

I. Write C/C++ programs to implement the following algorithm using Backtracking technique.

i) Subset Sum

1.

Code:

```
#include <iostream>
```

```
#include <vector>
```

```
using namespace std;
```

```
void subsetSum(vector<int>& nums, int target, vector<int>& subset, int index) {
```

```
    if (target == 0) {
```

```
        for (int num : subset)
```

```
            cout << num << " ";
```

```
        cout << endl;
```

```
        return;
```

```
    }
```

```
    if (index == nums.size() || target < 0)
```

```
        return;
```

```
    subset.push_back(nums[index]);
```

```
    subsetSum(nums, target - nums[index], subset, index + 1);
```

```
    subset.pop_back();
```

```
    subsetSum(nums, target, subset, index + 1);
```

```
}
```

```
int main() {
```

```
    vector<int> nums = {2, 3, 7, 8, 10};
```

```

int target = 10;

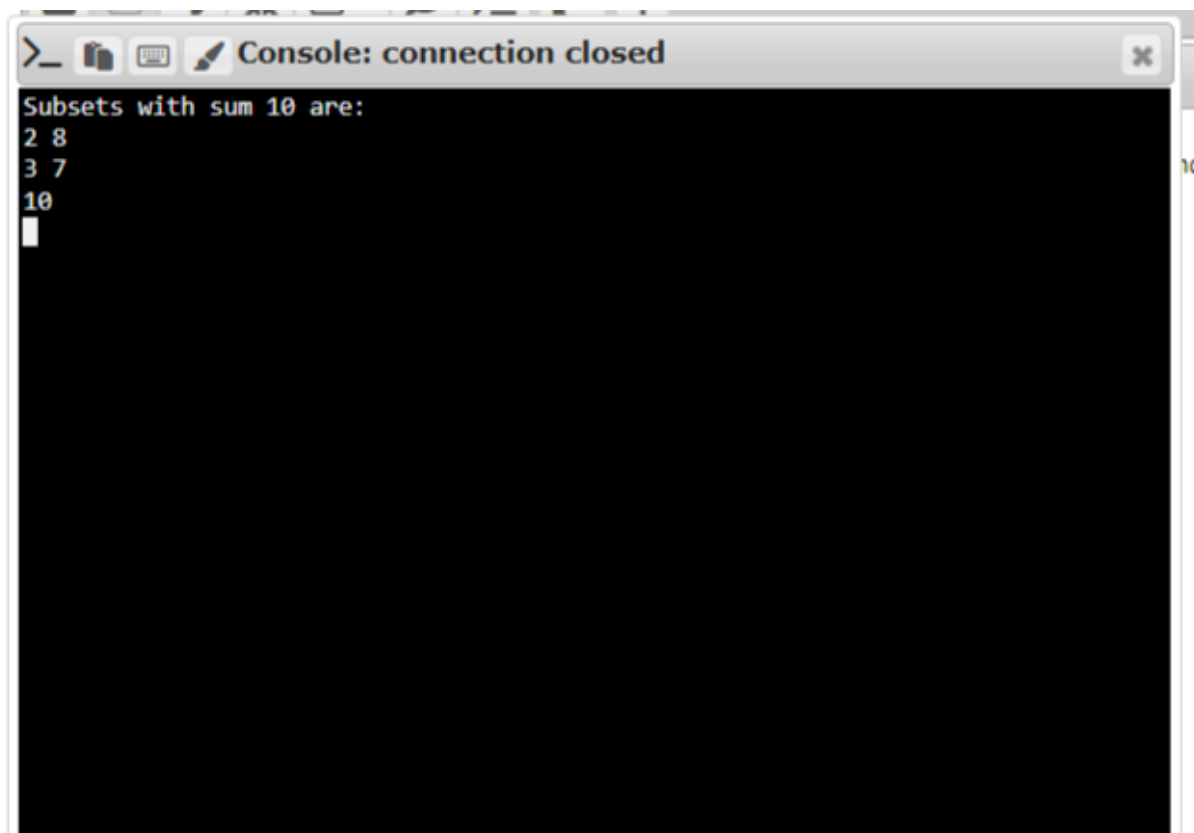
vector<int> subset;

cout << "Subsets with sum " << target << " are:\n";
subsetSum(nums, target, subset, 0);

return 0;
}

```

Output:



```

>_ Console: connection closed
Subsets with sum 10 are:
2 8
3 7
10

```

2.
 - II. Write C/C++ programs to implement the following algorithm using Branch and Bound technique.
 - i) 0/1 Knapsack problem
 - ii) Job Selection problem

0/1 :

```
#include <iostream>
```

```
#include <vector>

#include <algorithm>

using namespace std;

struct Item {
    int weight;
    int value;
    double ratio;
};

bool compare(Item a, Item b) {
    return a.ratio > b.ratio;
}

int knapsackBranchBound(int capacity, vector<Item>& items, int n) {

    sort(items.begin(), items.end(), compare);

    int currentWeight = 0;
    int currentValue = 0;
    int maxPossibleValue = 0;

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

        if (currentWeight + items[i].weight <= capacity) {
            currentWeight += items[i].weight;
            currentValue += items[i].value;
        } else {

            int remainingCapacity = capacity - currentWeight;
```

```

        double fraction = (double)remainingCapacity / items[i].weight;
        currentValue += items[i].value * fraction;
        break;
    }
}

return currentValue;
}

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

    int maxValue = knapsackBranchBound(capacity, items, n);

    cout << "Maximum value that can be obtained: " << maxValue << endl;

    return 0;
}

```

Output:

Output

/tmp/aGot8htokv.o

Maximum value that can be obtained: 240

Job selection:

Code:

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

using namespace std;

struct Job {
    int start;
    int finish;
};

bool compareJobs(const Job& a, const Job& b) {
    return a.finish < b.finish;
}

int maxNonOverlappingJobs(const vector<Job>& jobs) {
    int n = jobs.size();
    if (n == 0) return 0;

    vector<Job> sortedJobs = jobs;
    sort(sortedJobs.begin(), sortedJobs.end(), compareJobs);

    int count = 1;
    int prevFinish = sortedJobs[0].finish;

    for (int i = 1; i < n; ++i) {
        if (sortedJobs[i].start >= prevFinish) {
```

```

        count++;

        prevFinish = sortedJobs[i].finish;
    }
}

return count;
}

int main() {
    vector<Job> jobs = {{1, 3}, {2, 5}, {3, 8}, {5, 9}, {6, 10}, {8, 11}};
    cout << "Maximum number of non-overlapping jobs: " << maxNonOverlappingJobs(jobs) << endl;
    return 0;
}

```

Output

```

/tmp/aGot8htokv.o
Maximum number of non-overlapping jobs: 3

```

IV. Write C/C++ programs to implement the following String matching algorithms

- i) Naïve string Matching alg
- ii) Rain Karp alg.

1. Naive string matching:

Code:

```

#include <iostream>

#include <string>

```

```

using namespace std;

void naiveStringMatch(const string& text, const 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 << endl;
        }
    }
}

int main() {
    string text = "AABAACAADAABAABA";
    string pattern = "AABA";

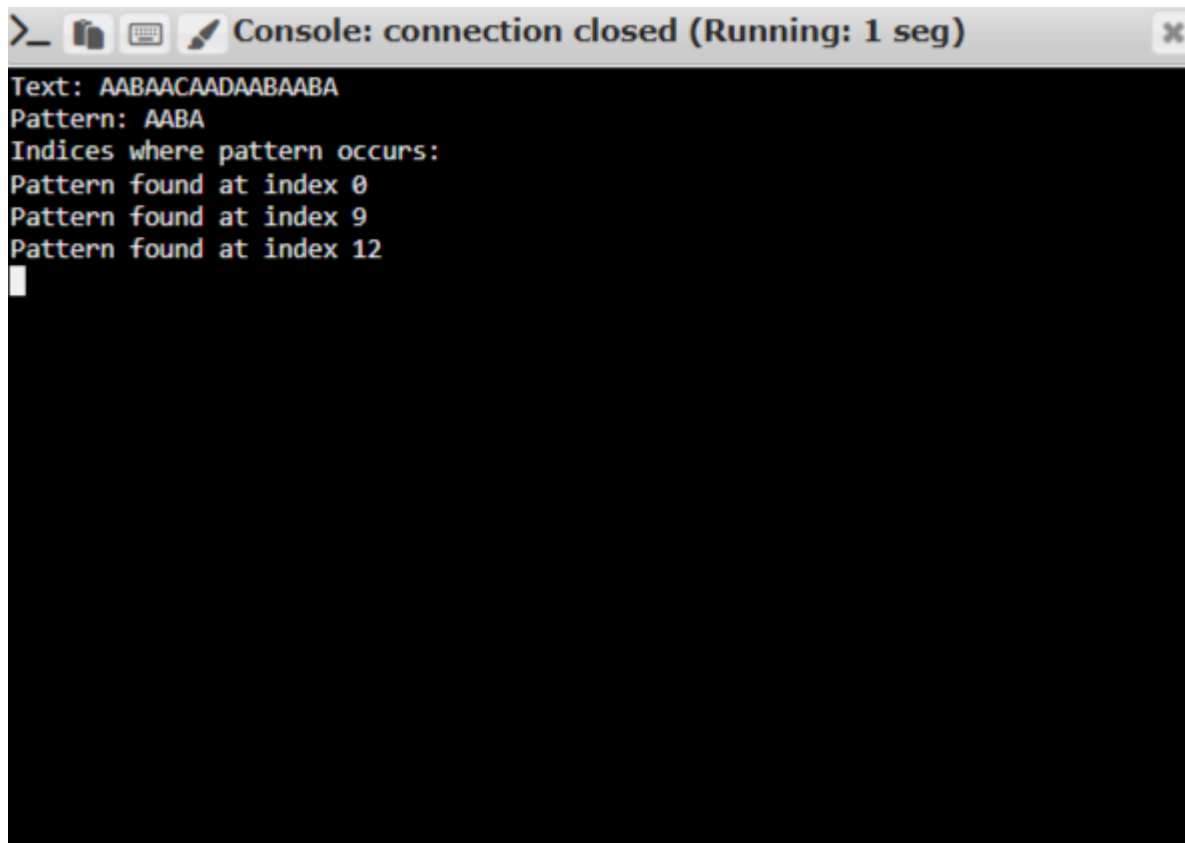
    cout << "Text: " << text << endl;
    cout << "Pattern: " << pattern << endl;

    cout << "Indices where pattern occurs:" << endl;
    naiveStringMatch(text, pattern);

    return 0;
}

```

Output:

A screenshot of a console window with a grey title bar. The title bar contains a close button (X) on the right and three icons (a magnifying glass, a keyboard, and a pencil) on the left. The text in the title bar is "Console: connection closed (Running: 1 seg)". The console area is black with white text. The output is as follows:

```
Text: AABAACAADAABAABA
Pattern: AABA
Indices where pattern occurs:
Pattern found at index 0
Pattern found at index 9
Pattern found at index 12
█
```

Rabin karp:

Code:

```
#include <iostream>
```

```
#include <string>
```

```
#include <cmath>
```

```
using namespace std;
```

```
void rabinKarp(const string& text, const string& pattern) {
```

```
    int n = text.length();
```

```
    int m = pattern.length();
```

```
    int prime = 101;
```

```
    int d = 256;
```

```
    int patternHash = 0;
```

```
    int textHash = 0;
```



```

for (int i = 0; i < m; ++i) {
    patternHash = (d * patternHash + pattern[i]) % prime;
    textHash = (d * textHash + text[i]) % prime;
}

```

```

int h = 1;
for (int i = 0; i < m - 1; ++i)
    h = (h * d) % prime;

```

```

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

```

```

    if (patternHash == textHash) {

```

```

        int j;
        for (j = 0; j < m; ++j) {
            if (text[i + j] != pattern[j])
                break;
        }

```

```

        if (j == m)
            cout << "Pattern found at index " << i << endl;
    }

```

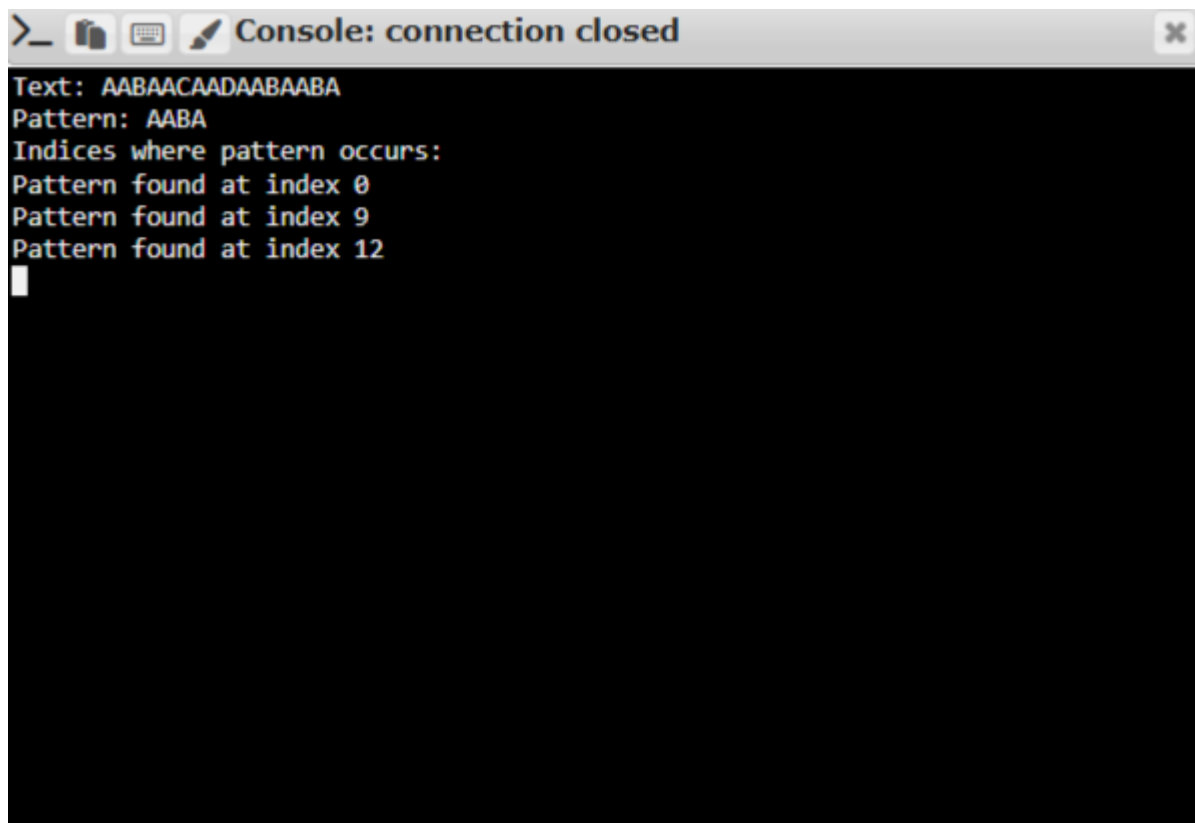
```

if (i < n - m) {
    textHash = (d * (textHash - text[i] * h) + text[i + m]) % prime;
    if (textHash < 0)
        textHash = (textHash + prime);
}
}

```

```
}
```

```
int main() {  
    string text = "AABAACAADAABAABA";  
    string pattern = "AABA";  
  
    cout << "Text: " << text << endl;  
    cout << "Pattern: " << pattern << endl;  
  
    cout << "Indices where pattern occurs:" << endl;  
    rabinKarp(text, pattern);  
  
    return 0;  
}
```



The screenshot shows a console window titled "Console: connection closed". The output of the program is as follows:

```
Text: AABAACAADAABAABA  
Pattern: AABA  
Indices where pattern occurs:  
Pattern found at index 0  
Pattern found at index 9  
Pattern found at index 12  
█
```

The console window has a standard Windows-style title bar with a close button (X) in the top right corner. The output text is displayed in a monospaced font on a black background.