i : coins)

ans[i] = 1;

for(int i = 2; i <= val; i++)

{

int x = 1 + ans[i - 1];

for(const auto& c : coins)

if(c < i)

x = std::min(x, 1 + ans[i - c]);

if(ans[i] != 0)

ans[i] = std::min(ans[i], x);

else

ans[i] = x;

}

std::cout<<ans[val]<<'\n';

}

**INPUT:** Enter the number of available coins

4

Enter the coins

1 2 7 10

Enter the value

14

**OUTPUT:** 2

**Question 3:** Given a set of non-negative integers, and a value sum, determine if there is a subset of the given set with sum equal to given sum.

**Algorithm**

Start

Algorithm for method main():

Start

Step 1: Input --> n

Step 2: Declare an integer array s of length n and input elements in it

Step 3: Input --> sum

Step 4: if check(s, n, sum) = true

Print(“found”)

else

Print(“Not found”)

Stop

Algorithm for method check(s[], n, sum):

Start

Step 1: if sum = 0

return true

Step 2: if n = 0 AND sum != 0

return false

Step 3: if s[ n-1 ] > sum

return check( s, n-1, sum )

Step 4: return check( s, n-1, sum ) OR check( s, n-1, sum – s[ n-1 ] )

Stop

Stop

**Code:**

#include <iostream>

#include <vector>

bool check(const std::vector<int> &s, int n, int sum)

{

if( sum == 0 )

return true;

if( n==0 && sum != 0 )

return false;

if(s[n-1] > sum)

return check(s, n-1, sum);

return check(s, n-1, sum) || check(s, n-1, sum - s[n-1]);

}

int main()

{

std::cout<<"Enter the number of elements in the set\n";

int n; std::cin>>n;

std::vector<int> s(n);

std::cout<<"Enter the elements of the set\n";

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

std::cin>>s[i-1];

std::cout<<"Enter the sum\n";

int sum; std::cin>>sum;

if(check(s, n, sum))

std::cout<<"Found\n";

else

std::cout<<"Not found\n";

}

**INPUT:**  Enter the number of elements in the set

4

Enter the elements of the set

1 4 3 2

Enter the sum

6

**OUTPUT:** Found

**Question 4:** Given a cost 2D-matrix and a position (m, n), write a function that returns cost of minimum cost-path to reach (m, n) from (0, 0).

**Algorithm**

Start

Algorithm for method main()

Start

Step 1: Declare two integer variables m, n and accept values in them

Step 2: Declare an integer array arr of size m \* n and accept values in it

Step 3: Input the coordinates of the destination position and store them in integer variables x and y

Step 4: print( find( arr, x-1, y-1 ) )

Stop

Algorithm for method find(arr, m, n)

Start

Step 1: if m < 0 OR n < 0

return ∞

Step 2: if m = 0 AND n = 0

return arr[ M ][ N ]

Step 3: return arr[M][N] + MIN( find(arr, m-1, n-1), find(arr, m, n-1), find(arr, m-1, n) )

Stop

Stop

**Code:**

#include <iostream>

#include <vector>

int inf = 1;

int min(std::vector<int> &&arr)

{

int ans = arr[0];

for(int i = 1; i < arr.size(); i++)

if(ans > arr[i])

ans = arr[i];

return ans;

}

int find(std::vector<std::vector<int>> &arr, int m, int n)

{

if(m < 0 || n < 0)

return inf;

if(m == 0 && n == 0)

return arr[m][n];

return arr[m][n] + min({find(arr, m-1, n-1), find(arr, m-1, n), find(arr, m, n-1)});

}

int main()

{

int m, n;

std::cin>>m>>n;

std::vector<std::vector<int>> arr(m, std::vector<int>(n));

for(auto& i : arr)

for(auto& j : i)

{

std::cin>>j;

inf += j;

}

std::cout<<"Enter the position\n";

int x, y; std::cin>>x>>y;

std::cout<<find(arr, x-1, y-1)<<'\n';

}

**INPUT:** 3 3

1 2 3

4 8 2

1 5 3

Enter the position

3 3

**OUTPUT:** 8