CODE

interval_schedule.c

```c
#include <stdio.h>
#include <stdlib.h>

void swap(int *a, int *b) {
   int tmp = *a;
   *a = *b;
   *b = tmp;
}

int partition(int arr[][2], int begin, int end) {
   int pivot = arr[end][1];
   int i = begin - 1;

   for (int j=begin; j<end; j++) {
       if (arr[j][1] < pivot) {
           i++;
           swap(&arr[j][1], &arr[i][1]);
       }
   }
   swap(&arr[i+1][1], &arr[end][1]);

   return i+1;
}

void quick_sort(int arr[][2], int begin, int end) {
   int pivot;

   if (begin < end) {
       pivot = partition(arr, begin, end);
       quick_sort(arr, begin, pivot-1);
       quick_sort(arr, pivot+1, end);
   }
}
```

```c
int *interval_scheduler(int tasks[][2], int len) {
   int j, *res_set = NULL;

   quick_sort(tasks, 0, len-1);
   res_set = malloc(len*sizeof(int)); // store indices of tasks
   if(res_set == NULL) {
       printf("MEMORY ALLOCATION ERROR.\n");
       exit(-1);
   }

   // add tasks[0]
   res_set[0] = 0;
   j = 1;
   for(int i=1; i<len; i++) {
       // if start time of tasks[i] greater than end time of last task
taken
       if(tasks[i][0] >= tasks[res_set[j-1]][1]) {
           res_set[j++] = i;
       }
   }

   // fill remaining part of res_set with some invalid int to indicate
end
   while(j<len) {
       res_set[j++] = -1;
   }
   return res_set;
}

int main() {
   int tasks[10][2] =
{{0,4},{1,2},{2,4},{3,5},{3,6},{5,6},{5,7},{6,7},{7,9},{8,10}};
   int *res = interval_scheduler(tasks, 10);

   printf("Given Tasks: \n");
   for(int i=0; i<10; i++) {
       printf("(%d, %d), ", tasks[i][0], tasks[i][1]);
```

```c
    }

    printf("\n\nSchedule of maximum non-overlapping tasks: \n");
    for(int i=0; i<10; i++) {
        if(res[i] == -1) {
            break;
        }
        printf("(%d, %d), ", tasks[res[i]][0], tasks[res[i]][1]);
    }
    printf("\n");

    free(res);
    return 0;
}
```

edit_dist.c

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int min(int a, int b, int c) {
    int curr_min = a;

    if(b < curr_min) {
        curr_min = b;
    }
    if(c < curr_min) {
        curr_min = c;
    }

    return curr_min;
}

int edit_dist(char *str1, char *str2, int str1_indx, int str2_indx, int
**memo) {
    /* Convert str1 to str2 */
    if(str1_indx < 0) {
        /* add remaining chars of str2 to str1 to make them equal
        no of insertions = no of chars left in str2 = str2_indx + 1 */
        return str2_indx + 1;
    }
    if(str2_indx < 0) {
        /* delete remaining chars of str1 to make str1 == str2
         no of deletions = no of chars left in str1 = str1_indx + 1 */
        return str1_indx + 1;
    }

    if(memo[str2_indx][str1_indx] != -1) {
        return memo[str2_indx][str1_indx];
    } else {
        if(str1[str1_indx] == str2[str2_indx]) {
            // no new operations. just move one indx back
            memo[str2_indx][str1_indx] = edit_dist(str1, str2,
str1_indx-1, str2_indx-1, memo);
            return memo[str2_indx][str1_indx];
        } else { //+1 for 1 operation
            //replace
            /* replace char at str1_indx (then str1[str1_indx] ==
str2[str2_indx])
             move back one indx now */
            int replace = 1 + edit_dist(str1, str2, str1_indx-1,
str2_indx-1, memo);

            //insert
            /* insert after str_indx1. (then str1[str1_indx+1] ==
str2[str2_indx])
             move back one indx now */
            int insert = 1 + edit_dist(str1, str2, str1_indx,
str2_indx-1, memo);

            //delete
            /* delete at str_indx1. pointer moves to str1_indx-1.
             stay at str2_indx and compare with str1_indx-1 */
            int delete = 1 + edit_dist(str1, str2, str1_indx-1,
str2_indx, memo);

            memo[str2_indx][str1_indx] = min(replace, insert, delete);
            return memo[str2_indx][str1_indx];
        }
    }
}

int main() {
    char *str1 = NULL, *str2 = NULL;
    int **memo = NULL, res, len1, len2;
    char *inp[] = {"bmsse", "cat", "sunday"};
    char *outp[] = {"bmscse", "cut", "saturday"};

    for (int i=0; i<3; i++) {
        str1 = inp[i];
        str2 = outp[i];
        len1 = strlen(str1);
        len2 = strlen(str2);

        memo = malloc(len2*sizeof(int *));
        if(memo == NULL) {
            printf("MEMORY ALLOCATION ERROR.\n");
            exit(-1);
        }
        for(int i=0; i<len2; i++) {
            memo[i] = malloc(len1*sizeof(int));
            if(memo[i] == NULL) {
                printf("MEMORY ALLOCATION ERROR.\n");
                exit(-1);
            }
            for(int j=0; j<len1; j++) {
                memo[i][j] = -1;
            }
        }
        res = edit_dist(str1, str2, len1-1, len2-1, memo);
        printf("%s -> %s: Edit Dist = %d\n", str1, str2, res);

        for(int i=0; i<len2; i++) {
            free(memo[i]);
        }
        free(memo);
    }

    return 0;
}
```

OUTPUTS

interval_schedule.c

```
Given Tasks:
(0, 2), (1, 4), (2, 4), (3, 5), (3, 6), (5, 6), (5, 7), (6, 7), (7, 9),
(8, 10),

Schedule of maximum non-overlapping tasks:
(0, 2), (2, 4), (5, 6), (6, 7), (7, 9),
```

edit_dist.c

```
bmsse -> bmscse: Edit Dist = 1
cat -> cut: Edit Dist = 1
sunday -> saturday: Edit Dist = 3
```