Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №1
по “Алгоритмам и структурам данных”
Тимус

Выполнил:
Студент группы P3206
Сорокин А.Н.
Преподаватели:
Косяков М.С.
Тараканов Д.С.

Санкт-Петербург
2024

Задача №1401 “Игроки”
Решение за $O(4^n)$.
Взглянем на следующий рисунок:

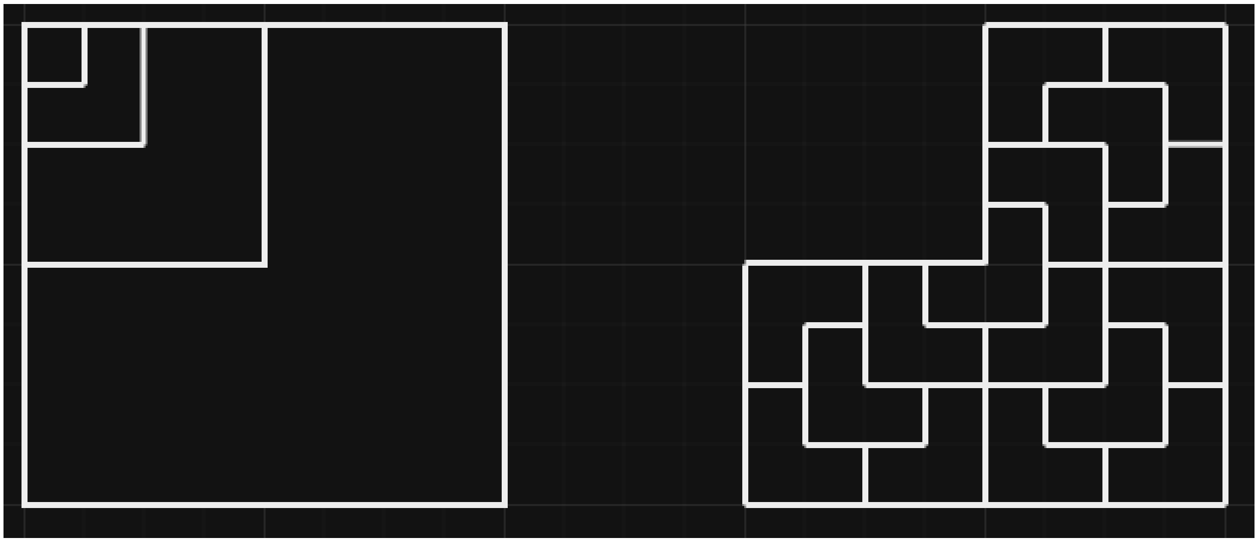

Рисунок слева: приложим к выколотой точке уголок и получим квадрат стороной 2^1. Этот квадрат состоит из квадрата стороны 2^0 и уголка размерности 2^1. Обложим квадрат уголками так, чтобы получился квадрат 2^2 и заметим, что он так же повторяет структуру “квадрат-уголок”. Следовательно, любой квадрат 2^n можно представить как квадрат 2^{n-1} и уголок 2^n.
Рисунок справа: Уголок размерности 2^k где $k > 1$, можно представить в виде четырех уголков рамерности 2^{k-1}, каждый из которых, в свою очередь, если $k - 1 > 1$, так же можно разбить на 4 уголка меньших размерностей.
Поскольку к точке можно приложить уголок с любой стороны, а паттерн “квадрат-уголок” в картине будет проглядываться всегда, независимо от положения точки, делаем вывод, что представленая в задаче ситуация, когда программа должна вернуть “-1”, никогда не возникнет.
Докажем правильность решения.
Из условия знаем, что количество уголков $\frac{2^{2n}-1}{3}$
Если каждый большой уголок порождает 4 маленьких, то для произвольного n сумма уголков равна сумме возрастающей

геометрической прогрессии $S_n = \frac{1*(4^n-1)}{(4-1)} = \frac{4^n-1}{3} = \frac{2^{2n}-1}{3}$, следовательно, выведенный способ правильно считает количество уголков.

Будем заполнять квадрат 2^n x 2^n рекурсивным проходом.
В каждом вызове функции square(n) будем вызывать square(n-1) и angle(n). Перед вызовом определим (sx, sy) – координаты левого верхнего края квадрата меньшей размерности, в котором находится выколотая точка (по этим же координатам определим вид уголка). Перед каждым подсчетом (sx, sy) будем нормировать (x,y) по координатам нового квадрата (см. код).
В функции angle(n) вызываются 4 функции angle(n-1), по виду уголка большей размерности определяется вид уголков меньшей размерности (см. рисунок).

Код решения:

```
#include <cmath>
#include <iostream>
#include <vector>
#define UP_LEFT 1
#define UP_RIGHT 2
#define DOWN_LEFT 3
#define DOWN_RIGHT 4
using namespace std;

/*
 * angle positions:
 *    X X   XX XX
 * XX XX   X X
 * 1    2    3    4
 */
void print_m(vector<vector<int>> m) {
  for (vector<int> v : m) {
    for (int i : v) {
      cout << i << " ";
    }
    cout << "\n";
```

```
  }
}
void angle(int n, int sx, int sy, int *c, vector<vector<int>> *matrix,
           int pos) {
  if (n == 1) {
    switch (pos) {
    case UP_LEFT: {
      matrix->at(sy)[sx] = *c;
      matrix->at(sy - 1)[sx] = *c;
      matrix->at(sy)[sx - 1] = *c;
      break;
    }
    case UP_RIGHT: {
      matrix->at(sy - 1)[sx - 1] = *c;
      matrix->at(sy)[sx - 1] = *c;
      matrix->at(sy)[sx] = *c;
      break;
    }
    case DOWN_LEFT: {
      matrix->at(sy - 1)[sx - 1] = *c;
      matrix->at(sy - 1)[sx] = *c;
      matrix->at(sy)[sx] = *c;
      break;
    }
    case DOWN_RIGHT: {
      matrix->at(sy - 1)[sx - 1] = *c;
      matrix->at(sy)[sx - 1] = *c;
      matrix->at(sy - 1)[sx] = *c;
      break;
    }
    }
    *c += 1;
    return;
  }
  switch (pos) {
  case UP_LEFT: {
```

```
    angle(n - 1, sx + pow(2, n - 2), sy + pow(2, n - 2), c, matrix, UP_LEFT);
    angle(n - 1, sx + pow(2, n - 1), sy + pow(2, n - 1), c, matrix, UP_LEFT);
    angle(n - 1, sx, sy + pow(2, n - 1), c, matrix, UP_RIGHT);
    angle(n - 1, sx + pow(2, n - 1), sy, c, matrix, DOWN_LEFT);
    break;
  }
  case UP_RIGHT: {
    angle(n - 1, sx + pow(2, n - 2), sy + pow(2, n - 2), c, matrix, UP_RIGHT);
    angle(n - 1, sx, sy + pow(2, n - 1), c, matrix, UP_RIGHT);
    angle(n - 1, sx, sy, c, matrix, DOWN_RIGHT);
    angle(n - 1, sx + pow(2, n - 1), sy + pow(2, n - 1), c, matrix, UP_LEFT);
    break;
  }
  case DOWN_LEFT: {
    angle(n - 1, sx + pow(2, n - 2), sy + pow(2, n - 2), c, matrix,
DOWN_LEFT);
    angle(n - 1, sx + pow(2, n - 1), sy, c, matrix, DOWN_LEFT);
    angle(n - 1, sx, sy, c, matrix, DOWN_RIGHT);
    angle(n - 1, sx + pow(2, n - 1), sy + pow(2, n - 1), c, matrix, UP_LEFT);
    break;
  }
  case DOWN_RIGHT:
    angle(n - 1, sx + pow(2, n - 2), sy + pow(2, n - 2), c, matrix,
DOWN_RIGHT);
    angle(n - 1, sx, sy, c, matrix, DOWN_RIGHT);
    angle(n - 1, sx + pow(2, n - 1), sy, c, matrix, DOWN_LEFT);
    angle(n - 1, sx, sy + pow(2, n - 1), c, matrix, UP_RIGHT);
    break;
  }
}

void square(int n, int sx, int sy, int x, int y, int *c,
            vector<vector<int>> *matrix) {
  if (n == 0) {
    *c += 1;
    return;
```

```
  }
  // normalize
  if ((sx != 1) || (sy != 1)) {
    x = (x <= pow(2, n)) ? x : x - pow(2, n);
    y = (y <= pow(2, n)) ? y : y - pow(2, n);
  }

  int new_sx = (x <= pow(2, n - 1)) ? sx : sx + pow(2, n - 1);
  int new_sy = (y <= pow(2, n - 1)) ? sy : sy + pow(2, n - 1);
  int pos = 0;
  if ((sx == new_sx) && (sy == new_sy)) {
    pos = UP_LEFT;
  } else if (new_sy == sy) {
    pos = UP_RIGHT;
  } else if (new_sx == sx) {
    pos = DOWN_LEFT;
  } else {
    pos = DOWN_RIGHT;
  }

  square(n - 1, new_sx, new_sy, x, y, c, matrix);
  angle(n, sx, sy, c, matrix, pos);
}

int main() {
  int n, x, y, c = 0;
  cin >> n >> y >> x;
  int buf = 0;

  vector<vector<int>> matrix;
  for (int i = 0; i < pow(2, n); ++i) {
    vector<int> m(pow(2, n), 0);
    matrix.push_back(m);
  }

  square(n, 1, 1, x, y, &c, &matrix);
```

```
  print_m(matrix);

  return 0;
}
```

Задача №2025 “Стенка на стенку”
Решение за O(n) (на каждый из тестов).
Для получения максимального количества боев необходимо, чтобы команд было как можно больше (k) и количество игроков в командах должно быть равномерно распределено (~ $\frac{n}{k}$). Тогда если представить команды как массив вида $x_1, x_2, \ldots x_k$, количество команд можно найти по следующей формуле:

$$\frac{1}{2}\sum_{i=1}^{k} x_i * (n - x_i)$$

Данная формула работает следующим образом: количество игроков в каждой конкретной команде перемножается на количество игроков за ее пределами (не в этой конкретной команде). Для каждой команды считывается произведение, результаты складываются и делятся на 2 (на каждый бой 1-2 приходится его зеркально отраженная копия 2-1).

Докажем справедливость следующего утверждения:
При равномерном распределении игроков по командам максимальное количество боев возможно только при максимально возможном количестве команд.

Пусть в k командах $\frac{n}{k}$ игроков, а в k-1 командах $\frac{n}{k-1}$ игроков.
Сравним $\frac{1}{2}\sum_{i=1}^{k} \frac{n}{k}(n - \frac{n}{k})$ и $\frac{1}{2}\sum_{i=1}^{k-1} \frac{n}{k-1}(n - \frac{n}{k-1})$

$$\frac{k*n}{k}(n - \frac{n}{k}) \qquad \frac{(k-1)*n}{k-1}(n - \frac{n}{k-1})$$

$$n^2(1 - \frac{1}{k}) \ > \ n^2(1 - \frac{1}{k-1})$$

Для k произведение больше, чем для k-1 при любом k > 2.

Докажем справедливость следующего утверждения:

При максимальном количестве команд максимальное количество боев возможно только при равномерном распределении игроков.

Пусть один набор из k команд $x, x, \ldots, x$, а другой $x-l, x, x \ldots, x, x+l$, где l > 0

Сравним $\frac{1}{2}\sum_{i=1}^{k} x(n-x)$ и $\frac{1}{2}(\sum_{i=1}^{k-2} x(n-x)+(x+l)(n-(x+l))+(x-l)(n-(x-l)))$

$$\frac{kx(n-x)}{2} > \frac{kx(n-x)-2l^2}{2}$$

Произведение больше при равномерном распределении игроков по командам.

Код решения:

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

using namespace std;

int main() {
  size_t t;
  long long n, k;
  vector<long long> nk, v;
  cin >> t;
  for (size_t i = 0; i < t; ++i) {
    cin >> n >> k;
    nk.clear();
    for (size_t j = 0; j < k; ++j) {
      nk.push_back(n / k);
    }
    for (size_t j = 0; j < (n % k); ++j) {
      ++nk.at(j);
    }

    long long s = 0;
    for (long long ni : nk) {
      s += ni * (n - ni);
    }
```

```
    v.push_back(s / 2);
  }
  for (long long j : v) {
    cout << j << "\n";
  }

  return 0;
}
```