Университет ИТМО
Кафедра ВТ

Задачи 1 - 5
Алгоритмы и Структуры Данных

Выполнил: Федоров Сергей
Группа: P3212

Санкт-Петербург
2020 г.

- # Задача 1 - Стенка на стенку - 2025

Самоочевидно (можно доказать), что наибольшее количество боев в нашей народной забаве произойдет в тот момент, когда в каждой команде будет примерно одинаковое количество человек, то есть распределены по командам равномерно.

Читаем для каждого боя параметры, считаем среднее количество людей в командах, считаем кол-во команда с “+1” человеком. В конце, считаем кол-во боев, путем перемножения кол-ва человек в каждой команде с кол-вом остальных людей.

Код:

```
//
// Created by Sergey Fedorov on 07/02/2020.
//

#include <stdio.h>
#include "../../utils/common_utils_string.c"

typedef struct {int men, teams;} test;

int main(){
    // 1 ⩽ T ⩽ 10 ⇒ max 2 sym
    int t = str_to_int(get_line(2));

    int max_fights[t];

    for(int i = 0; i < t; i++){
        // 2 ⩽ teams ⩽ men ⩽ 10^4 (5 sym) ⇒ max 11 sym
        char *cur_line = get_line(11);
        char** data = str_split(cur_line, ' ');
        test cur_test = (test) {str_to_int(data[0]), str_to_int(data[1])};

        int average_man_per_team = cur_test.men / cur_test.teams;
        int teams_with_extra = (cur_test.men % cur_test.teams);
        int normal_teams = cur_test.teams - teams_with_extra;

        max_fights[i] =
                (
                    normal_teams * average_man_per_team * (cur_test.men -
average_man_per_team) +
                    teams_with_extra * (average_man_per_team + 1) * (cur_test.men -
(average_man_per_team + 1))
                ) / 2;
    }

    for (int i = 0; i < t; i++) {
        printf("%i\n", max_fights[i]);
    }
}
```

- **Задача 2 - Куча камней - 1005**

Кол-во различных исходов раскладывания камней на две кучки при n <= 20 - достаточно мало поэтому, прибегаем к рекурсивному перебору всех вариантов. Выбираем такой в котором минимальная разница весов.

Чтобы было не так больно это все перебирать, поставил проверку: если сумма одной кучки больше чем полусумма всех камней, то возвращаем значение не продолжая раскладывать дальше.

Из todo: не уверен что это бы подошло под формат сдачи конкретной задачи, но возможно как вариант более эффективного способа, пусть и приближенного (на n <= 20, скорее всего выдавало бы точный ответ), можно использовать метод имитации отжига.

Код:

```
//
// Created by Sergey Fedorov on 07/02/2020.
//

#include "../../utils/common_utils_string.c"

// O(2^(n+1)). // Tried to minimise by cutting off comparison with half-sum.
int place_stones(const int *stones, int i, int n, int bunch_1, int bunch_2, int sum){
    int cur_stone = stones[i];
    if (bunch_1 ⩾ sum / 2) {
        return abs(bunch_1*2 - sum);
    } else if (bunch_2 ⩾ sum / 2) {
        return abs(bunch_2*2 - sum);
    } else if (i < n) {
        int var_1 = place_stones(stones, i + 1, n, bunch_1 + cur_stone, bunch_2, sum);
        int var_2 = place_stones(stones, i + 1, n, bunch_1, bunch_2 + cur_stone, sum);
        return var_1 < var_2 ? var_1 : var_2;
    } else {
        return abs(bunch_1 - bunch_2);
    }
}

int main() {
    // 1 ⩽ N ⩽ 20 ⇒ max 2 sym
    int n = str_to_int(get_line(2));

    // 1 ⩽ Wi ⩽ 100 000 ⇒ ~ 5 sym
    char **raw_stones = str_split(get_line(5), ' ');

    int stones[n];
    int weight_sum = 0;
    for (int i = 0; i < n; i++) {
        stones[i] = str_to_int(raw_stones[i]);
        weight_sum += stones[i];
    }
    // qsort_ints(stones, n); Presumably on big N. Sorting will be useful.

    int minimal_diff = place_stones(stones, 0, n, 0, 0, weight_sum);

    printf("%i\n", minimal_diff);
}
```

- # Задача 3 - Дуоны - 1155

Проверка на то что задача решается:
Сумма всех дуоннов делится на 2. Суммы несмежных вершин (а таких набора два) равны меж собой.

Далее работаем с одним набором несмежных вершин, я выбрал `B, D, E, G`. Убираем по одной паре дуоннов за раз, до того момента пока во всех выбранных вершинах не останется по 0 дуоннов. Если вдруг удалить дуоон невозможно, потому что на смежных вершинах нет пары, то можем через три действия (1 добавление и 2 убирания) взять дуоон с несмежной вершины.

Код:

```
//
// Created by Sergey Fedorov on 08/02/2020.
//

#include "../../utils/common_utils_string.c"

int checkIfSolvable(const int cameras[8]){
    int all_sum = 0;
    for (int i = 0; i < 8; ++i) {
        all_sum += cameras[i];
    }

    return (
            all_sum % 2 == 0
            &&
            cameras[0] + cameras[2] + cameras[5] + cameras[7] == cameras[1] + cameras[3]
+ cameras[4] + cameras[6]
            );
}

void simpleReduce(int *first, int *second, char l_1, char l_2){
    (*first)--; (*second)--;
    printf("%c%c-\n", l_1, l_2);
}

void complexReduce(int *first, int *second, char distant) {
    (*first)--; (*second)--;
    if (distant == 'H')      printf("EA+\nBA-\nEH-\n");
    else if (distant == 'F') printf("EA+\nFE-\nDA-\n");
    else if (distant == 'C') printf("FB+\nEF-\nCB-\n");
    else if (distant == 'A') printf("FB+\nGF-\nAB-\n");
}

void reduceVertex(int *cameras, const int cur_connections[5], char *letters) {
    if (cameras[cur_connections[0]] > 0){
        int simpleDone = 0;
        for (int i = 1; i <= 3; i++) {
            if (cameras[cur_connections[i]] > 0) {
                simpleReduce(&cameras[cur_connections[0]], &cameras[cur_connections[i]],
letters[cur_connections[0]], letters[cur_connections[i]]);
                simpleDone = 1;
                break;
            }
        }
        if (!simpleDone) {
```

```
            complexReduce(&cameras[cur_connections[0]], &cameras[cur_connections[4]],
letters[cur_connections[4]]);
        }
    }
}

int main(){

    // 1 2 1 2 2 1 2 1
    // A B C D E F G H
    // 0 1 2 3 4 5 6 7
    int cameras[8];
    char letters[8] = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H'};

    int connections[4][5] = {//   distant
            {1, 0, 2, 5, 7}, // B → A, C, F | H
            {3, 0, 2, 7, 5}, // D → A, C, H | F
            {4, 0, 5, 7, 2}, // E → A, F, H | C
            {6, 7, 2, 5, 0}  // G → H, C, F | A
    };

    // 7 spaces + 8 numbers [0;100] ⇒ ~ 7 * 1 + 8 * 2 = 23 sym
    char **read_symbols = str_split(get_line(23), ' ');

    for (int i = 0; i < 8; ++i) {
        cameras[i] = str_to_int(read_symbols[i]);
    }

    if (!checkIfSolvable(cameras)){
        printf("IMPOSSIBLE");
    } else {
        while(cameras[1] + cameras[3] + cameras[4] + cameras[6] > 0){
            for (int i = 0; i < 4; ++i) {
                reduceVertex(cameras, connections[i], letters);
            }
        }
    }

}
```

- **Задача 4 - Гиперпереход - 1296**

Бежим по очереди гравитационных интенсивностей и аккумулируем наше знание о результате:
Каждый раз прибавляем новую интенсивность к текущей сумме и сравниваем с максимальной за период, если больше, то текущая становится новой максимальной. Если случилось так что сумма стало отрицательной, то дальнейший ее вклад будет только отрицательным а значит делаем ограничение снизу по нулю. Ответом является (естественно) максимум.

Код:

```
//
// Created by Sergey Fedorov on 08/02/2020.
//

#include "../../utils/common_utils_string.c"

int main(){
    // 0 ⩽ N ⩽ 60000 ⟹ max 5 sym
    int n = str_to_int(get_line(5));

    int grav_sum = 0;
    int max_grav = 0;

    for (int i = 0; i < n; ++i) {
        // -30000 ⩽ p_i ⩽ +30000 ⟹ 6 sym
        int cur_intensity = str_to_int(get_line(6));
        grav_sum += cur_intensity;

        if (grav_sum < 0){
            grav_sum = 0;
        } else if (grav_sum ⩾ max_grav) {
            max_grav = grav_sum;
        }
    }

    printf("%i", max_grav);
}
```

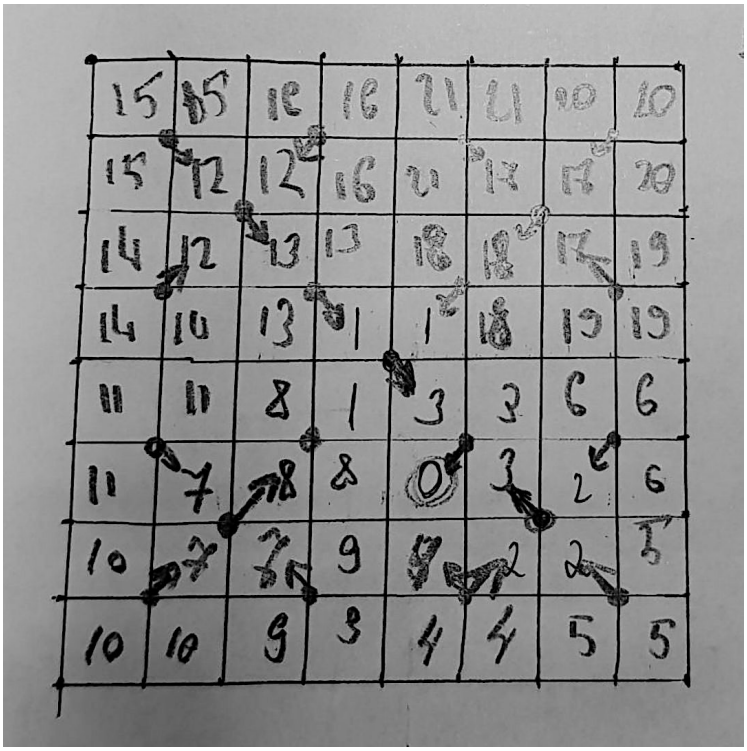
- **Задача 5 - Игроки - 1401**

Заполнять квадрат будем, рекурсивно спускаясь путем деления квадрат на четыре равных квадрата и сравнения позиции пропущенной точки (далее дырка) с центром квадрата.

Кстати тут можно заметить что так как стороны квадрата $2^n x 2^n$ то всегда можно поделить квадрат на 4 части, а значит всегда можно надеятся на решение. (можно вывести формулу про делимость общего кол-ва ячеек без дырки на 3, так как каждая фигура заполняет 3 ячееки, но это всего лишь необходимое условие, но не достаточное для того чтобы сказать что точно все получится расставить. Но чисто для проверки при n <= 9 каждый раз это условие выполняется). Выглядит так, что Чичиков прав и всегда можно заполнить такой квадрат.

На каждой итерации ищем центр квадрата и смотрим в какой стороне (четверти) от него находится дырка. В смежные от центра ячейки, кроме той, которая смотрит в сторону дырки, ставим фигуру, и рекурсивно идем в четверть-квадрат с дыркой. Последним шагом будет квадрат размером 2x2. Затем, рекурсивно идем в оставшиеся 3 стороны, однако теперь за дырку будем каждый раз считать ту ячейку которую заполнили фигурой смежной текущему центру.

Графический пример:



Код:

```
//
// Created by Sergey Fedorov on 13/02/2020.
//

#include "../../utils/common_utils_string.c"

#include <math.h>

typedef struct{
    int _1, _2;
} tuple_2;
```

```c
int fig_atom = 0;

void printMatrix(int width, int height, int **matrix){
    for (int y = 0; y < height; ++y) {
        for (int x = 0; x < width; ++x) {
            printf("%i", matrix[y][x]);
            if (x =≠ width - 1){
                printf(" ");
            }
        }
        printf("\n");
    }
}

void generateMatrix(int **currentMatrix, tuple_2 range_x, tuple_2 range_y, tuple_2
ex_point){
    fig_atom++;
    int x_len = range_x._2 - range_x._1;
    int y_len = range_y._2 - range_y._1;

    int middle_coord_x = range_x._1 + (x_len) / 2;
    int middle_coord_y = range_y._1 + (y_len) / 2;

    int is_smallest = (x_len ⩽ 2 || y_len ⩽ 2) ? 1 : 0;

    if /*left_top_quarter*/          (ex_point._1 ⩽ middle_coord_x && ex_point._2 ⩽
middle_coord_y) {
        currentMatrix[middle_coord_x+1][middle_coord_y] = fig_atom;
        currentMatrix[middle_coord_x][middle_coord_y+1] = fig_atom;
        currentMatrix[middle_coord_x+1][middle_coord_y+1] = fig_atom;

        if (is_smallest) return;

        generateMatrix(currentMatrix, (tuple_2) {range_x._1, middle_coord_x}, (tuple_2)
{range_y._1, middle_coord_y}, ex_point);

        generateMatrix(currentMatrix, (tuple_2) {range_x._1, middle_coord_x}, (tuple_2)
{middle_coord_y + 1, range_y._2}, (tuple_2) {middle_coord_x, middle_coord_y + 1});
        generateMatrix(currentMatrix, (tuple_2) {middle_coord_x + 1, range_x._2},
(tuple_2) {range_y._1, middle_coord_y}, (tuple_2) {middle_coord_x + 1, middle_coord_y});
        generateMatrix(currentMatrix, (tuple_2) {middle_coord_x + 1, range_x._2},
(tuple_2) {middle_coord_y + 1, range_y._2}, (tuple_2) {middle_coord_x + 1, middle_coord_y
+ 1});
    } else if /*left_bottom_quarter*/ (ex_point._1 ⩽ middle_coord_x && ex_point._2 >
middle_coord_y) {
        currentMatrix[middle_coord_x][middle_coord_y] = fig_atom;
        currentMatrix[middle_coord_x+1][middle_coord_y] = fig_atom;
        currentMatrix[middle_coord_x+1][middle_coord_y+1] = fig_atom;

        if (is_smallest) return;

        generateMatrix(currentMatrix, (tuple_2) {range_x._1, middle_coord_x}, (tuple_2)
{middle_coord_y + 1, range_y._2}, ex_point);

        generateMatrix(currentMatrix, (tuple_2) {range_x._1, middle_coord_x}, (tuple_2)
{range_y._1, middle_coord_y}, (tuple_2) {middle_coord_x, middle_coord_y});
        generateMatrix(currentMatrix, (tuple_2) {middle_coord_x + 1, range_x._2},
(tuple_2) {range_y._1, middle_coord_y}, (tuple_2) {middle_coord_x + 1, middle_coord_y});
```

```
        generateMatrix(currentMatrix, (tuple_2) {middle_coord_x + 1, range_x._2},
(tuple_2) {middle_coord_y + 1, range_y._2}, (tuple_2) {middle_coord_x + 1, middle_coord_y
+ 1});
    } else if /*right_top_quarter*/   (ex_point._1 > middle_coord_x && ex_point._2 ⩽
middle_coord_y) {
        currentMatrix[middle_coord_x][middle_coord_y] = fig_atom;
        currentMatrix[middle_coord_x][middle_coord_y+1] = fig_atom;
        currentMatrix[middle_coord_x+1][middle_coord_y+1] = fig_atom;

        if (is_smallest) return;

        generateMatrix(currentMatrix, (tuple_2) {middle_coord_x + 1, range_x._2},
(tuple_2) {range_y._1, middle_coord_y}, ex_point);

        generateMatrix(currentMatrix, (tuple_2) {range_x._1, middle_coord_x}, (tuple_2)
{range_y._1, middle_coord_y}, (tuple_2) {middle_coord_x, middle_coord_y});
        generateMatrix(currentMatrix, (tuple_2) {range_x._1, middle_coord_x}, (tuple_2)
{middle_coord_y + 1, range_y._2}, (tuple_2) {middle_coord_x, middle_coord_y + 1});
        generateMatrix(currentMatrix, (tuple_2) {middle_coord_x + 1, range_x._2},
(tuple_2) {middle_coord_y + 1, range_y._2}, (tuple_2) {middle_coord_x + 1, middle_coord_y
+ 1});
    } else if /*right_bottom_quarter*/ (ex_point._1 > middle_coord_x && ex_point._2 >
middle_coord_y) {
        currentMatrix[middle_coord_x][middle_coord_y] = fig_atom;
        currentMatrix[middle_coord_x+1][middle_coord_y] = fig_atom;
        currentMatrix[middle_coord_x][middle_coord_y+1] = fig_atom;

        if (is_smallest) return;

        generateMatrix(currentMatrix, (tuple_2) {middle_coord_x + 1, range_x._2},
(tuple_2) {middle_coord_y + 1, range_y._2}, ex_point);

        generateMatrix(currentMatrix, (tuple_2) {range_x._1, middle_coord_x}, (tuple_2)
{range_y._1, middle_coord_y}, (tuple_2) {middle_coord_x, middle_coord_y});
        generateMatrix(currentMatrix, (tuple_2) {range_x._1, middle_coord_x}, (tuple_2)
{middle_coord_y + 1, range_y._2}, (tuple_2) {middle_coord_x, middle_coord_y + 1});
        generateMatrix(currentMatrix, (tuple_2) {middle_coord_x + 1, range_x._2},
(tuple_2) {range_y._1, middle_coord_y}, (tuple_2) {middle_coord_x + 1, middle_coord_y});
    }
}

int main(){
    int n = str_to_int(get_line(1));
    int two_to_n = (int) pow(2, n);

    // 512 512 ⟹ max 7 sym
    char **coords = str_split(get_line(7), ' ');
    tuple_2 point = {str_to_int(coords[0]) - 1, str_to_int(coords[1]) - 1};

    int **result = malloc(two_to_n * sizeof(int *));
    for (int i = 0; i < two_to_n; ++i) result[i] = calloc(two_to_n, sizeof(int));

    generateMatrix(result, (tuple_2) {0, two_to_n-1}, (tuple_2) {0, two_to_n-1}, point);

    printMatrix(two_to_n, two_to_n, result);
}
```