Code Similarity Analysis Report

Analysis Summary

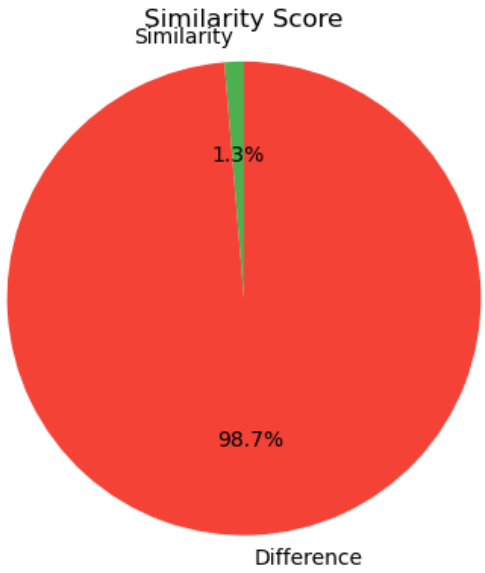
Comparison between: temp1.c and temp2.c

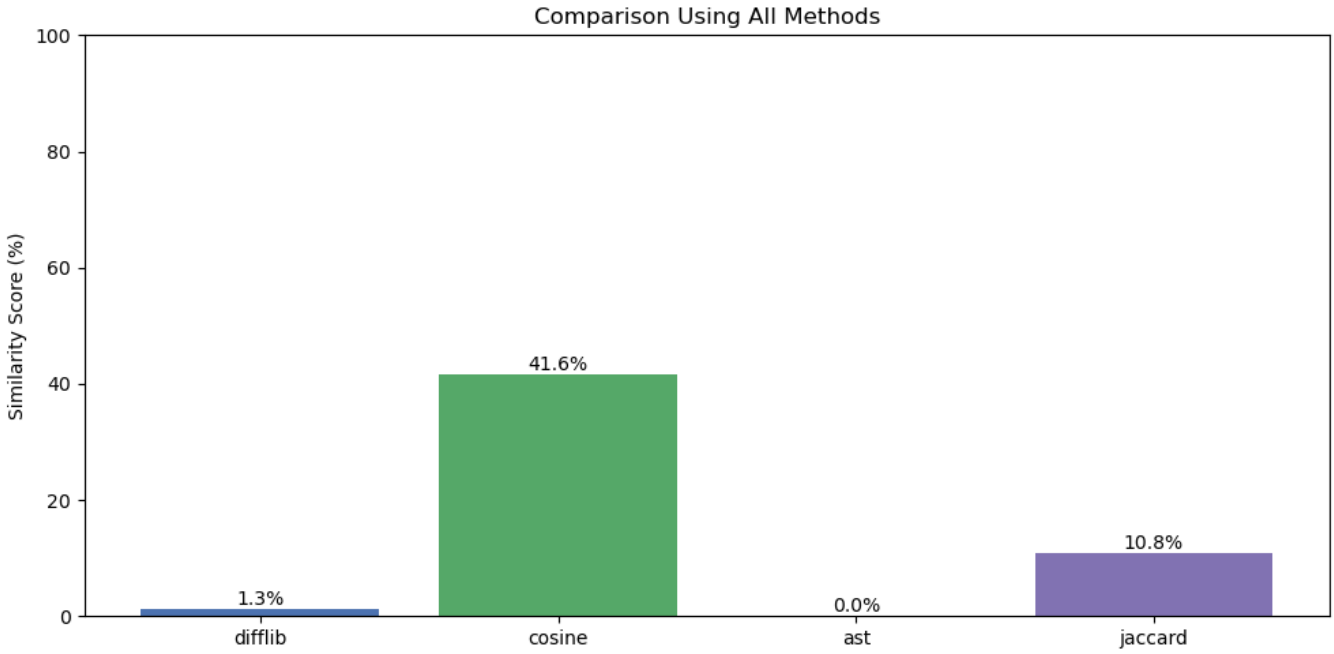
Selected Method: DIFFLIB

Similarity Score: 1.32%

Plagiarism Threshold (70%) Exceeded: No

Similarity Visualizations





Code Similarity Report

Preprocessing Details

Before comparison, the following preprocessing steps were applied:

1. All comments were removed

2. All identifiers were normalized (variables ? vN, functions ? fN, etc.)

Original vs Preprocessed Code

Original temp1.c:

```
// Banker's Algorithm
    #include<stdio.h>
    int main()
    {
        int n , m , i , j , k;
        n = 5;
        m = 4;
        int alloc[ 5 ] [ 4 ] = { { 0 , 1 ,1, 0 },
                                 { 1,2,3,1} ,
                                 {1,3,6,5} ,
                                 {0,6,3,2} ,
                                 {0,0,1,4} } ;
        int max[ 5 ] [ 4 ] = { {0,2,1,0} ,
                               {1,6,5,2} ,
                               {2,3,6,6} ,
                               {0,6,5,2} ,
                               {0,6,5,6} } ;
        int avail[4] = {1,5,2,0} ;
        int f[n] , ans[n] , ind = 0 ;
        for (k = 0; k < n; k++) {
            f[k] = 0;
        }
        int need[n][m];
        for (i = 0; i < n; i++) {
            for (j = 0; j < m; j++)
                need[i][j] = max[i][j] - alloc[i][j] ;
        }
        int y = 0;
        for (k = 0; k < 5; k++){
            for (i = 0; i < n; i++){
                if (f[i] == 0){
                    int flag = 0;
                   ...
```

Preprocessed temp1.c:

// Banker's Algorithm

```
    int main()
    {
```

Code Similarity Report

```
        int n , m , i , j , k;
        n = 5;
        m = 4;
        int alloc[ 5 ] [ 4 ] = { { 0 , 1 ,1, 0 },
                                 { 1,2,3,1} ,
                                 {1,3,6,5} ,
                                 {0,6,3,2} ,
                                 {0,0,1,4} } ;
        int max[ 5 ] [ 4 ] = { {0,2,1,0} ,
                               {1,6,5,2} ,
                               {2,3,6,6} ,
                               {0,6,5,2} ,
                               {0,6,5,6} } ;
        int avail[4] = {1,5,2,0} ;
        int f[n] , ans[n] , ind = 0 ;
        for (k = 0; k < n; k++) {
           f[k] = 0;
        }
        int need[n][m];
        for (i = 0; i < n; i++) {
           for (j = 0; j < m; j++)
              need[i][j] = max[i][j] - alloc[i][j] ;
        }
        int y = 0;
        for (k = 0; k < 5; k++){
           for (i = 0; i < n; i++){
              if (f[i] == 0){
                 int flag = 0;
                 for (j = 0; j < ...
```

Original temp2.c:

```
#include <stdio.h>
#include <stdlib.h>
void calculatedifference(int request[], int head, int** diff, int n) {
   for (int i = 0; i < n; i++) {
      diff[i][0] = abs(head - request[i]);
   }
}



int findMIN(int** diff, int n) {
   int index = -1;
   int minimum = 1e9;
   for (int i = 0; i < n; i++) {
      if (!diff[i][1] && minimum > diff[i][0]) {
         minimum = diff[i][0];
         index = i;
      }
   }
   return index;
```

Code Similarity Report

```
}
void shortestSeekTimeFirst(int request[], int head, int n) {
    if (n == 0) {
        return;
    }
    int** diff = malloc(n * sizeof(int*));
    for (int i = 0; i < n; i++) {
        diff[i] = malloc(2 * sizeof(int));
    }
    int* seeksequence = malloc((n + 1) * sizeof(int));
    int seekcount = 0;
    for (int i = 0; i < n; i++) {
        seeksequence[i] = head;
        calculatedifference(request, head, diff, n);
        int index = findMIN(diff, n);
        diff[index][1] = 1;
        seekcount += diff[index][0];
        head = reque...
```

Preprocessed temp2.c:

```
void calculatedifference(int request[], int head, int** diff, int n) {
    for (int i = 0; i < n; i++) {
        diff[i][0] = abs(head - request[i]);
    }
}


int findMIN(int** diff, int n) {
    int index = -1;
    int minimum = 1e9;
    for (int i = 0; i < n; i++) {
        if (!diff[i][1] && minimum > diff[i][0]) {
            minimum = diff[i][0];
            index = i;
        }
    }
    return index;
}
void shortestSeekTimeFirst(int request[], int head, int n) {
    if (n == 0) {
        return;
    }
    int** diff = malloc(n * sizeof(int*));
    for (int i = 0; i < n; i++) {
        diff[i] = malloc(2 * sizeof(int));
    }
    int* seeksequence = malloc((n + 1) * sizeof(int));
    int seekcount = 0;
    for (int i = 0; i < n; i++) {
```

Code Similarity Report

```
    seeksequence[i] = head;
    calculatedifference(request, head, diff, n);
    int index = findMIN(diff, n);
    diff[index][1] = 1;
    seekcount += diff[index][0];
    head = request[index];
  }
  seeksequence[n] ...
```

Detailed Differences (Preprocessed Code)

```
--- file1
+++ file2
@@ -1,67 +1,57 @@
-// Banker's Algorithm
-
-    int main()
-    {
-        int n , m , i , j , k;
-        n = 5;
-        m = 4;
-        int alloc[ 5 ] [ 4 ] = { { 0 , 1 ,1, 0 },
-                                 { 1,2,3,1} ,
-                                 {1,3,6,5} ,
-                                 {0,6,3,2} ,
-                                 {0,0,1,4} } ;
-        int max[ 5 ] [ 4 ] = { {0,2,1,0} ,
-                         {1,6,5,2} ,
-                         {2,3,6,6} ,
-                         {0,6,5,2} ,
-                         {0,6,5,6} } ;
-        int avail[4] = {1,5,2,0} ;
-        int f[n] , ans[n] , ind = 0 ;
-        for (k = 0; k < n; k++) {
-            f[k] = 0;
+
+
+void calculatedifference(int request[], int head, int** diff, int n) {
+    for (int i = 0; i < n; i++) {
+        diff[i][0] = abs(head - request[i]);
+    }
+}
+
+
+int findMIN(int** diff, int n) {
+    int index = -1;
+    int minimum = 1e9;
+    for (int i = 0; i < n; i++) {
+        if (!diff[i][1] && minimum > diff[i][0]) {
+            minimum = diff[i][0];
```

Code Similarity Report

```
+            index = i;
         }
-        int need[n][m];
-        for (i = 0; i < n; i++) {
-            for (j = 0; j < m; j++)
-                need[i][j] = max[i][j] - alloc[i][j] ;
-        }
-        int y = 0;
-        for (k = 0; k < 5; k++){
-            for (i = 0; i < n; i++){
-                if (f[i] == 0){
-                    int flag = 0;
-                    for (j = 0; j < m; j++) {
-                        if(need[i][j] > avail[j]){
-                            flag = 1;
-                            break;
-                                        }
-                    }
-                    if ( flag == 0 ) {
-                        ans[ind++] = i;
-                        for (y = 0; y < m; y++)
-                            avail[y] += alloc[i][y] ;
-                        f[i] = 1;
-                    }
-                }
-            }
-        }
-        int flag = 1;
-        for(int i=0;i<n;i++)
-        {
-        if(f[i] == 0)
-        {
-            flag = 0;
-            printf(" The following system is not safe ");
-            break;
-        }
-        }
-
-        if (flag == 1)
-        {
-        printf(" Following is the SAFE Sequence \n ");
-        for (i = 0; i < n - 1; i++)
-            printf(" P%d -> " , ans[i]);
-        printf(" P%d ", ans[n - 1]);
-        }
-        return(0);
     }
+    return index;
+}
+void shortestSeekTimeFirst(int request[], int head, int n) {
+    if (n == 0) {
```

Code Similarity Report

```
+        return;
+    }
+    int** diff = malloc(n * sizeof(int*));
+    for (int i = 0; i < n; i++) {
+        diff[i] = malloc(2 * sizeof(int));
+    }
+    int* seeksequence = malloc((n + 1) * sizeof(int));
+    int seekcount = 0;
+    for (int i = 0; i < n; i++) {
+        seeksequence[i] = head;
+        calculatedifference(request, head, diff, n);
+        int index = findMIN(diff, n);
+        diff[index][1] = 1;
+        seekcount += diff[index][0];
+        head = request[index];
+    }
+    seeksequence[n] = head;
+    printf("Total number of seek operations = %d\n", seekcount);
+    printf("Seek sequence is:\n");
+    for (int i = 0; i <= n; i++) {
+        printf("%d\n", seeksequence[i]);
+    }
+    for (int i = 0; i < n; i++) {
+        free(diff[i]);
+    }
+    free(diff);
+    free(seeksequence);
+}
+int main() {
+    int n = 8;
+    int* proc = malloc(n * sizeof(int));
+    proc[0] = 176; proc[1] = 79; proc[2] = 34; proc[3] = 60; proc[4] = 92; proc[5] = 11; proc[6] = 41; proc[7]
= 114;
+    shortestSeekTimeFirst(proc, 50, n);
+    free(proc);
+    return 0;}
```