Resolução do problema 'Arte Valiosa' utilizando a estrutura de dados Disjoint Set Union (DSU)

Álex Sousa Cruz[1], **Wladimir Araujo Tavares**[1]

[1] Universidade Federal do Ceara (UFC) - Campus Quixadá
Av. Jose de Freitas Queiroz, 5003 – Cedro, 63902-580 - Quixadá-CE

{alexsousa1435, wladimirufc}@gmail.com

Abstract. *This paper describes a possible solution for the "Arte valiosa" problem that appeared in the regional phase of the 2019 XXIV Maratona de programação da SBC. The approach used to solve the problem was to reduce it, which apparently would be from geometry, to just a simple analysis using the DSU data structure to facilitate finding the answer.*

Resumo. *Este artigo descreve uma possível solução para o problema "Arte Valiosa" que apareceu na fase regional da XXIV Maratona de programação da SBC 2019. Nela, a abordagem usada para resolução do problema foi de reduzir o problema, que aparentemente seria de geometria, para apenas uma análise simples de conjuntos usando a estrutura de dados DSU para facilitar o encontro da resposta.*

1. Introdução

Anualmente ocorre o evento chamado Maratona de Programação organizada pela Sociedade Brasileira de Computação (SBC) e nesse evento existente desde o ano de 1996 é realizada uma prova modo contest com vários problemas de computação para ser realizada por equipes de três membros durante cinco horas e que por meio de um computador deverão escrever soluções para os problemas computacionais presentes nessa prova.

No ano de 2019 foi realizada a XXIV Maratona de Programação da SBC que trazia logo de início o problema "Arte Valiosa", para a qual durante o contest foram submetidas 482 tentativas de acerto e apenas 106 foram consideradas corretas pelo judge BOCA de um total de 687 equipes participantes. Por ser um problema que requer uma interpretação geométrica para ser solucionado, é justificável as poucas respostas aceitas durante a prova.

Brevemente, o problema descreve as dimensões de um salão retangular onde existe uma entrada e transversalmente na outra extremidade uma arte valiosa. Nesse salão, há também sensores que conseguem detectar a presença de alguém até determinado raio porém o tamanho desse raio é específico para cada sensor. Sabendo que as posições da arte e a entrada são fixas, deve-se determinar se há como alguém roubar a arte valiosa sem que seja detectado pelos sensores.

O método utilizado para a solução desse problema foi feito através da percepção que a resposta pode ser dada analisando os limites dos raios presentes no sensores e a intersecção deles. Assim, criando um conjunto de sensores que se intersectam podemos armazenar os limites deles e analisar junto com a dimensão do salão.

2. Preliminares

O algoritmo Disjoint Set Union ou também conhecido como UnionFind por conta das sua principal funcionalidade é uma estrutura de dados muito poderosa para quando precisamos trabalhar com as operações de unir conjuntos e buscar a que conjunto ele pertence [Maxim 2019].

A ideia dele é fazer com que cada elemento seja um nó de uma árvore e ao usar os métodos para unir os conjuntos os nós serem conectados apontando para um nó pai. Esse nó pai representará todo o conjunto. Com essa ideia, precisamos apenas de um vetor de 'pais' onde cada elemento inicialmente aponta para ele mesmo, sendo ele o próprio representando do seu conjunto e quando unimos dois conjuntos escolhemos um dos 'pais' para apontar para o outro elemento. Quando precisamos buscar o elemento que representa o conjunto, precisamos apenas buscar quando o pai de um determinado elemento é ele mesmo, caso contrário eu verifico quem é o pai que aponta para aquele elemento.

Ao fazer várias operações de união, pode-se perceber que o tamanho da árvore gerada por essas operações tende à quantidade de elementos que a formam, fazendo com que a operação *find* custe tanto quanto se estivesse buscando em um array não-ordenado. Para resolver esta questão é feito uma otimização no método de união para que o custo de se buscar o pai do subset seja tão desprezável ao ponto de ter uma complexidade próximo a $O(1)$ [Maxim 2019].

A abordagem utilizada para que se tenha o desempenho desejado foi uma otimização pelo nível das árvores, como 'rank', também conhecido como *Union By Rank*. Nessa otimização, o *rank* inicial de todos os elementos são 0, e quando é efetuada a união de dois elementos, será atribuído o pai àquele que tiver menor rank, que consequentemente levará menos operações de *find* para poder chegar no valor absoluto do pai. Após isso, é acrescido 1 se os elementos tiverem o mesmo valor de *rank*.

3. Descrição do Problema

A Mona Dura é uma das obras de arte mais valiosas do museu da Nlogônia. A famosa pintura fica em exibição num salão retangular de M por N metros. A entrada do salão fica em um canto, e a Mona fica no canto diagonalmente oposto à entrada.

Para impedir roubos, o salão dispõe de sensores de movimento, que são ativados toda noite quando o museu fecha. Cada sensor tem um valor de sensibilidade S_i, tal que o sensor dispara um alarme se detectar qualquer movimento a no máximo S_i metros de distância dele.

Um ladrão invadiu o museu esta noite com a intenção de roubar a Mona Dura. Para isso, ele precisa entrar no salão e chegar até a pintura sem ser detectado por nenhum sensor de movimento. Ou seja, ele tem que manter uma distância maior do que S_i metros do i-ésimo sensor o tempo todo, para todos os sensores.

O ladrão obteve acesso às plantas do museu, e portanto sabe as dimensões do salão e as coordenadas e sensibilidades de cada um dos sensores. Dadas essas informações, sua tarefa é determinar se o roubo é possível ou não.

Entrada

A primeira linha contém três inteiros, M, N e K, as dimensões do salão e o número de sensores de movimento, respectivamente ($10 \leq M, N \leq 10^4, 1 \leq K \leq 1000$). A entrada do salão fica no ponto (0, 0) e a pintura fica no ponto (M, N).

Cada uma das K linhas seguintes corresponde a um dos K sensores e contém três inteiros, X, Y e S, onde (X, Y) indica a localização do sensor e S indica a sua sensibilidade ($0 < X < M, 0 < Y < N, 0 < S \leq 10^4$). Todas as dimensões e coordenadas da entrada são em metros. É garantido que todos os sensores têm coordenadas distintas.

Saída

Seu programa deve produzir uma única linha contendo o caractere 'S' caso seja for possível roubar a pintura, ou o caractere 'N' caso contrário.

4. Solucionando o problema com DSU

Primeiramente, para solucionar o problema pensaremos em que casos o ladrão será impedido de passar pela sala com dois sensores. Caso haja espaço entre dois sensores, o ladrão consegue passar, caso contrário ele será identificado. Para que não exista espaço para que ele não passe, precisamos que os raios dos sensores toquem as bordas do salão e também toquem ambos os sensores de tal forma que é formado uma barreira que impede de chegar ao ponto M, N. Analisando isso, é possível ver que os casos que essa barreira é formada será quando um conjunto de sensores que estão em intersecção tocam quando:

- O raio de um conjunto de sensores que se intersectam toca a **borda esquerda** e a **borda superior** do salão;
- O raio de um conjunto de sensores que se intersectam toca a **borda esquerda** e a **borda direita** do salão;
- O raio de um conjunto de sensores que se intersectam toca a **borda inferior** e a **borda superior** do salão;
- O raio de um conjunto de sensores que se intersectam toca a **borda inferior** e a **borda direita** do salão.

Conhecendo esses quatro casos, podemos utilizar a estrutura DSU para que sejam criados conjuntos de sensores que se intersectam e agrupá-los em um mesmo conjunto e após isso podemos efetuar a análise de todos eles. Inicialmente, podemos iniciar à DSU dizendo que cada sensor faz parte de um conjunto isolado dos outros.

Sendo assim, o primeiro passo será conhecer quais pares de sensores tem essa propriedade. Para isso, olharemos os pares i, j tal que $1 \leq i, j \leq K, i \neq j$ e também criar uma função *intersec(i, j)* que verifica se dois sensores se intersectam. Caso se intersectem, eles serão unidos pelo método *union_sets(i,j)* da estrutura de dados DSU. Após isso, podemos tratar cada subconjunto buscando os limites inferiores, superiores, lateral direito e esquerdo dos sensores pela seguinte relação:

- Limite inferior (*MaxY*): coordenada cartesiana Y + raio do sensor i;
- Limite superior (*MinY*): coordenada cartesiana Y - raio do sensor i;
- Limite lateral esquerdo (*MinX*): coordenada cartesiana X - raio do sensor i;
- Limite lateral direito (*MaxX*): coordenada cartesiana X + raio do sensor i.

Com esses dados, podemos utilizar o método de verificação descrito anteriormente, juntamente à estrutura DSU e solucionar o problema.

Algorithm 1 Solução para o problema 'Arte Valiosa'

```
for all s in S do
    make_set(s)
end for
for all j in S do
    for all i in S do
        if i ≠ j then
            if intersec(i, j) then
                union_sets(i, j)
            end if
        end if
    end for
end for
answer := 'S'
for all subset in DSU do
    for all s in subset do
        minX := min(s.x - s.radius, minX)
        minY := min(s.y - s.radius, minY)
        maxX := max(s.x + s.radius, maxX)
        maxY := max(s.y + s.radius, maxY)
    end for
    if (minX ≤ 0 and (minY ≤ or maxX ≥ N)) or (maxY ≥ M and (maxY ≤ 0 or
    maxX ≥ N)) then
        answer := 'N'
    end if
end for
```

5. Conclusão

Através da metodologia abordada no problema, conseguiu-se obter êxito na criação de uma solução do problema que apareceu na XXIV Maratona de Programação da SBC de 2019 'Arte Valiosa'. Utilizando os passos descritos no pseudocódigo 1 foi escrito uma versão na linguagem C++ e submetida no juiz online *beecrowd* que foi aceita dentro da plataforma dentro dos limites de tempo de execução e memória. O código-fonte também pode ser visto em um repositório[1] da escrita desse artigo. A respeito de sua complexidade temos que ao analisar todos os pares no primeiro passo do algoritmo teremos um total de N^2 verificações e para a verificação de pertencer ao mesmo conjunto $O(1)$ por otimização *Union By Rank* e $O(1)$ por amortização da função inversa de *Ackermann* que está presente ao fazer ao unir dois conjuntos. Nessa primeira parte já se pode ter definido uma complexidade de $K^2 \times (2K)$ implicando $O(K^2)$. Quando se analisa os passos para obter a resposta final, temos um total de W iterações com $W = size(DSU)$. Ao final a complexidade final é de $O(K^2)$. Para programas em C++ uma aproximação de 10^8 operações próximo a 1 segundo e quando vimos o tamanho máximo de K e seu máximo de iteração (10^6) consegue-se concluir que o algoritmo atende aos tempos 1s de tempo limite do algoritmo.

[1] https://github.com/alequisk/semana-universitaria/blob/main/2022/codigo.cpp

Referências

[Maxim 2019] Maxim, I. (2019). Disjoint set union. In *E-Maxx, Algorithms translations and community additions*, pages 120–133. Ivanov Maxim.