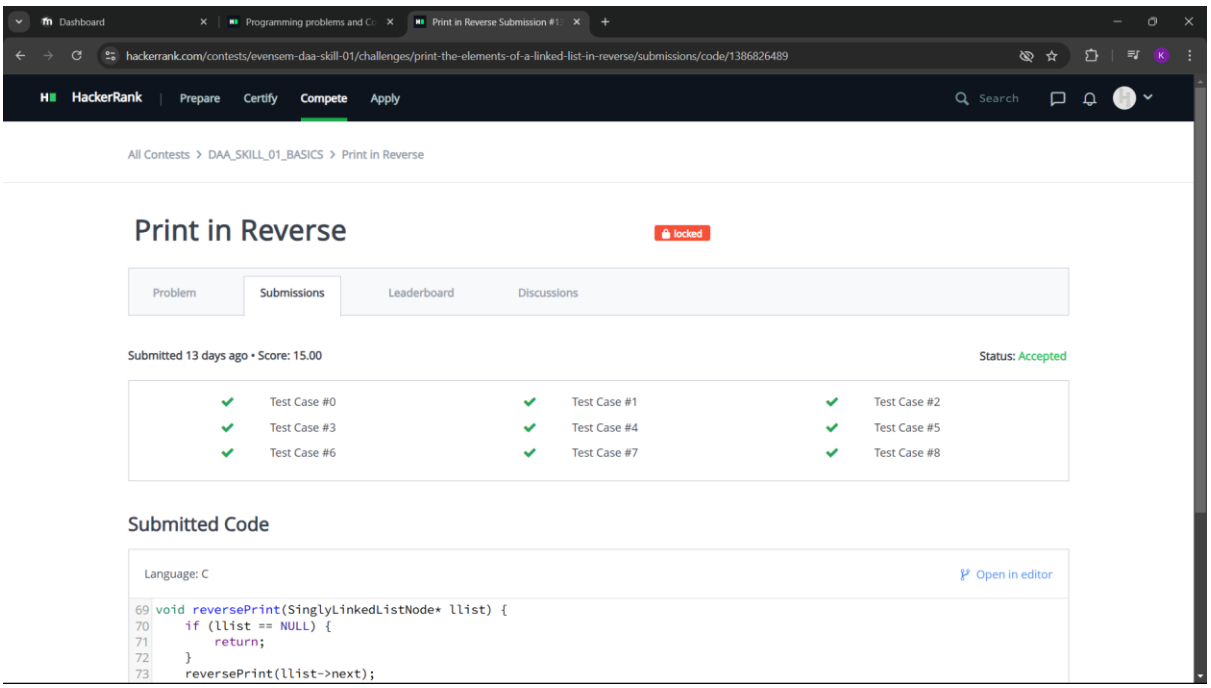
1. Print in Reverse

Code:

```
void reversePrint(SinglyLinkedListNode* llist) {
    if (llist == NULL) {
        return;
    }
    reversePrint(llist->next);
    printf("%d\n", llist->data);
}
```

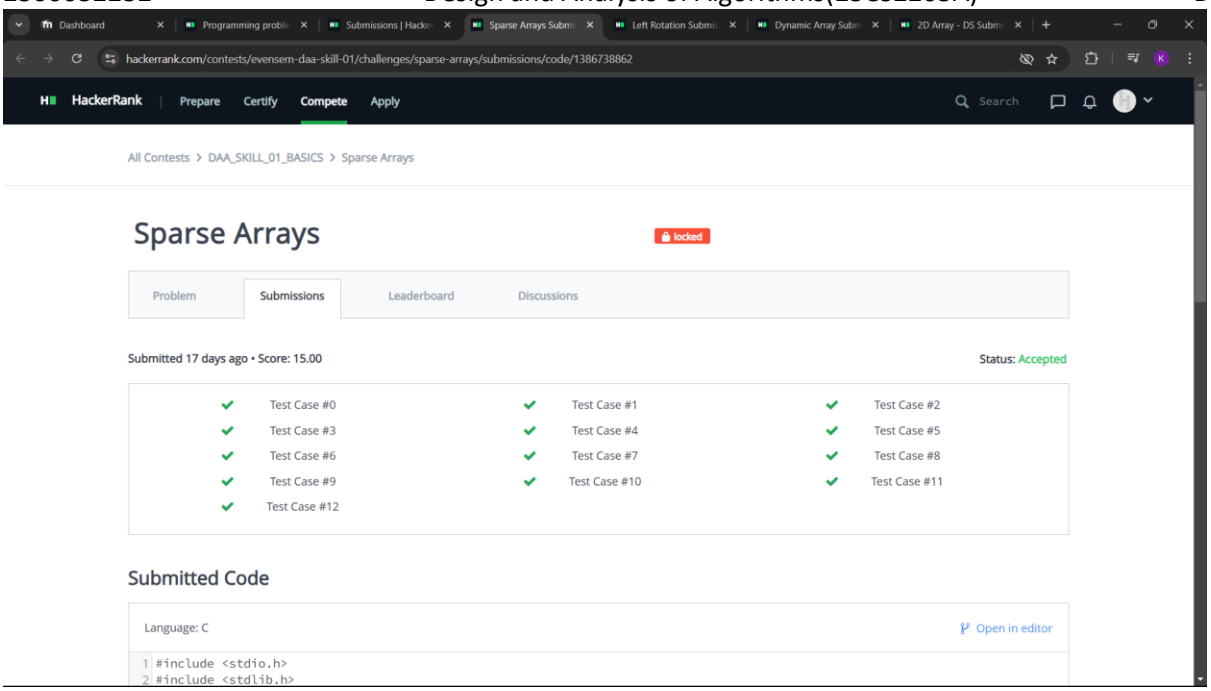


2. Sparse Arrays

Code:

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

int* matchingStrings(int stringList_count, char** stringList, int queries_count, char** queries, int*
result_count) {
    *result_count = queries_count;
    int* results = malloc(queries_count * sizeof(int));

    for (int i = 0; i < queries_count; i++) {
        int count = 0;
        for (int j = 0; j < stringList_count; j++) {
            if (strcmp(queries[i], stringList[j]) == 0) {
                count++;
            }
        }
        results[i] = count;
    }

    return results;
}
```

```
int main() {
    int stringList_count;
    scanf("%d", &stringList_count);
    getchar();

    char** stringList = malloc(stringList_count * sizeof(char*));
    for (int i = 0; i < stringList_count; i++) {
        stringList[i] = malloc(1024 * sizeof(char));
        scanf("%s", stringList[i]);
    }

    int queries_count;
    scanf("%d", &queries_count);
    getchar();

    char** queries = malloc(queries_count * sizeof(char*));
    for (int i = 0; i < queries_count; i++) {
        queries[i] = malloc(1024 * sizeof(char));
        scanf("%s", queries[i]);
    }

    int result_count;
    int* results = matchingStrings(stringList_count, stringList, queries_count, queries, &result_count);

    for (int i = 0; i < result_count; i++) {
        printf("%d\n", results[i]);
    }

    for (int i = 0; i < stringList_count; i++) {
        free(stringList[i]);
    }
    free(stringList);

    for (int i = 0; i < queries_count; i++) {
        free(queries[i]);
    }
    free(queries);

    free(results);

    return 0;
}
```

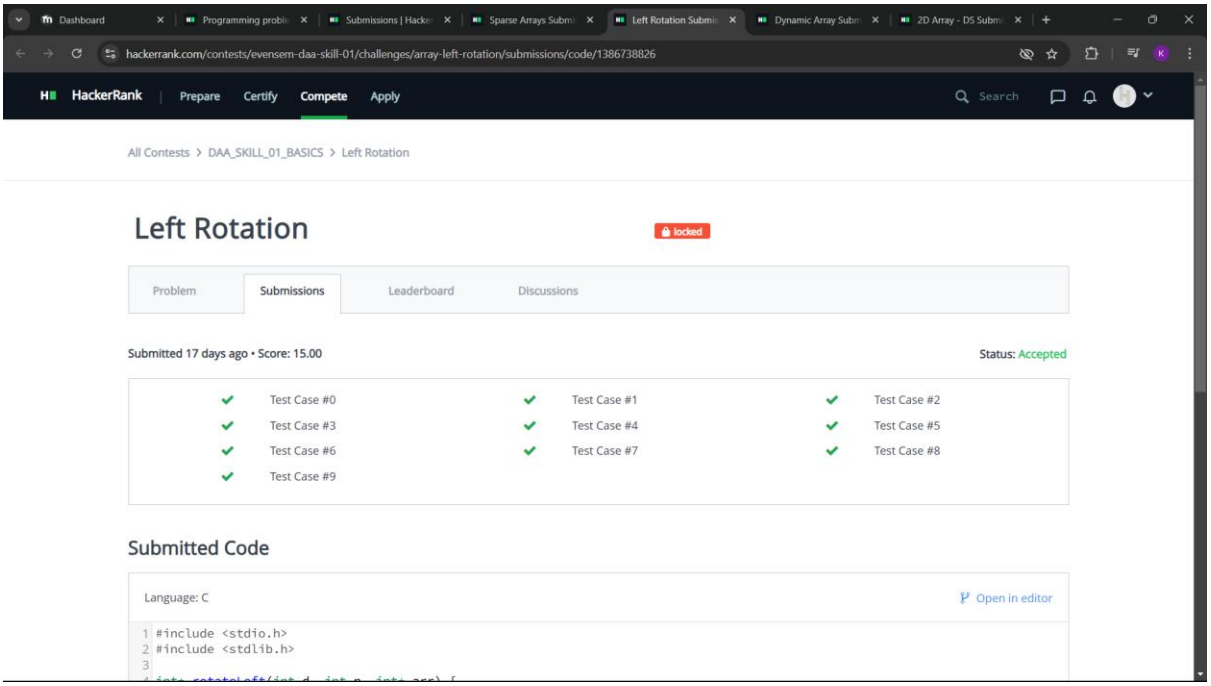


3. Left rotation

Code:

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

int* rotateLeft(int d, int n, int* arr) {
    int* rotated = malloc(n * sizeof(int));
    for (int i = 0; i < n; i++) {
        rotated[i] = arr[(i + d) % n];
    }
    return rotated;
}

int main() {
    int n, d;
    scanf("%d %d", &n, &d);
    int* arr = malloc(n * sizeof(int));
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    int* result = rotateLeft(d % n, n, arr);

    for (int i = 0; i < n; i++) {
        printf("%d", result[i]);
        if (i != n - 1) {
            printf(" ");
        }
    }
    printf("\n");

    free(arr);
    free(result);
    return 0;
```

}



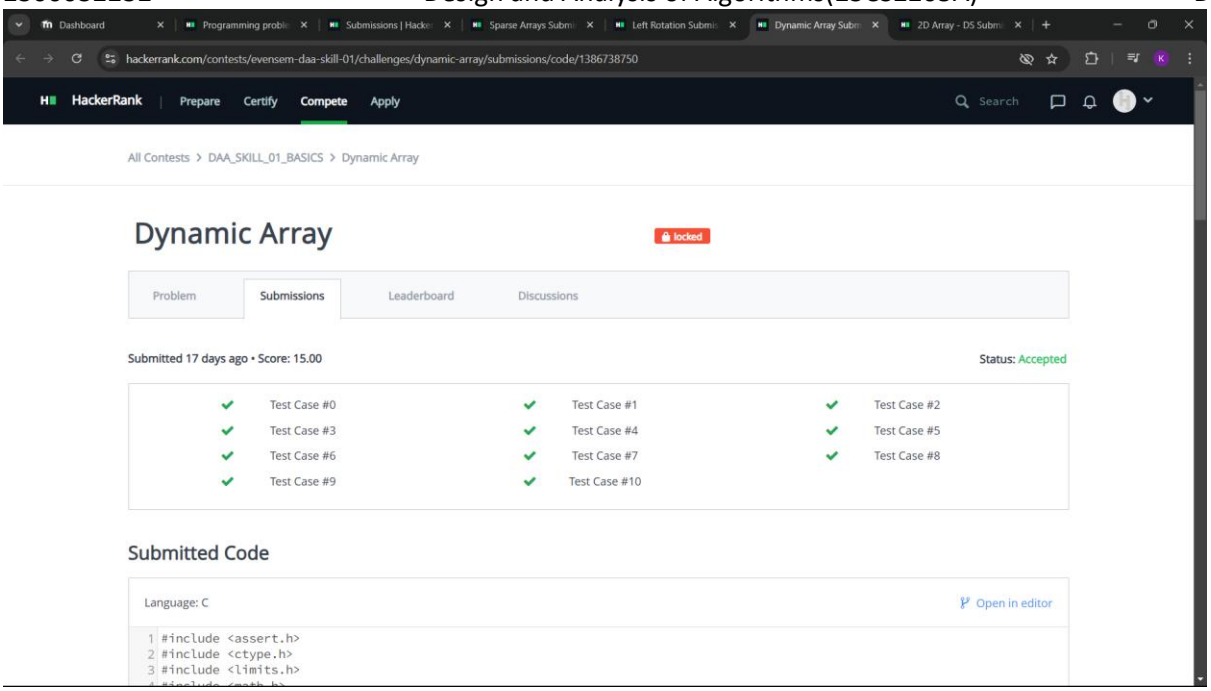
4. Dynamic Array

Code:

```
#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();
char* ltrim(char*);
char* rtrim(char*);
char** split_string(char*);
int parse_int(char*);

int* dynamicArray(int n, int queries_rows, int queries_columns, int** queries, int* result_count) {
   int** arr = malloc(n * sizeof(int*));
   int* sizes = calloc(n, sizeof(int));
   for (int i = 0; i < n; i++) arr[i] = malloc(0);

   int lastAnswer = 0;
   int* results = malloc(queries_rows * sizeof(int));
   *result_count = 0;

   for (int i = 0; i < queries_rows; i++) {
      int type = queries[i][0];
      int x = queries[i][1];
      int y = queries[i][2];
```

```
        int idx = (x ^ lastAnswer) % n;

        if (type == 1) {
            sizes[idx]++;
            arr[idx] = realloc(arr[idx], sizes[idx] * sizeof(int));
            arr[idx][sizes[idx] - 1] = y;
        } else if (type == 2) {
            lastAnswer = arr[idx][y % sizes[idx]];
            results[*result_count] = lastAnswer;
            (*result_count)++;
        }
    }

    for (int i = 0; i < n; i++) free(arr[i]);
    free(arr);
    free(sizes);

    return results;
}

int main() {
    FILE* fptr = fopen(getenv("OUTPUT_PATH"), "w");
    char** first_multiple_input = split_string(rtrim(readline()));
    int n = parse_int(*(first_multiple_input + 0));
    int q = parse_int(*(first_multiple_input + 1));

    int** queries = malloc(q * sizeof(int*));
    for (int i = 0; i < q; i++) {
        *(queries + i) = malloc(3 * (sizeof(int)));
        char** queries_item_temp = split_string(rtrim(readline()));
        for (int j = 0; j < 3; j++) {
            int queries_item = parse_int(*(queries_item_temp + j));
            *(*(queries + i) + j) = queries_item;
        }
    }

    int result_count;
    int* result = dynamicArray(n, q, 3, queries, &result_count);

    for (int i = 0; i < result_count; i++) {
        fprintf(fptr, "%d", *(result + i));
        if (i != result_count - 1) fprintf(fptr, "\n");
    }

    fprintf(fptr, "\n");
    fclose(fptr);
    return 0;
}

char* readline() {
    size_t alloc_length = 1024, data_length = 0;
    char* data = malloc(alloc_length);
```

```
    while (true) {
        char* cursor = data + data_length;
        char* line = fgets(cursor, alloc_length - data_length, stdin);
        if (!line) break;
        data_length += strlen(cursor);
        if (data_length < alloc_length - 1 || data[data_length - 1] == '\n') break;
        alloc_length <<= 1;
        data = realloc(data, alloc_length);
    }
    if (data[data_length - 1] == '\n') {
        data[data_length - 1] = '\0';
        data = realloc(data, data_length);
    } else {
        data = realloc(data, data_length + 1);
        data[data_length] = '\0';
    }
    return data;
}

char* ltrim(char* str) {
    if (!str || !*str) return str;
    while (*str != '\0' && isspace(*str)) str++;
    return str;
}

char* rtrim(char* str) {
    if (!str || !*str) return str;
    char* end = str + strlen(str) - 1;
    while (end >= str && isspace(*end)) end--;
    *(end + 1) = '\0';
    return str;
}

char** split_string(char* str) {
    char** splits = NULL;
    char* token = strtok(str, " ");
    int spaces = 0;
    while (token) {
        splits = realloc(splits, sizeof(char*) * ++spaces);
        splits[spaces - 1] = token;
        token = strtok(NULL, " ");
    }
    return splits;
}

int parse_int(char* str) {
    char* endptr;
    int value = strtol(str, &endptr, 10);
    return value;
}
```



5. 2D Array – DS

Code:

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

int hourglassSum(int arr_rows, int arr_columns, int** arr) {
    int maxSum = INT_MIN;

    for (int i = 0; i < arr_rows - 2; i++) {
        for (int j = 0; j < arr_columns - 2; j++) {
            int sum = arr[i][j] + arr[i][j + 1] + arr[i][j + 2] +
                      arr[i + 1][j + 1] +
                      arr[i + 2][j] + arr[i + 2][j + 1] + arr[i + 2][j + 2];

            if (sum > maxSum) {
                maxSum = sum;
            }
        }
    }

    return maxSum;
}

int main() {
    int** arr = malloc(6 * sizeof(int*));

    for (int i = 0; i < 6; i++) {
        *(arr + i) = malloc(6 * sizeof(int));
        for (int j = 0; j < 6; j++) {
            scanf("%d", &arr[i][j]);
        }
    }
```

```
    int result = hourglassSum(6, 6, arr);
    printf("%d\n", result);

    for (int i = 0; i < 6; i++) {
       free(arr[i]);
    }
    free(arr);

    return 0;
}
```