Teste Programador C++ - Incidência de Luminosidade

Imagine uma região bidimensional na qual diversas fontes de luz estão distribuídas em diferentes posições. Cada fonte emite uma luz que se propaga pelo espaço, alcançando os pontos ao seu redor. Entretanto, esse ambiente não é completamente livre: ao longo do trajeto da luz existem obstáculos de diferentes formas geométricas (como retângulos, linhas e círculos) que interceptam os feixes e reduzem gradualmente sua intensidade.

O desafio consiste em, dado um ponto P, determinar a luminosidade total que incide sobre ele, levando em conta tanto a contribuição de todas as fontes quanto as perdas provocadas pelas bordas atravessadas dos obstáculos.

Fonte de luz

Cada fonte de luz existe de forma independente, sem interferir na intensidade das demais. Sua luminosidade é reduzida apenas quando atravessa um obstáculo. Além disso, cada fonte pode possuir um valor inicial distinto, variando de **0 a 1000**.

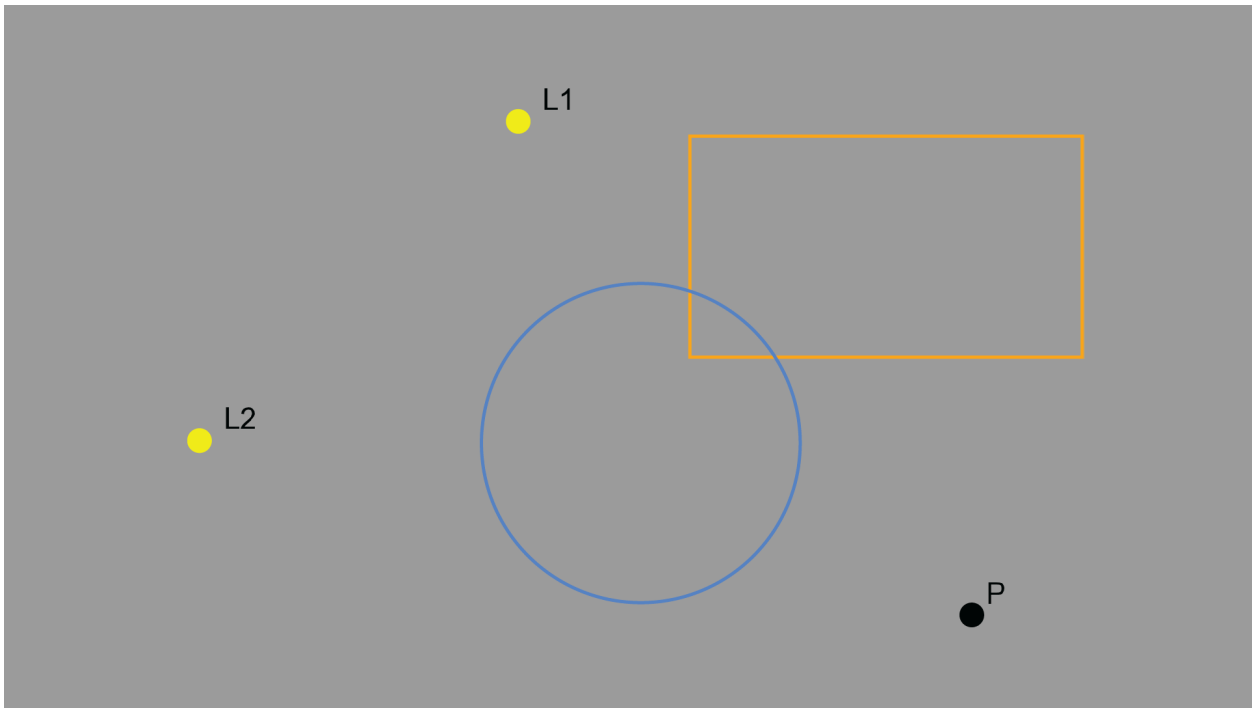
Obstáculos

Os obstáculos são representados como figuras ocas, sendo suas fronteiras as responsáveis por reduzir a luminosidade. Assim, um feixe de luz que atravessa um retângulo, entrando e saindo da figura, tem sua intensidade reduzida duas vezes. Da mesma forma, se uma fonte de luz estiver dentro de um obstáculo, qualquer ponto localizado fora dele sofrerá ao menos uma redução.

De forma análoga às fontes de luz, cada obstáculo possui uma taxa de redução individual, que pode variar entre 1% e 100%. Além disso, obstáculos podem ser sobrepostos e, nesse caso, suas fronteiras compartilham o mesmo espaço, em que cada uma aplica a sua redução sobre o feixe de luz.

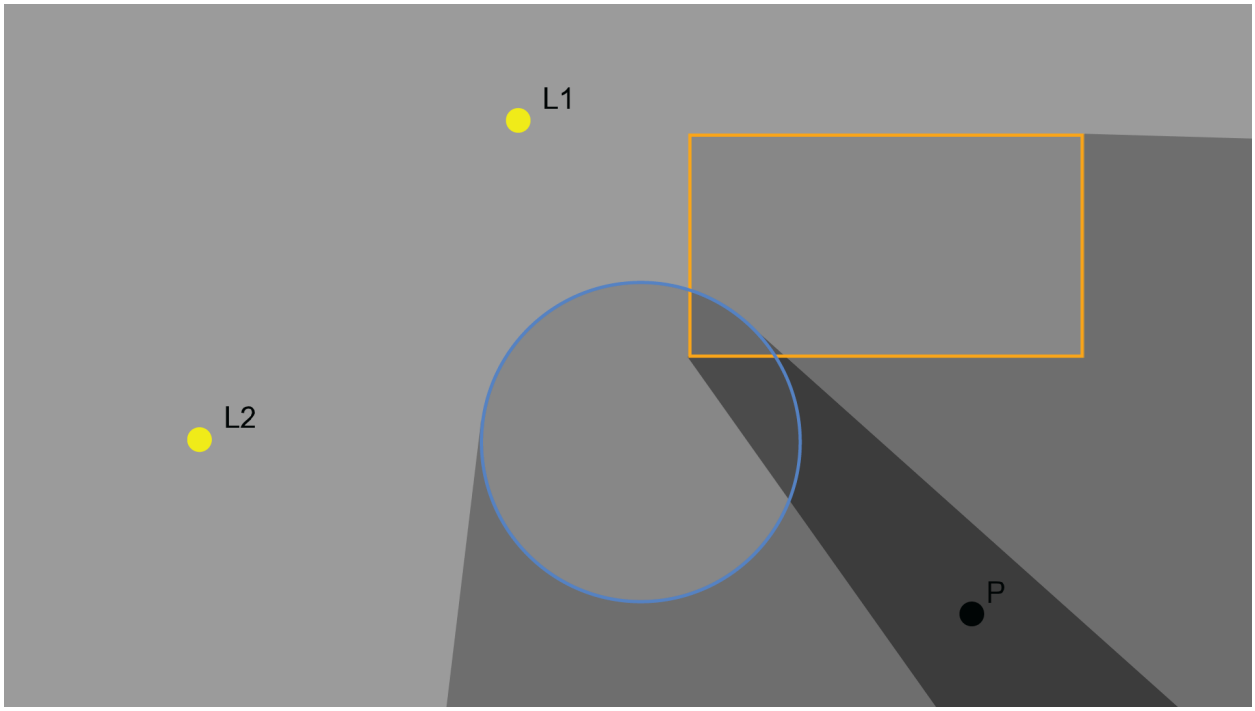
Exemplo

Neste exemplo, temos uma região com duas fontes de luz, **L1** e **L2**, com luminosidades de **100** e **300**, respectivamente. Além disso, há dois obstáculos, um **retângulo** e um **círculo**, ambos com taxa de redução de **50%**.

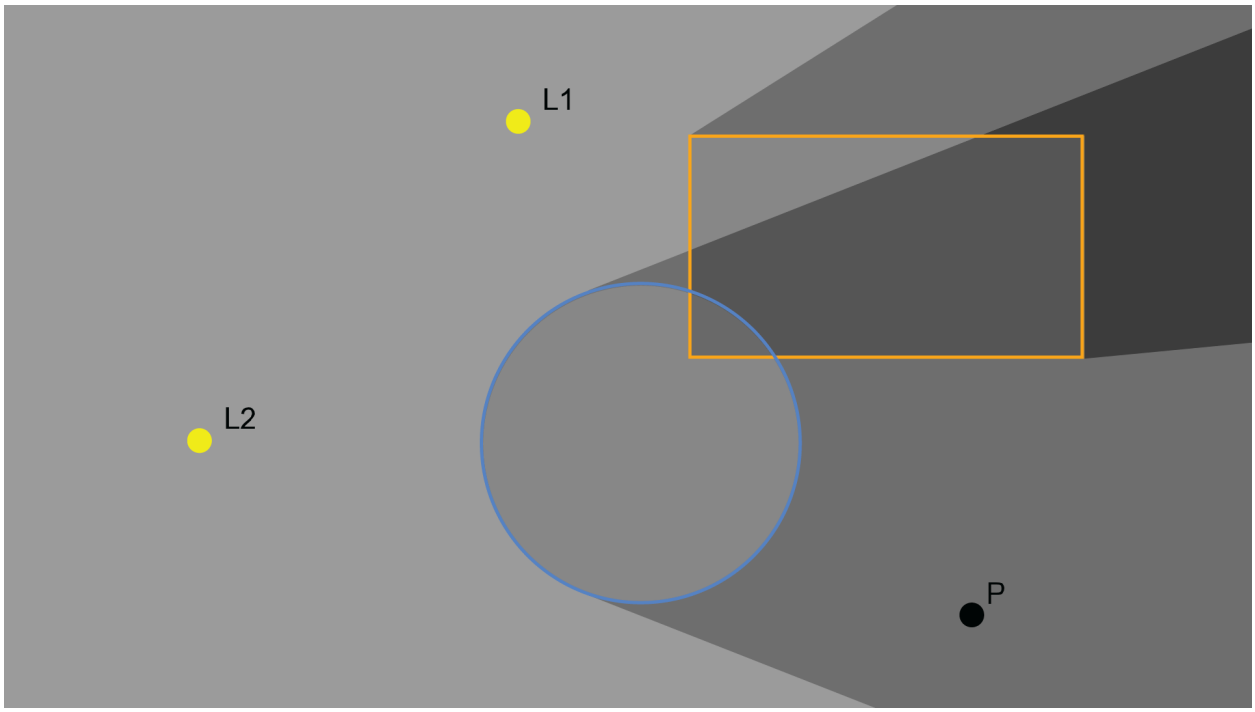


Para calcular a luminosidade total incidente em P, devemos calcular a luminosidade oriunda de cada fonte de luz.

1. L1: A luz de L1 atravessa quatro bordas, cada uma reduzindo em 50% a intensidade, logo: $P1 = 100 * 50\% * 50\% * 50\% * 50\% = 6,25$



2. A luz de L2 atravessa duas bordas, cada uma reduzindo em 50% a intensidade, logo:



$$P2 = 300 * 50\% * 50\% = 75$$

Portanto, a **luminosidade total incidente em P** é:

$$P = P1 + P2$$
$$P = 6.25 + 75 = 81.25$$

Entrada

O programa deve ler um arquivo de texto, nomeado ***regiao.txt***, contendo as informações da região. Nesse arquivo, cada linha pode ser interpretada como um comando no seguinte formato:

```
comando parametro1 parametro2 parametro3 ... parametro99
```

No formato acima, o primeiro valor lido de cada linha é o identificador do comando, e os demais, seus parâmetros. O valor de qualquer ponto deve levar em consideração todas as luzes e obstáculos, portanto, a ordem dos comandos não importa.

Descrição dos comandos

- **R Descrição:** Cria um obstáculo no formato de um retângulo. Parâmetros:
  - id: inteiro (identificador do obstáculo).
  - taxa de redução: inteiro.
  - Origem(Eixo X): inteiro.
  - Origem(Eixo Y): inteiro.
  - Altura: inteiro.
  - Largura: inteiro.
  - Exemplo: Um retângulo do ponto (0,0) até o ponto (20, 100) que reduz 20% da luminosidade: `R 1 20 0 0 100 20`
- **C Descrição:** Cria um obstáculo no formato de um círculo. Parâmetros:
  - Id: inteiro (identificador do obstáculo).
  - Taxa de redução: inteiro.
  - Origem(Eixo X): inteiro.
  - Origem(Eixo Y): inteiro.
  - Raio: inteiro. Exemplo: Um círculo do ponto (50,50) com 10 de raio que reduz 10% da luminosidade: `C 2 10 50 50 10`
- **L Descrição:** Cria um obstáculo no formato de uma linha. Parâmetros:

        - Id: inteiro (identificador do obstáculo).
        - Taxa de redução: inteiro.
        - Origem(Eixo X): inteiro.
        - Origem(Eixo Y): inteiro.
        - Fim(Eixo X): inteiro.
        - Fim(Eixo Y): inteiro. Exemplo: Uma linha do ponto (10,0) até o ponto (20, 0) que reduz 1% da luminosidade: `L 3 1 10 0 20 0`
- **F Descrição:** Cria um fonte de luz Parâmetros:
    - Id: inteiro (identificador da fonte de luz).
    - Luminosidade: real.
    - Coordenada(Eixo X): inteiro.
    - Coordenada(Eixo Y): inteiro. Exemplo: Uma fonte de luz no ponto (5,5) com 101.1 de luminosidade: `F 1 101.1 5 5`
- **P Descrição:** Calcula a incidência total de luminosidade em um ponto Parâmetros:
    - Id: inteiro (identificador do ponto).
    - Coordenada(Eixo X): inteiro.
    - Coordenada(Eixo Y): inteiro. Exemplo: Calcular a incidência de luminosidade do ponto com id = 0 em (1,15): `P 0 1 15`

Obs: Note que alguns feixes podem cruzar exatamente a fronteira dos objetos (ex.: quinas ou tangentes); nesse caso, deve considerar somente uma única redução, já que o feixe não o adentrou.

Saída

Para a saída do programa será impresso (na saída padrão) somente os valores de todos os pontos calculados em ordem crescente ordenado pelo seus IDs. Cada valor impresso deve ter o seguinte formato:

```
P{Id do ponto} = XX.XX
```

Por exemplo:
```
P0 = 101.10
P1 = 15.00
P2 = 0.01
```

O que será avaliado

Durante a avaliação do programa, serão considerados os seguintes critérios:

Correção e precisão
O programa deve funcionar corretamente, produzindo os resultados esperados de acordo com as regras do problema.

Todos os comandos de entrada devem ser interpretados corretamente.

A saída deve seguir exatamente o formato especificado, com valores numéricos consistentes e duas casas decimais.

Casos de borda (como fontes dentro de obstáculos e sobreposições) devem ser tratados de forma coerente.

Clareza e qualidade do código
O código deve ser organizado, legível e bem estruturado.

As funções, variáveis e estruturas devem ter nomes significativos e consistentes.

O fluxo geral do programa deve ser fácil de compreender.

A solução deve estar dividida em partes lógicas (leitura, cálculo e saída), sem acoplamento desnecessário.

O uso de recursos e bibliotecas deve ser feito de forma adequada à linguagem e versão escolhida pelo candidato.

Documentação e comentários
O código deve conter comentários pontuais e úteis, explicando partes importantes da lógica.

Os comentários devem ajudar na leitura, mas não repetir o que já está claro pelo código.

Espera-se um breve resumo no início do arquivo ou das principais funções, descrevendo o propósito geral.

Comentários excessivos, redundantes ou confusos podem reduzir a nota; o objetivo é clareza, não quantidade.

Lógica e raciocínio
Será avaliada a clareza do raciocínio usado para resolver o problema.

A solução deve demonstrar entendimento dos conceitos básicos de geometria e cálculo lógico.

O programa deve ser estruturado de forma coerente e fácil de seguir.

Valorizam-se soluções simples e diretas, evitando código desnecessariamente complexo, aderindo aos princípios de KISS.

Conceitos e boas práticas (bônus técnico)
Não é obrigatório, mas será valorizado se estiver presente e bem aplicado.

Uso adequado de abstrações, interfaces ou padrões de projeto (quando fizer sentido).

Separação clara entre tipos de objetos (fontes, obstáculos, pontos).

Tratamento cuidadoso de erros e validação de entrada.

Estrutura modular que facilita manutenção ou expansão futura.

Esses elementos demonstram domínio de conceitos mais avançados e podem aumentar a nota final se forem aplicados com clareza e propósito.

Eficiência e desempenho
O programa deve funcionar de forma eficiente para o tamanho esperado dos dados.

Não é necessário otimizar ao extremo, mas deve evitar repetições desnecessárias e estruturas ineficientes. A otimização é bem-vinda, mas deve ser equilibrada, para não introduzir muita complexidade e problemas de manutenção para um ganho marginal ou desnecessário em otimização.

A complexidade geral deve ser adequada ao contexto do problema.

Capacidade de explicação
O candidato deve ser capaz de explicar claramente o funcionamento do seu código.

Deve saber justificar suas escolhas de estrutura, lógica e abordagem.

A explicação faz parte da avaliação e demonstra entendimento real da solução.

Desejamos um ótimo teste e boa sorte!