Geometria Computacional

Departamento de Ciência da Computação – IME/USP

Segundo Semestre de 2007

Algoritmo de Shamos

Deve-se a Shamos o algoritmo de divisão e conquista visto na aula 2 para, dada uma coleção de n pontos no plano, determinar um par de pontos mais próximos da coleção. A parte mais delicada desse algoritmo é provar que ele está correto. **Por que esse algoritmo de fato devolve um par de pontos mais próximos?**

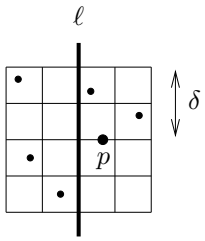
O algoritmo considera uma linha divisória vertical ℓ que deixa metade dos pontos da coleção à esquerda e metade à direita. A idéia do algoritmo é calcular um par de pontos na metade esquerda o mais próximo possível, um par de pontos na metade direita o mais próximo possível, e finalmente, se necessário, um par de pontos o mais próximo possível com um ponto em uma metade e o outro na outra.

A divisão da coleção de pontos nas duas metades, que chamaremos de metade esquerda e metade direita, é feita pela rotina Divide. A rotina usa para tanto dois vetores, p_x e p_y, que apresentam a lista de pontos ordenada um pela coordenada x e outro pela coordenada y. Veja o código da rotina nas transparências da aula 2. O que a rotina Divide devolve efetivamente são dois pares de vetores, (p_x^l, p_y^l) e (p_x^r, p_y^r), que contém a lista dos $\lfloor n/2 \rfloor$ ou $\lceil n/2 \rceil$ pontos da metade esquerda e da metade direita, ordenados pela coordenada x e pela coordenada y respectivamente. É fácil ver que a rotina Divide funciona corretamente, desde que não haja pontos com coordenadas x coincidentes, ou com coordenadas y coincidentes.

A obtenção de um par de pontos mais próximos na metade esquerda e na metade direita é feita recursivamente, e portanto funciona corretamente por um argumento indutivo.

A parte delicada é a final, em que se determina, se necessário, um par de pontos o mais próximo possível com um ponto em uma metade e o outro na outra. Essa é a principal tarefa da rotina Combina. Seja δ o mínimo entre a distância dos dois pares de pontos obtidos recursivamente. A pergunta a ser respondida para garantir que o algoritmo de fato dá a resposta correta é: por que a rotina Combina determina se existe um par de pontos, cada um de um lado da linha ℓ, a uma distância menor que δ e, em caso afirmativo, encontra um tal par de pontos assim, à distância mínima?

Para efetuar essa tarefa, é fácil ver que Combina precisa apenas olhar para os pontos que estão a uma distância não mais que δ da linha ℓ. A primeira etapa de Combina é exatamente selecionar tais pontos. Isso é feito nas linhas 3–6. Fixe a atenção em um dos pontos selecionados, digamos, o ponto p. Considere a figura abaixo. Na figura, cada um dos quadrados está totalmente contido em uma das duas metades. Ademais, cada um dos quadrados tem lado $\delta/2$. Como a diagonal de um tal quadrado mede $\delta/\sqrt{2} < \delta$, não pode haver dois pontos da coleção em um mesmo quadrado destes. (Ou tais pontos estariam em uma mesma metade e estariam a uma distância menor que δ, uma contradição à definição de δ.) Como a linha horizontal do meio da figura passa exatamente pelo ponto p, qualquer ponto selecionado que esteja a uma distância menor que δ de p deve estar num dos quadrados da figura. Ou seja, há no máximo 14 pontos selecionados que podem estar a uma distância menor que δ de p. Destes no máximo 14, não mais que sete estão acima de p.



A segunda etapa de Combina consiste em percorrer os pontos selecionados calculando a distância entre cada um deles e os (no máximo) sete pontos selecionados seguintes em ordem da coordenada y. Observe que a seleção dos pontos, feita nas linhas 3–6 da rotina Combina, já os mantém ordenados pela coordenada y. Apenas um ponto dentre esses sete pontos têm chance de estar a menos do que δ do ponto em questão e na outra metade em relação ao ponto. De todas as distâncias calculadas, a rotina Combina mantém a menor, desde que ela seja menor que δ, e um dos pares que atingiu esse mínimo. Ao final, a rotina devolve ou esse par, ou um dos dois pares obtidos recursivamente, aquele que for o mais próximo destes três.

Exercícios

1). Ajuste o algoritmo dado nas transparências da aula 2 para que funcione mesmo que a coleção de pontos dada inclua pontos com coordenadas x ou y coincidentes.

2). Refine o algoritmo e o argumento que garante que ele funciona para que, na rotina Combina, cada ponto da faixa central seja comparado com menos que sete pontos.
[Note que para a ordem de complexidade do algoritmo tanto faz se para cada $m \in M$ calculamos 7, 6 ou 100000 distâncias—o algoritmo continua tendo complexidade de tempo $O(n \log n)$.]

3). [CLRS 33.4-1] O Prof. Maqui Sperto teve uma idéia genial e veio com um novo esquema para que o algoritmo encontre o par mais-próximo verificando, na sua fase Combinar, somente a distância entre cada ponto $m \in M$ e os 5 pontos que estão a seguir de m em M. A idéia é sempre colocar os pontos da reta l no conjunto E da esquerda. Então, não poderá haver um par de pontos coincidentes sobre a reta l com um ponto em E e outro ponto em D. Portanto, no máximo 6 pontos podem estar na retângulo de dimensões $\delta \times 2\delta$. Onde está a bobagem no esquema proposto pelo professor Sperto.

4). [CLRS 33.4-2] Sem aumentar a complexidade de tempo assintótica do algoritmo, mostre como garantir que o conjunto de pontos passados para a primeira chamada recursiva do algoritmo não contenha pontos coincidentes. Prove que então é suficiente que o algoritmo verifique os 6 (e não 7) pontos que seguem cada ponto em M. Por que não é suficiente verificar somente 5 pontos? Ou é suficiente?

5). Considere a fase Combinar do algoritmo de divisão-e-conquista para o Problema do Par Mais-Próximo, visto em sala de aula. Modifique a rotina Combina para que seja calculada a distância entre cada ponto e apenas pontos do outro lado da partição feita em Divide. Mostre que, nesta fase, para cada ponto na esquerda, é **suficiente** o algoritmo calcular a distância entre dele com no máximo 6 pontos na direira.

6). Para a modificação proposta no exercício anterior, você consegue mostrar um exemplo onde é realmente **necessário** calcularmos a distância entre cada ponto e 6 pontos $d_1, \ldots, d_6$ do outro lado da partição? (Ou seja, o seu exemplo deve mostrar um ponto e da esquerda, digamos, e pontos $d_1, \ldots, d_6$ da direita na proximidade do ponto e de tal forma que se o algoritmo calcula a distância entre e e apenas cada um dos pontos $d_1, \ldots, d_5$ então o algoritmo não devolve o par mais-próximo (que por azar é o par $\{e, d_6\}$.) Se você não conseguir encontrar um tal exemplo, então tente mostrar que é **suficiente** o algoritmo calcular a distância entre cada ponto de um lado e menos do que 6 pontos do outro lado.

7). [CLRS 33.4-3] A distância entre dois pontos pode ser definida de diversas maneiras além da Euclidiana. No plano, a L_m-distância entre dois pontos $p = (p_x, p_y)$ e $q = (q_x, q_y)$ é dada por $(|p_x - q_x|^m + |p_y - q_y|^m)^{1/m}$. Portanto, a distância Euclidiana é a L_2-distância. Modifique o algoritmo de divisão-e-conquista para o Problema do Par Mais Próximo de tal forma que ele devolva o par de pontos mais-próximo em relação a L_1-distância (também conhecida como *Manhattan distance*).

8). [CLRS 33.4-4] Dados dois pontos p e q no plano, a L_∞-distância entre eles é dada por $\max\{|p_x - q_x|, |p_y - q_y|\}$. Modifique o algoritmo para o par mais próximo para que ele encontre um par mais-próximo de acordo com a L_∞-distância.

Observação: Sempre que descrever uma modificação de um algoritmo dado, exiba o resultado da modificação em pseudo-código.