INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA MINAS GERAIS Campus Formiga

Graduação em Ciência da Computação

Disciplina: Projeto e Análise de Algoritmos
Professores: Walace de Almeida Rodrigues
Atividade: 1o Trabalho Prático

Instruções para o trabalho

Este documento descreve os requisitos do trabalho prático. As tarefas serão identicadas pelo sinal ■.

1. Resumo:

Geometria Computacional é o estudo de algoritmos para resolver problemas geométricos em um computador. Esse estudo emergiu como campo de projeto e análise de algoritmos no final da década de 1970 e desde então esta área vem atraindo crescente atenção. Seu sucesso como campo de pesquisa pode por um lado ser explicado pela beleza dos problemas estudados e das soluções obtidas, e por outro lado, pelo domínio de diversas aplicações – computação gráfica, sistemas de informações geográficas (GIS), robótica, e outros – em que algoritmos geométricos têm um papel fundamental.

Diversos tipos de problemas são considerados dentro da geometria computacional e um grupo importante desses são modelados por pontos representados por coordenadas inteiras e por polígonos. Nesse grupo, problemas que envolvem coordenadas em ponto flutuante e/ou regiões definidas por curvas são discretizados durante a modelagem.

O presente trabalho tem como foco o desenvolvimento de estruturas de dados e algoritmos para atacar solucionar os problemas clássicos nessa área que podem ser modelados no plano cartesiano de duas dimensões, representados por coordenadas inteiras, polígonos e círculos.

2. Estruturas de dados:

Os objetos complexos mais comuns encontrados no projeto de algoritmos geométricos são os conjuntos e as sequências (conjuntos ordenados). Estruturas de dados particularmente apropriadas para esses objetos combinatórios complexos são bem descritas na literatura padrão sobre algoritmos. O objetivo aqui é desenvolver um arcabouço elegante para auxiliar na modelagem dos problemas e sua solução computacional, em outras palavras, oferecer as capacidades funcionais e o desempenho computacional necessários – fique atento quanto as restrições!

1) TAD para Conjunto Ordenado

Desenvolva uma TAD para representar conjuntos ordenados segundo as especificações que seguem. Seja S um conjunto representado por uma estrutura de dados e seja u um elemento arbitrário de um conjunto universal em que S é um subconjunto. As operações fundamentais que ocorrem na manipulação desses conjuntos são:

- $S.member(u)$ – Tem-se que $u \in S$? (resposta sim/não)
- $S.insert(u)$ – Adiciona u em S.
- $S.delete(u)$ – Remove u de S.

Considere agora a manipulação de conjuntos de conjuntos. Suponha que $C = \{S_1, S_2, \ldots, S_k\}$ é uma coleção de conjuntos (com interseção vazia por pares). As operações fundamentais nessa coleção são:

- $C.member(A)$ – Tem-se que $A \subseteq C$? (resposta sim/não)
- $C.insert(A)$ – Adiciona A em C, se $A \cap S_i = \varnothing$ para todo $1 \leq i \leq k$.

- $C.delete(A)$ – Remove A de C.
- $C.find(u)$ – Retorna S_j, se $u \in S_j$.

Também são úteis as seguintes operações sobre os conjuntos A e B:

- $union(A, B)$ – Retorna $A \cup B$.
- $intersection(A, B)$ – Retorna $A \cap B$.
- $difference(A, B)$ – Retorna $A - B$.

Quando o conjunto universal é totalmente ordenado, as seguintes operações são importantes:

- $S.min()$ – Retorna o elemento mínimo de S.
- $S.split(u)$ – Particiona S em $\{S_1, S_2\}$ tal que $S_1 = \{v \mid v \in S \text{ e } v \leq u\}$ e $S_2 = S - S_1$. Retorna $C = \{S_1, S_2\}$
- $S.concatenate(A)$ – Assumindo que, para arbitrários $u' \in S$ e $u'' \in A$, tem-se que $u' \leq u''$, faz $S = S \cup A$.

■ Implemente o TAD conjunto ordenado com as operações apresentadas, com $O(n \log n)$ para cada operação. Justifique qualquer opção contrária.

2) Outras estruturas

Juntamente com os conjuntos ordenados, as estruturas de dados padrões (listas, pilhas, filas) são usadas extensivamente nos algoritmos da geometria computacional. Embora a natureza dos problemas geométricos tenha levado muitas vezes ao desenvolvimento de estruturas de dados não convencionais específicas, as estruturas de dados padrões geralmente são suficientes para tratar a grande maioria dos problemas clássicos.

■ Desenvolva representações adequadas (com atenção para atender as complexidades dos algoritmos propostos adiante) para: (1) Ponto, (2) Segmento de reta, (3) Linha, (4) Polígono e (5) Círculo.

3. Problemas clássicos:

■ Desenvolva procedimentos eficientes para resolver cada uma das questões seguintes. Calcule e apresente junto com o seu algoritmo a complexidade da sua solução, considerando que soluções de qualidade muito ruim poderão sofrer penalidades:

- Calcular a distância entre dois pontos.
- Calcular a distância entre um ponto e uma reta.
- Calcular a área de um polígono.
- Calcular a área de uma círculo.
- Predicado convexidade de um polígono. (slide 27-28 do pdf auxiliar)
- Dobrar polígonos em triângulos até um único triângulo. (slide 25 do pdf auxiliar)
- Predicado orientação 2D. (slides 29-31 do pdf auxiliar)
- Predicado qual lado do círculo. (slides 32-33 do pdf auxiliar)
- Encontrar ponto mais próximo de um segmento de reta. (slide 35 do pdf auxiliar)
- Determinar a interseção de segmentos de reta. (slide 37-44 do pdf auxiliar)
- Predicado ponto dentro do polígono. (slides 46-47 do pdf auxiliar)

■ Também desenvolva algoritmos para resolver três dos problemas abaixo em $O(n \log n)$, ou melhor. Você pode escolher entre o problema do fecho convexo e o problema do círculo mínimo, resolva um desses. Os outros dois são obrigatórios. (Combinem entre os grupos de modo a balancear as escolhas: todos os problemas deverão ser atacados pela turma e não havendo acordo a distribuição será feita por sorteio.)

- Problema do par mais próximo – Dados n pontos, queremos encontrar dois deles que estejam à distância mínima. Uma aplicação prática deste problema é em controle de tráfego aéreo: os dois aviões que estão em maior perigo de colisão são aqueles que estão mais próximos.
- Problema do fecho convexo – Convexidade é uma propriedade geométrica bastante importante. Um conjunto de pontos é convexo se, para cada par de pontos no conjunto, o segmento de reta

entre eles está inteiramente contido no conjunto. O problema do fecho convexo consiste em, dados n pontos, encontrar o fecho convexo desses pontos. Uma das aplicações práticas deste problema se encontra em robótica. Se o fecho convexo de um robô não colide com obstáculos então o robô também não colide.

- Problema do círculo mínimo – Dados n pontos, encontrar o círculo de menor raio que contenha todos os pontos. Uma aplicação prática deste problema é na localização de distribuidores de serviços: por exemplo, onde posicionar um hospital de modo a atender toda uma comunidade no menor tempo possível.
- Diagrama de Voronoi – Dado um conjunto S de n pontos no plano, determinar para cada ponto p em S qual é a região $V(p)$ dos pontos do plano que estão mais perto de p do que de qualquer outro ponto em S. As n regiões $V(p)$ formam uma partição do plano chamada de Diagrama de Voronoi. Imagine uma vasta floresta contendo vários pontos de observação de incêndio. O conjunto das árvores que estão mais próximas de um determinado posto p determina a região $V(p)$ das árvores que são de responsabilidade do ponto p.

■ Desenvolva uma interface adequada para realizar a entrada de dados e a apresentação gráfica dos resultados dos algoritmos.

4. Informações gerais:

O trabalho será individual ou em grupo de dois. A data de entrega será em 1-nov-2017. As implementações serão apresentadas nos dias seguintes, a combinar durante as aulas. As linguagens de programação utilizadas nas implementações podem ser: C, C++, Java ou Phyton. Caso deseje utilizar outra, combine antes com o professor.