Лекция 9. Метод сканирующей прямой

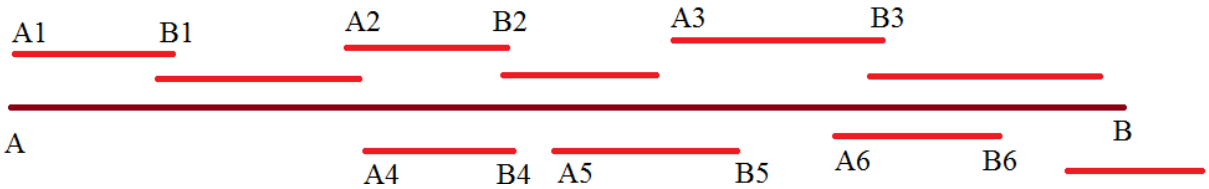
Алгоритм решения задачи покрытия отрезка отрезками. Метод сканирующей прямой (scanline). Сложность алгоритма O(N log N). Задача поиска точки, которая покрыта наибольшим количеством отрезков. Длина объединения отрезков на прямой.

Общее представление

Рассмотрим задачу пересечения отрезков на плоскости. Дано N отрезков координатами своих концов. Необходимо определить – есть ли среди них хотя бы два пересекающихся отрезка. Если такие отрезки есть, то необходимо выбрать любой из них. Тривиальный алгоритм решения задачи при помощи перебора всех пар отрезков имеет сложность $O(N^2)$. Возможно уменьшить сложность решения задачи до $O(N \log N)$ при помощи метода сканирующей прямой. Метод сканирующей прямой и будет рассмотрен в данной лекции.

Задача покрытия отрезка отрезками

Пусть имеется некоторый отрезок AB. Пусть имеется N отрезков A_iB_i, расположенных так, как указано на рисунке.

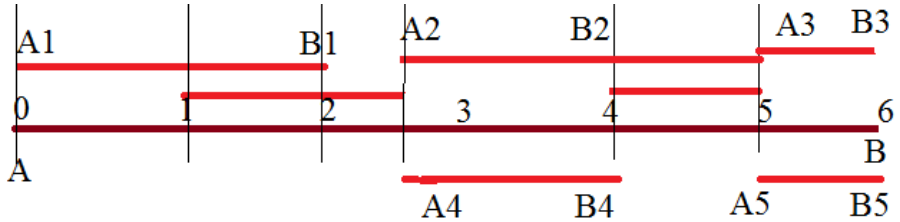


Требуется определить, покрывается ли отрезок AB отрезками A_iB_i. Если отрезок AB покрывается отрезками, то для каждой точки P отрезка AB найдется отрезок A_iB_i, на который может быть спроецирована точка P.

Метод решения задачи. Границы отрезков A_iB_i будем хранить в отсортированном массиве. Вместе с границами отрезков будем хранить указатели на начало и конец отрезков: 1 – начало отрезка, -1 – конец отрезка. Идем по отрезку AB слева направо (двигаем сканирующую вертикальную прямую), и, как только встречаем начало отрезка A_iB_i, прибавляем 1 к счетчику отрезков, как только встречаем конец отрезка – вычитаем 1 из счетчика. Таким образом, речь идет о двух «событиях»:

1. Начался отрезок – встретился левый конец какого-то отрезка.
2. Закончился отрезок – встретился правый конец какого-то отрезка.

Если счетчик все время больше нуля, пока мы идем по отрезку, то это означает, что отрезок покрывается отрезками. Если в какой-то момент счетчик стал равен нулю, то это означает, что в данной координате отрезок не покрывается отрезками.



Отрезок	Координаты точки, тип точки (начало или конец отрезка)	Отсортированный массив границ отрезков. При равных координатах начала отрезков должны идти ранее концов отрезков (необходим собственный компаратор)
A_1B_1	(0,1), (2,-1)	(0,1), (1,1), (2,-1), (2.5, 1), (2.5, 1), (2.5, -1), (4,1), (4, -1), (5,1), (5,1), (5,-1), (5,-1), (6,-1), (6,-1)
A_2B_2	(2.5, 1), (5,-1)	

A_3B_3	(5,1), (6,-1)	
A_4B_4	(2.5, 1), (4, -1)	
A_5B_5	(5,1), (6,-1)	
A_6B_6	(1,1), (2.5, -1)	
A_7B_7	(4,1), (5,-1)	

Проход по отсортированному массиву позволяет наращивать счетчик res на величину type отрезка. Если счетчик res станет в какой-то момент равен нулю, но при этом сканирующая прямая еще не достигла конца отрезка AB, то отрезок AB не покрывается отрезками.

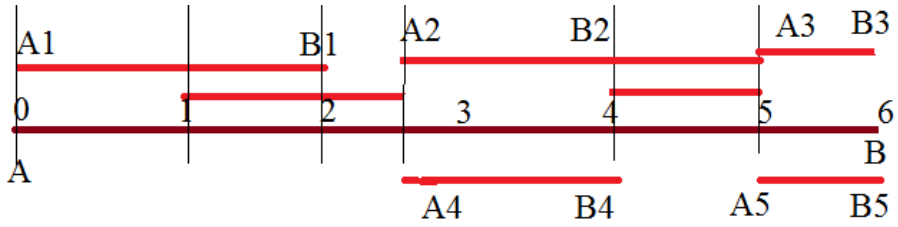
```
struct point   // Структура для точки
{
     int x;
     int type;
     point(int x, int type) : x(x), type(type) {}
};

bool comp(point a, point b)
{
     returna.x < b.x || a.x == b.x && a.type > b.type;
} // Компаратор позволяет начала отрезков хранить раньше концов
int main()
{
     int n, x, y, x1, x2, res = 0;
     cin >> n;
     cin >> x1 >> x2;
     vector<point> ot(2 * n);
     for (int i = 0; i < n; ++i)
     {
          cin >> x >> y;
          if (y < x1 || x > x2)
               // Не учитываем отрезки, лежащие вне исходного
               continue;
          ot[2 * i] = point(x, 1);
          ot[2 * i + 1] = point(x, -1);
     }
     sort(ot.begin(), ot.end(), comp);
     for (auto i : ot)
     {
          res += i.type;
          if (!res)
          {
               cout << "NO";
               return 0;
          }
     }
     cout << "YES";
     return 0;
}
```

Сложность алгоритма $O(N\ log\ N)$

Задача поиска точки, которая покрыта наибольшим количеством отрезков.

Пусть имеется некоторый отрезок AB. Пусть имеется N отрезков A_iB_i, расположенных так, как указано на рисунке.



Найти на отрезке AB такую точку, которая покрыта наибольшим количеством отрезков.

Для решения задачи нужно использовать алгоритм покрытия отрезков отрезками. Максимальное значение res достигаться в той точке, которая покрыта наибольшим количеством отрезков. На приведенном рисунке такой точкой будет, например, точка A_3.

Сложность алгоритма $O(N\ log\ N)$

Длина объединения отрезков на прямой.

Пусть имеется некоторая прямая AB. Пусть имеется N отрезков A_iB_i, расположенных на данной прямой. Необходимо найти объединение отрезков и указать его длину.

Идея решения основана на задаче покрытия отрезка отрезками. Если счетчик в процессе передвижения сканирующей прямой положителен, то увеличиваем длину объединения ans на величину x[i] - x[i - 1].

```
if (res > 0)
{
     ans += ot[i].x - ot[i-1].x;
}
```

Сложность алгоритма $O(N\ log\ N)$.

Разумеется, реализация идеи решения зависит от особенностей задачи. При реализации необходимо учесть все тонкости условия и требуемого результата. Рассмотрим задачу «Забор».

Задача «Забор»

Как известно, красить забор Тому Сойеру помогали многочисленные друзья. Каждый друг покрасил несколько подряд идущих досок, при этом какие-то доски могли быть покрашены несколько раз, а какие-то доски могли остаться непокрашенными. Определите общее количество покрашенных досок.

Входные данные

В первой строке содержится натуральное число $N \le 10^5$ – количество друзей Тома Сойера. Далее идет N пар целых неотрицательных чисел – номер (от начала забора) доски, с которой друг начал красить забор и номер доски, на которой он закончил покраску. Каждый друг покрасил непрерывный участок забора, включая две заданные доски. Номера досок – целые числа от 1 до 10^9.

Выходные данные

Программа должна вывести единственное число – суммарное количество покрашенных досок.

```
struct point
{
	// Структура для хранения точек
	int x;
	int type;
	point(int x, int type) : x(x), type(type) {}
};
bool comp(point a, point b){
	// Компаратор. Начала отрезков хранятся раньше концов при
	// равенстве координат
	if (a.x == b.x)
		return a.type > b.type;
	else
		return a.x < b.x;
}

int main()
{
	int n, x, y, balance = 0;
	long long ans = 0;
	cin >> n;
	vector<point> ot(2 * n);
	for (int i = 0; i < n; ++i){
		cin >> x >> y;
		ot[2 * i] = point(x, 1);
		ot[2 * i + 1] = point(y, -1);
	}
	sort(ot.begin(), ot.end(), comp);
	int l = 0, r = 0; // Левая и правая граница диапазона досок
	for (int i = 0; i < 2 * n; ++i)
	{
		balance += ot[i].type;
		// Вычисления баланса по аналогии со скобочной
		// последовательностью
		if (i != 0 && balance == 0)
		{
			ans += ot[r].x - ot[l].x + 1;
			l = r + 1;
			r++;
		}
		else
			r++;
	}
	cout << ans;
	return 0;
}
```

Объединение прямоугольников

Задача заключается в том, что на плоскости имеется N ($1 \leq N \leq 1000$) прямоугольников, стороны которых параллельны осям координат. Координаты вершин прямоугольников целочисленные и по модулю не превосходят 10^9. Требуется вычислить площадь объединения данных прямоугольников.

Для решения задачи мы воспользуемся методом сканирующей прямой. Для начала выпишем в массив все x-координаты вершин прямоугольников и отсортируем их по возрастанию. В другой массив выпишем все горизонтальные отрезки, являющиеся сторонами прямоугольников. Каждый такой отрезок имеет свою y-координату, координаты левого и правого концов по оси x, а также тип отрезка – открывающий или закрывающий. Так как мы будем перебирать отрезки в порядке возрастания y-координат, то в прямоугольнике нижний отрезок считается открывающим, а верхний – закрывающим. Отсортируем отрезки по y-координате.

Далее перейдем к подсчету площади объединения. Будем идти по x-координатам слева направо. Пусть мы зафиксировали текущую координату x_i. Рассмотрим вертикальную полосу от x_i до x_{i+1} и посчитаем площадь объединения внутри этой полосы. Сделать это несложно, так как размер полосы по горизонтали равен $x_{i+1} - x_i$, поэтому остается только посчитать суммарную длину объединения по вертикали. Сделать это можно с помощью сканирующей прямой, проходя по отрезкам снизу вверх. Данная задача эквивалентна задаче о длине объединения отрезков на прямой. Важно, что при подсчете необходимо игнорировать отрезки, не пересекающие данную полосу. После этого к ответу достаточно прибавить произведение ширины полосы на найденную длину объединения по вертикали.

Таким образом, получаем асимптотику решения $O(N^2)$.

```
struct Segment // Структура для горизонтального отрезка
{
    int xl, xr, y, type;
    Segment(int xl, int xr, int y, int type) : xl(xl), xr(xr),
y(y), type(type) {}  // Конструктор
    bool operator<(const Segment& segment) const
    {
        // Оператор < для сортировки отрезков
        return y < segment.y;
    }
};

int main()
```

```
{
    int n;  // Количество прямоугольников
    cin >> n;
    vector<int> x;  // Вектор для x-координат
    vector<Segment> seg; // Вектор для горизонтальных отрезков
    for (int i = 0; i < n; ++i)
    {
        // Считываем прямоугольники
        int x1, y1, x2, y2;
        cin >> x1 >> y1 >> x2 >> y2;
        // x1, y1 - левый нижний угол, x2, y2 - правый верхний
        // Добавляем x-координаты в отдельный вектор
        x.push_back(x1);
        x.push_back(x2);
        // Нижний отрезок открывает прямоугольник,
        seg.push_back(Segment(x1, x2, y1, 1));
        // а верхний - закрывает
        seg.push_back(Segment(x1, x2, y2, -1));
    }
    sort(x.begin(), x.end());    // Сортируем координаты
    sort(seg.begin(), seg.end());    // и отрезки
    ll answer = 0;  // Суммарная площадь - ответ на задачу
    for (int i = 1; i < 2 * n; ++i)
    {
        // Перебираем вертикальную полосу
        int prevY = 0, cnt = 0;
        // Переменные для координаты первого открытого
        // прямоугольника и количества открытых прямоугольников
        for (int j = 0; j < 2 * n; ++j)
        {
            // Перебираем горизонтальный отрезок
            if (seg[j].xr <= x[i - 1] || seg[j].xl >= x[i])
            {
                // Если отрезок не пересекает текущую полосу -
                // пропускаем его
                continue;
            }
            if (cnt == 0)
            {
                // Если текущий отрезок - первый, запоминаем его
                prevY = seg[j].y;
            }
            cnt += seg[j].type;
            // Увеличиваем или уменьшаем кол-во открытых
            // отрезков
            if (cnt == 0)
            {
                // Если все прямоугольники закрылись -
                // прибавляем их площадь
```

```
                answer += (ll)(seg[j].y - prevY) * (ll)(x[i] -
x[i - 1]);
            }
        }
    }
    cout << answer;
    return 0;
}
```

Литература

1. Длина объединения отрезков на прямой за O (N log N). http://www.e-maxx-ru.1gb.ru/algo/length_of_segments_union
2. Зимняя школа по программированию. Харьков, ХНУРЭ, 2010. День третий. Сканирующая прямая.
3. Видеозаписи лекций ЛКШ. Отрезки на прямой. http://sis.khashaev.ru/2013/july/b-prime/042VezYjC7g/
4. Сайт дистанционной подготовки по информатике. http://informatics.mccme.ru/