SoulMate Roulette

Tutorial · CC3036 · 2021

Diogo Peralta Cordeiro

Das imagens à ordenação dos pares

A tarefa tem três etapas: interpretar as imagens binárias, representar cada olho pelos vértices do seu invólucro convexo e ordenar todos os pares segundo a distância definida. Não é necessário obter um emparelhamento perfeito. Um cidadão pode aparecer em muitos pares, porque as atribuições finais ficam a cargo dos analistas.

Mantém o enunciado ao lado deste tutorial. O formato das imagens, a normalização e o desempate definem o resultado. Uma alternativa geometricamente razoável pode produzir uma resposta diferente.

Ler e validar o cartão completo

Existem $N = 1337$ imagens de 398 bytes. A entrada tem, portanto, de conter $1337 \times 398 = 532126$ bytes. Lê, no máximo, mais um byte para detetar dados adicionais. Uma única chamada ao método habitual `read` de um fluxo não tem de preencher o destino. A solução Java usa `readNBytes` e verifica o comprimento devolvido.

A imagem i ocupa o intervalo de bytes $[398i, 398(i+1))$. Interpreta os campos do cabeçalho como valores little-endian. Verifica a assinatura, o tamanho, o cabeçalho DIB de 40 bytes, as dimensões, o número de planos, a profundidade de um bit, a ausência de compressão, o início dos píxeis e a paleta preta e branca.

Os 42 bits úteis de uma linha ocupam seis bytes; o BMP preenche a linha até oito bytes. Com os 62 bytes do cabeçalho e da paleta, obtém-se $62 + 42 \times 8 = 398$. Uma altura positiva indica armazenamento de baixo para cima. A solução Java usa `ImageIO` para descodificar os píxeis depois de verificar o cabeçalho.

Não escrevas percentagens durante a leitura. Se a última imagem for inválida, a saída continua a ter de conter apenas a mensagem de erro. Valida o conjunto completo antes de produzir pares.

Construir o invólucro de cada imagem

Recolhe as coordenadas dos píxeis brancos. Menos de três pontos, ou um conjunto inteiramente colinear, torna a entrada inválida.

O invólucro convexo é o menor polígono convexo que contém os pontos. O código fornecido usa Graham scan:

1. Escolhe o ponto com a menor coordenada vertical, desempatando pela coordenada horizontal.
2. Ordena os restantes pontos pelo ângulo em torno desse ponto. Quando o ângulo coincide, coloca primeiro o ponto mais próximo.

3. Mantém uma pilha de candidatos a vértice. Uma viragem no sentido dos ponteiros do relógio elimina o candidato intermédio; numa situação colinear, conserva-se o extremo.

A orientação é dada pelo sinal do produto vetorial:

$$(b_x - a_x)(c_y - a_y) - (b_y - a_y)(c_x - a_x).$$

Os sinais positivo, negativo e nulo distinguem as duas orientações e a colinearidade. Excluem-se os pontos intermédios de arestas retas. A implementação repete o primeiro vértice no fim para fechar o polígono; essa repetição não altera os mínimos nem os máximos das distâncias.

O invólucro é uma escolha de representação exigida pelo exercício. A distância de Hausdorff também pode ser definida para conjuntos de pontos não convexos.

Calcular a distância discreta

Para cada vértice do primeiro invólucro, encontra o vértice mais próximo no segundo. A maior dessas distâncias mínimas é a distância dirigida. Repete no sentido contrário e conserva o maior dos dois resultados.

```
funcao dirigidaAoQuadrado(A, B):
    maior = 0
    para cada ponto a em A:
        menor = infinito
        para cada ponto b em B:
            menor = min(menor, distanciaAoQuadrado(a, b))
        maior = max(maior, menor)
    devolver maior

distancia = sqrt(max(dirigidaAoQuadrado(A, B),
                     dirigidaAoQuadrado(B, A)))
```

As distâncias ao quadrado permitem usar apenas multiplicações e somas no ciclo interior. Como a raiz quadrada é crescente nos números não negativos, aplicá-la depois dos mínimos e máximos dá o mesmo resultado.

Considera os triângulos $A = \{(0,0), (2,0), (0,2)\}$ e $B = \{(1,0), (3,0), (1,2)\}$. Cada vértice tem um vizinho mais próximo a distância 1 no outro triângulo, pelo que ambas as distâncias dirigidas valem 1. A compatibilidade é aproximadamente $98{,}3164\%$, apresentada como `98.32%`. Conjuntos iguais têm distância 0 e compatibilidade `100.00%`.

Não uses a distância de um ponto a uma aresta. O enunciado considera vértices; uma posição no meio de um segmento não é candidata a ponto mais próximo.

Ordenar de forma determinística

Gera cada par uma só vez, com $a < b$. Existem $M = N(N-1)/2 = 893116$ pares. Guarda os identificadores e a compatibilidade sem arredondamento:

$$100 \left(1 - \frac{H(A,B)}{\sqrt{42^2 + 42^2}}\right).$$

Ordena pela percentagem decrescente, depois pelo primeiro identificador crescente e finalmente pelo segundo identificador crescente. Os identificadores são inteiros: a ordem lexicográfica

colocaria `10` antes de `2`. Uma ordenação estável, por si só, não define o desempate quando a coleção inicial não tem uma ordem garantida, como acontece num mapa de dispersão.

Arredonda apenas ao escrever. Duas percentagens apresentadas como `98.32%` podem ter uma ordem estrita. Usa `Locale.ROOT` com `%.2f` para garantir o ponto decimal, mesmo num computador configurado em português. A escrita com buffer é relevante para quase novecentas mil linhas.

Complexidade e limites práticos

Sejam P o número máximo de píxeis brancos por imagem e H o número máximo de vértices de um invólucro. A leitura custa $O(N \times 42^2)$ e os invólucros custam $O(NP \log P)$. A distância direta custa $O(H^2)$ por par e a ordenação custa $O(M \log M)$.

No total, obtém-se $O(N \times 42^2 + NP \log P + N^2H^2 + M \log M)$, com $O(NH + M)$ de armazenamento após a descodificação, além do buffer de entrada. Distingue estas grandezas: o cálculo direto da distância é quadrático nos tamanhos dos invólucros, não linear apenas por os polígonos serem convexos. Os objetos e as cadeias de caracteres Java também consomem memória.

Compilar e verificar

Usa JDK 17 ou posterior:

```
javac Solution.java InputGenerator.java
tar -xzf example_io.tar.gz
java -Xmx550m Solution < example_input.raw > actual.txt
cmp example_output.txt actual.txt
python3 test.py
```

O teste compara a saída completa do exemplo e verifica entrada truncada, bytes adicionais, uma dimensão incorreta e uma forma vazia. Outros casos úteis são invólucros iguais, pontos colineares, formas diferentes com pontuações iguais e identificadores com números de algarismos diferentes.

Para gerar outro cartão:

```
java InputGenerator bmps
sh from_bmps_to_raw.sh bmps generated.raw
java -Xmx550m Solution < generated.raw > generated-output.txt
```

O gerador usa a semente 42. O script concatena os BMP pela ordem numérica dos identificadores e verifica os 1337 ficheiros antes de escrever o cartão. Os créditos dos algoritmos encontram-se em `Solution.java`.