MC558 — Análise de Algoritmos II

Cid C. de Souza   Orlando Lee

31 de maio de 2023

Créditos

A maior parte do conteúdo deste conjunto de slides foi inteiramente baseado em um conjunto de slides preparado pelo Prof. Cid Carvalho de Souza.

Apenas modifiquei um pouco a apresentação e introduzi alguns outros exemplos que pareciam mais adequado para esta instância da disciplina.

Orlando Lee
Novembro de 2016

Contas superior e inferior de um problema

Seja P um problema e suponha que n é um parâmetro que denota o tamanho de uma instância de P.

- Dizemos que P tem cota superior $O(g(n))$ se **existe algum algoritmo** que resolve P com complexidade $O(g(n))$.
- Por exemplo, se P é o problema de ordenação, então o InsertionSort fornece uma cota superior de $O(n^2)$.

  Melhor ainda, o MergeSort fornece uma cota superior de $O(n \log n)$.
- O interesse é encontrar a **melhor/menor** cota superior de um problema. Ou seja, o melhor algoritmo que resolve o problema.

Contas superior e inferior de um problema

Seja P um problema e suponha que n é um parâmetro que denota o tamanho de uma instância de P.

- Dizemos que P tem cota inferior $\Omega(f(n))$ se **qualquer algoritmo** que resolve P tem complexidade $\Omega(f(n))$.
- Por exemplo, o problema de ordenação tem cota inferior $\Omega(n \log n)$.
- Qualquer problema P tem uma cota inferior trivial $\Omega(n)$, onde n é o tamanho da entrada de P. (**Por quê?**)
- O interesse é encontrar a **melhor/maior** cota inferior de um problema.

Contas superior e inferior de um problema

Seja P um problema e suponha que n é um parâmetro que denota o tamanho de uma instância de P.

- Um algoritmo é ótimo para um problema P se sua complexidade coincide com uma cota inferior de P.
- Em outras palavras, temos uma cota superior que coincide com uma cota inferior.
- Por exemplo, o problema da ordenação tem cota inferior $\Omega(n \lg n)$ e existe algoritmo de ordenação de complexidade $O(n \lg n)$ (heapsort e mergesort).
- Não se conhece muitos problemas para os quais as cotas superior e inferior coincidem.

Reduções

Esquema básico de uma redução de Turing:

Problema A:	Problema B:
• Instância: I_A • Solução: S_A	• Instância: I_B • Solução: S_B

Definição. Uma redução do problema A ao problema B é um par de transformações τ_I, τ_S tal que para toda instância I_A de A:

- τ_I transforma I_A em uma instância I_B de B, **e**
- τ_S transforma uma solução S_B de I_B em uma solução S_A de I_A.

Reduções

Esquema básico de uma redução de Turing:

$$\begin{tikzcd}[column sep=huge]
A & B \\
I_A \arrow[r, "\tau_I"] & I_B \\
S_A & S_B \arrow[l, "\tau_S"']
\end{tikzcd}$$

Quando usar reduções?

- **Situação 1:** quero encontrar um algoritmo para resolver o problema A e conheço um **algoritmo** que resolve B, ou seja, determinar uma **cota superior** para o problema A;
- **Situação 2:** quero determinar uma **cota inferior** para o problema B e conheço uma **cota inferior** para o problema A.

Exemplo

Problema A: dados uma matriz M de posto completo e um vetor b, encontrar uma solução do sistema linear $Mx = b$.

Problema B: dados uma matriz quadrada simétrica P (isto é, $p_{ij} = p_{ji}$) e um vetor d, encontrar uma solução do sistema linear $Px = d$.

Suponha que temos um programa que resolve B e queremos saber como resolver A.

Podemos fazer uma redução de A para B.

- τ_I: compute a matriz $P = M^T M$ e o vetor $d = M^T b$.
- Álgebra Linear: Um vetor x é solução de $Mx = b$ se e somente se, x é solução de $M^T Mx = M^T b$, ou seja, $Px = d$.
- τ_S: é a função identidade.

Exemplo

- Um vetor x é solução de $Mx = b$ se e somente se é solução de $M^T Mx = M^T b$, ou seja, $Px = d$.
- A redução mostra como resolver o problema A, compondo a redução com o algoritmo que resolve o problema B.
- **Conclusão:** resolver sistemas lineares da forma $Px = b$ quando P é **simétrica** é pelo menos tão difícil quanto resolver um sistema linear $Mx = b$ em que M é uma matriz qualquer de posto completo.

Reduções

Definição: Um problema A é redutível a um problema B em tempo $f(n)$ se existe uma redução como esquematizada abaixo:

$$\begin{tikzcd}
A & & B \\
I_A \arrow[rr, "\tau_I"] & & I_B \\
S_A & & S_B \arrow[ll, "\tau_S"']
\end{tikzcd}$$

onde $n = |I_A|$ e, τ_I e τ_S custam $O(f(n))$.

Notação: $A \propto_{f(n)} B$.

Reduções

Definição: Um problema A é redutível a um problema B em tempo $f(n)$ se existe uma redução como esquematizada abaixo:

$$\begin{tikzcd}
A & B \\
I_A \arrow[r, "\tau_I"] & I_B \\
S_A & S_B \arrow[l, "\tau_S"']
\end{tikzcd}$$

onde $n = |I_A|$ e, τ_I e τ_S custam $O(f(n))$.

Se $f(n)$ for um polinômio então dizemos que (τ_I, τ_S) é uma redução polinomial de A a B e que A é polinomialmente redutível a B.

Observações

$$
\begin{tikzcd}
A & B \\
I_A \arrow[r, "\tau_I"] & I_B \arrow[d, bend left, "\pi_B"] \\
S_A & S_B \arrow[l, "\tau_S"']
\end{tikzcd}
$$

- Conhecendo um algoritmo π_B que resolve B, temos imediatamente um algoritmo π_A que resolve qualquer instância de A:

$$\pi_A = \tau_I \circ \pi_B \circ \tau_S.$$

- a complexidade de π_A é a soma das complexidades de τ_I, π_B e τ_S e deve ser expressa em função do tamanho de $n = |I_A|$. Isto fornece uma cota superior para A.

Observações

$$\begin{tikzcd} A & & B \\ I_A \arrow[rr, "\tau_I"] & & I_B \arrow[d, bend left, "\pi_B"] \\ S_A & & S_B \arrow[ll, "\tau_S"'] \end{tikzcd}$$

- Se π_B tem complexidade $O(g(n))$ (cota superior de B) e $g(n) \in \Omega(f(n))$ então $O(g(n))$ também é uma cota superior de A.
  - Se $g(n) \notin \Omega(f(n))$, a cota superior ainda vale?
    Não necessariamente.

Observações

$$\begin{tikzcd}
A & & B \\
I_A \arrow[rr, "\tau_I"] & & I_B \\
S_A & & S_B \arrow[ll, "\tau_S"']
\end{tikzcd}$$

- Se $\Omega(h(n))$ é uma cota inferior para o problema A e $f(n) \in o(h(n))$, então $\Omega(h(n))$ também é cota inferior para o problema B.
  - Por que temos a restrição de que $f(n) \in o(h(n))$?
  - Lembre-se que $o(h(n))$ e $\Omega(h(n))$ são disjuntos.

Observações

$$\begin{tikzcd}
A & & B \\
I_A \arrow[rr, "\tau_I"] & & I_B \\
S_A & & S_B \arrow[ll, "\tau_S"']
\end{tikzcd}$$

- Em uma redução **não** é necessário explicar **como resolver** o problema B, apenas como τ_I e τ_S funcionam
- a complexidade da **redução** é a soma das complexidades de τ_I e τ_S (ou equivalentemente, a maior das duas).

Reduções polinomiais

- Nesta disciplina estamos interessados em algoritmos polinomiais para resolver problemas.
- Escrevemos $A \propto_{\text{poli}} B$ se existe uma redução de custo polinomial de A para B e dizemos que A é polinomialmente redutível a B.
- Neste caso, dizemos que B é **pelo menos tão difícil quanto** A.
- **Motivação:** se B pode ser resolvido por um algoritmo polinomial, então A também pode. Equivalentemente, se A **não** pode ser resolvido em tempo polinomial, então B também **não** pode.
- Esta noção é muito importante no estudo da Teoria da Complexidade quando estudamos a aparente inexistência de algoritmos polinomiais para uma grande classe de problemas, os chamados problemas NP-difíceis/NP-completos.

Problema do casamento cíclico de strings (CSM)

Entrada: alfabeto Σ e strings sobre Σ de tamanho n:

$$A = a_0 a_1 \ldots a_{n-1} \text{ e } B = b_0 b_1 \ldots b_{n-1}.$$

Objetivo: decidir se B é um deslocamento cíclico de A.
Ou seja, existe $k \in \{0, 1, \ldots, n-1\}$ tal que $a_{(i+k) \bmod n} = b_i$ para todo $i = 0, 1, \ldots, n-1$?

Exemplo: para $A = acgtact$ e $B = gtactac$ $(n = 7)$ temos $k = 2$.

Como se resolve o CSM?

Exemplos de reduções

Problema do casamento de strings (SM)

Entrada: alfabeto Σ e strings sobre Σ:

$$A = a_0 a_1 \ldots a_{n-1} \text{ e } B = b_0 b_1 \ldots b_{m-1}, \text{com } m \leq n.$$

Objetivo: encontrar a primeira ocorrência de B em A ou concluir que B não é subcadeia de A.

Ou seja, determinar o menor índice $k \in \{0, 1, \ldots, n-1\}$ tal que $a_{(i+k) \bmod n} = b_i$ para todo $i = 0, 1, \ldots, m-1$ ou devolver $k = -1$.

Exemplo: para $A = acgttaccgtacccg$ e $B = tac$ ($n = 15$ e $m = 3$) temos $k = 4$.

Observação: o problema SM pode ser resolvido em tempo $O(n+m)$ pelo algoritmo KMP de Knuth, Morris and Pratt (1977).

CSM $\propto$ SM

Redução: CSM $\propto_n$ SM

- Instância de CSM: $I_{CSM} = (A, B, n)$.
- τ_I constrói a instância de SM:

$$I_{SM} = (A', 2n, B, n), \text{ onde } A' = A||A.$$

  Portanto, τ_I custa $O(n)$.
- Se k é a solução de SM para I_{SM}, então k também é a solução de I_{CSM}. Logo, τ_S custa $O(1)$ e a redução custa $O(n)$.

Exemplo:

- $I_{CSM} = (acgtact, gtactac, 7)$
- $I_{SM} = (acgtactacgtact, 14, gtactac, 7)$
- $S_{SM} = S_{CSM} = \{k = 2\}$

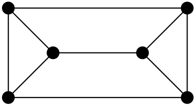
Exemplos de reduções

Problema da existência de triângulo (PET)

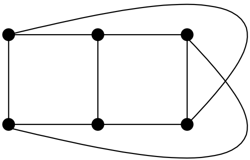
Entrada: grafo conexo simples $G = (V, E)$ com $n = |V|$ e $m = |E|$ na forma de matriz de adjacência.

Objetivo: decidir se G contém um triângulo.

Exemplo:



SIM



NÃO

Observações sobre o PET

- Há um algoritmo trivial de complexidade $O(n^3)$: verificar todas as triplas de vértices.
- Existe um algoritmo $O(mn)$ que é muito bom para grafos esparsos. (Exercício)
- Seja $A = A(G)$ a matriz de adjacência de G.
- Se $A^2 = A \times A$, então $a^2_{ij} = \sum_{k=1}^{n} a_{ik} a_{kj}$. Então

$$a^2_{ij} > 0 \Leftrightarrow \exists k \in \{1, \ldots, n\} \text{ tal que } a_{ik} = a_{kj} = 1.$$

- Portanto, (i, j, k) corresponde a um triângulo se, e somente se, $a^2_{ij} > 0$ e $a_{ij} = 1$.

Exemplos de reduções

Problema da Multiplicação de Matrizes Quadradas (MMQ)

Entrada: matrizes quadradas (de inteiros) A e B de ordem n.

Objetivo: calcular o produto $P = A \times B$.

Observações:

- há um algoritmo óbvio de complexidade $O(n^3)$;
- MMQ pode ser resolvido em tempo $O(n^{\log 7 = 2.807})$ pelo algoritmo de Strassen (1969) ou em tempo $O(n^{2.376})$ pelo algoritmo de Coppersmith e Winograd (1990).

PET $\propto$ MMQ

Redução: PET $\propto_{n^2}$ MMQ

- Instância de PET: $I_{PET} = A(G)$.
- τ_I constrói a instância de MMQ:

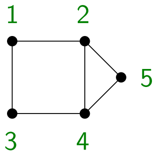
$$I_{MMQ} = (A, A, n), \text{ onde } A = A(G).$$

  Portanto, τ_I custa $O(n^2)$.

- Se $S_{MMQ} = P$ é a solução de MMQ para I_{MMQ}, então a solução de I_{PET} é obtida pelo algoritmo abaixo:

  para $i = 1$ **até** n **faça**
      para $j = 1$ **até** n **faça**
          se $p_{ij} > 0$ **e** $a_{ij} = 1$ **então devolva** SIM
  devolva NÃO

  Logo, τ_S custa $O(n^2)$.



$A(G)$

	1	2	3	4	5
1	0	1	1	0	0
2	1	0	0	1	1
3	1	0	0	1	0
4	0	1	1	0	1
5	0	1	0	1	0

$P = A(G) \times A(G)$

	1	2	3	4	5
1	2	0	0	2	1
2	0	3	2	1	1
3	0	2	2	0	1
4	2	1	0	3	1
5	1	1	1	1	2

Multiplicação de Matrizes Simétricas (MMS)

Entrada: matrizes simétricas (de inteiros) A e B de ordem n.

Objetivo: calcular o produto $P = A \times B$.

Observações:

- MMS é um caso particular de MMQ: a redução MMS $\propto_{n^2}$ MMQ é imediata;
  Portanto, MMQ é pelo menos tão difícil quanto MMS.
- Será que MMS é pelo menos tão difícil quanto MMQ?
  Menos óbvio.

MMQ $\propto$ MMS

Redução: MMQ $\propto_{n^2}$ MMS

- Instância de MMQ: $I_{MMQ} = (A, B, n)$.
- τ_I constrói a instância de MMS: $I_{MMS} = (A', B', 2n)$ onde

$$A' = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \quad \text{e} \quad B' = \begin{bmatrix} 0 & B^T \\ B & 0 \end{bmatrix}$$

  Portanto, τ_I custa $O(n^2)$.
- A solução de MMS é:

$$P' = A'B' = \begin{bmatrix} AB & 0 \\ 0 & A^T B^T \end{bmatrix}$$

MMQ $\propto$ MMS

- A função τ_S pode ser implementada pelo algoritmo abaixo:

  para $i = 1$ **até** n **faça**
      para $j = 1$ **até** n **faça**
          $p_{ij} \leftarrow p'_{ij}$ $\triangleright$ copia AB para P

  Logo, τ_S custa $O(n^2)$.

- Por esta redução, se MMQ tem cota inferior em $\Omega(h(n))$, então MMS também tem cota inferior em $\Omega(h(n))$.
- Note que $h(n) \in \Omega(n^2)$. (Por quê?)

  Uma cota inferior trivial para qualquer problema é o tamanho da entrada.

Você está entendendo mesmo?

Suponha que um problema A pode ser reduzido a um problema B.

$$
\begin{tikzcd}
A & B \\
I_A \arrow[r, "\tau_I"] & I_B \\
S_A & S_B \arrow[l, "\tau_S"']
\end{tikzcd}
$$

Suponha que $n = |I_A|$, $\tau_I, \tau_S \in O(n^2)$ e $|I_B| = O(n^2)$. Suponha também que uma instância de B com tamanho m pode ser resolvida em tempo $O(m \log m)$.

- A pode ser resolvido em tempo $O(n \log n)$? Não sei... Talvez?
- A pode ser resolvido em tempo $O(n^2 \log n)$? Sim!

Você está entendendo mesmo?

Suponha que um problema A pode ser reduzido a um problema B.

$$\begin{tikzcd}[column sep=huge] A & B \\ I_A \arrow[r, "\tau_I"] & I_B \\ S_A & S_B \arrow[l, "\tau_S"'] \end{tikzcd}$$

Agora suponha que $n = |I_A|$, $\tau_I, \tau_S \in O(n^2)$ e $|I_B| = O(n \log n)$.

- Se A tem cota inferior $\Omega(n^{1.7})$ então B também tem?

  Não sei... Talvez?
- Se A tem cota inferior $\Omega(n^2 \log n)$ então B também tem? Sim!

Observação. Note que a cota inferior de B está em função de $n = |I_A|$. Para obter a cota inferior correspondente de B em função de $|I_B|$, basta escrever n em função de $|I_B|$.

Erros comuns ao usar reduções

- Usar a redução na ordem inversa: em vez de fazer a redução $A \propto B$, prova-se que $B \propto A$ e conclui-se (erroneamente) que B é pelo menos tão difícil quanto A.
- Dada a redução $A \propto B$, achar que toda instância de B tem que ser mapeada em alguma instância de A. O mapeamento τ_I é injetor (não necessariamente bijetor).
- Usar o algoritmo produzido por uma redução sem se preocupar com a existência de outro mais eficiente. A redução de $A \propto B$ não é necessariamente o modo mais eficiente de resolver A.

Princípio da Chaleira

- Este princípio é uma espécie de anedota que satiriza o processo de pensamento que ocorre quando se trabalha com reduções.
- Nimrod tem uma receita antiga para fazer o melhor chá do Universo: ele pega uma chaleira **vazia**, enche-a de água, aquece-a até a água começar a ferver, desliga o fogo, acrescenta as folhas de chá, mexe levemente por 7 segundos, espera 42 segundos e pronto!
- Um dia caiu uma chuva fortíssima e Nimrod notou que a chaleira tinha 70% de água. O que ele deve fazer para preparar seu chá?
- Jogue fora toda a água e reduza ao problema anterior(!!).

Reduções

Definição: Um problema A é redutível a um problema B em tempo $f(n)$ se existe uma redução como esquematizada abaixo:

$$\begin{array}{ccc} A & & B \\ I_A & \xrightarrow{\tau_I} & I_B \\ S_A & \xleftarrow[\tau_S]{} & S_B \end{array}$$

onde $n = |I_A|$ e, τ_I e τ_S custam $O(f(n))$.

Notação: $A \propto_{f(n)} B$.

Reduções para obter cotas inferiores

$$\begin{array}{ccc} A & & B \\ I_A & \xrightarrow{\quad\tau_I\quad} & I_B \\ S_A & \xleftarrow[\quad\tau_S\quad]{} & S_B \end{array}$$

- Se $\Omega(h(n))$ é uma cota inferior para o problema A e $f(n) \in o(h(n))$, então $\Omega(h(n))$ também é cota inferior para o problema B.

Reduções para obter cotas inferiores

- Veremos algumas reduções que nos permitem obter cotas inferiores para vários problemas.
- Sejam A e B dois problemas. Suponha que A tem cota inferior $\Omega(h(n))$.
- Se $A \propto_{f(n)} B$ e $f(n) \in o(h(n))$, então B também tem cota inferior $\Omega(h(n))$.
- **Atenção!** Resultados sobre cota inferior **dependem** do **modelo de computação** adotado.

  Por exemplo, o Problema da Ordenação tem cota inferior $\Omega(n \log n)$ no **modelo de árvores binárias de decisão**.

Problema da Ordenação (ORD)

Entrada: sequência de elementos comparáveis de comprimento n

$$X = (x_1, x_2, \ldots, x_n).$$

Objetivo: encontrar uma **permutação ordenada** de X.

Observações:

- ORD tem cota inferior $\Omega(n \lg n)$ no modelo mais geral de **árvore algébrica de decisão**.
- Informalmente, neste modelo em cada nó é feita uma computação de um polinômio de n variáveis. Há 3 possíveis resultados > 0, $= 0$ ou < 0 (a árvore é ternária).
- Todos os resultados de cotas inferiores que veremos são para este modelo.

Problema da Unicidade de Elementos (UE)

Entrada: sequência de elementos comparáveis de comprimento n

$$X = (x_1, x_2, \dots, x_n).$$

Objetivo: decidir se todos os elementos de X são **distintos**.

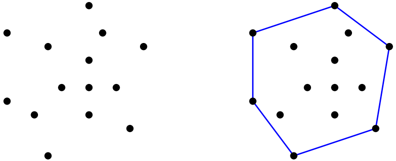
Observações:

- no **modelo de árvore algébrica de decisão**, UE tem cota inferior $\Omega(n \lg n)$.
- o problema pode ser resolvido em tempo $O(n \lg n)$. (Como?)

Problema da Envoltória Convexa (EC)

Entrada: conjunto $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ de n pontos no plano.

Objetivo: encontrar o **menor polígono convexo** que contém os n pontos.



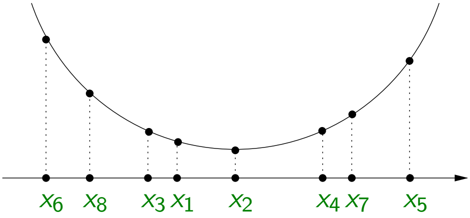
Observações:

- a saída é a ordem cíclica anti-horária dos vértices do polígono;
- problema clássico de Geometria Computacional: pode ser resolvido em tempo $O(n \lg n)$ usando a estratégia de divisão-e-conquista.

Redução: ORD $\propto_n$ EC

- Instância de ORD: $I_{ORD} = (x_1, x_2 \ldots, x_n)$.
- τ_I constrói a instância de EC:

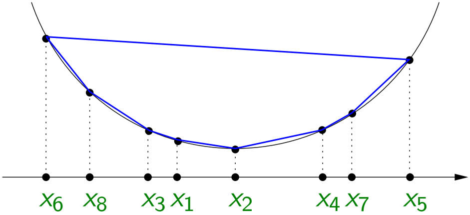
$$I_{EC} = \{(x_1, x_1^2), (x_2, x_2^2), \ldots, (x_n, x_n^2)\}.$$



Claramente τ_I custa $O(n)$.

ORD $\propto_n$ EC

- A solução de I_{EC} é uma ordem cíclica de pontos.
- τ_S determina o ponto que tem **menor abcissa** e lista os próximos pontos seguindo a ordem cíclica anti-horária.



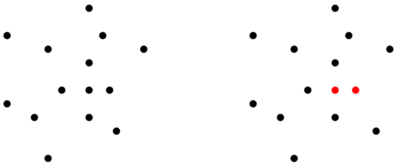
Claramente, τ_S custa $O(n)$.

- Segue que $\Omega(n \lg n)$ é uma cota inferior para EC.

Problema do Par Mais Próximo (PMP)

Entrada: coleção $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ de n pontos no plano.

Objetivo: encontrar um **par de pontos** que estejam **a menor distância**.



Observação:

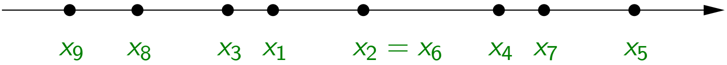
- problema clássico em Geometria Computacional: pode ser resolvido em tempo $O(n \lg n)$.

Redução: UE $\propto_n$ PMP

- Instância de UE: $I_{UE} = (x_1, x_2, \ldots, x_n)$.
- τ_I constrói a instância de PMP:

$$I_{PMP} = \{(x_1, 0), (x_2, 0), \ldots, (x_n, 0)\}.$$

Claramente, τ_I custa $O(n)$.

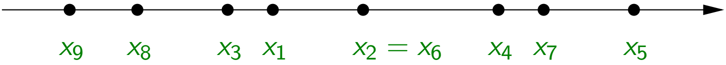


UE $\propto_n$ PMP

- A solução de I_{PMP} é um par de pontos $(x_i, 0), (x_j, 0)$.
- τ_S verifica se a **distância** entre os dois pontos é **zero**.

  Se SIM então a resposta de I_{UE} é NÃO.
  Caso contrário, a resposta de I_{UE} é SIM.

  Claramente, τ_S custa $O(1)$.



- Logo, $\Omega(n \lg n)$ é uma cota inferior para PMP.

Problema da 3-Soma (3SUM)

Entrada: sequência $X = (x_1, x_2, \ldots, x_n)$ de reais.

Objetivo: determinar se existem índices distintos i, j e k tais que:

$$x_i + x_j + x_k = 0.$$

Exemplo: $X = (0.75, -0.4, 1, -1, 0.25, -0.7)$
solução $i = 1$, $j = 4$ e $k = 5$.

Observações:

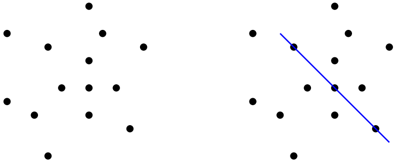
- o problema pode ser resolvido em tempo $O(n^2)$ (Como?)
- por muito tempo acreditou-se que $\Omega(n^2)$ fosse uma cota inferior para 3SUM, mas ninguém sabia provar isto!
  A única cota inferior conhecida era o trivial $\Omega(n)$.

Colinearidade

Problema da Colinearidade (COL)

Entrada: conjunto $S := \{(x_1, y_1), \ldots, (x_n, y_n)\}$ de n pontos no plano.

Objetivo: determinar se **três pontos distintos** de S são **colineares**.



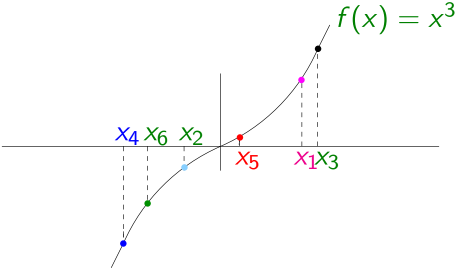
Observação:

- o problema pode ser resolvido em tempo $O(n^2)$
- acredita-se que $\Omega(n^2)$ é uma cota inferior para COL, mas ninguém sabe provar isto!

Redução: 3SUM $\propto_n$ COL

- Instância de 3SUM: $I_{3SUM} = (x_1, x_2, \ldots, x_n)$.
- τ_I constrói a instância de COL:

$$I_{COL} = \{(x_i, x_i^3) : i = 1, 2, \ldots, n\}.$$

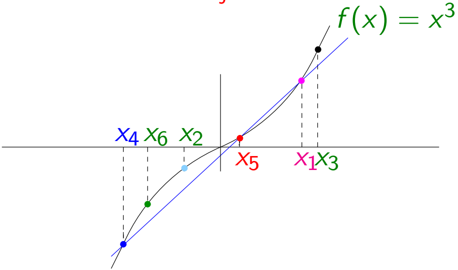


Exemplo: $X = (0.75, -0.4, 1, -1, 0.25, -0.7)$

Claramente, τ_I custa $O(n)$.

3SUM $\propto_n$ COL

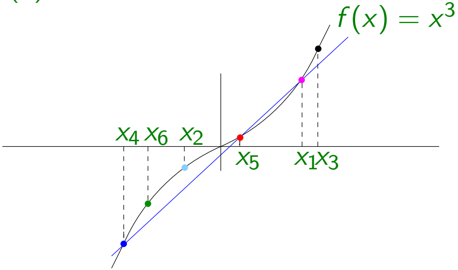
- A solução de I_{COL} (se houver) é uma tripla de pontos colineares distintos $(x_i, x_i^3), (x_j, x_j^3), (x_k, x_k^3)$.
- neste caso, τ_S devolve x_i, x_j, x_k, senão devolve NÃO EXISTE.
- Vetores e geometria: três pontos $(x_i, x_i^3), (x_j, x_j^3), (x_k, x_k^3)$ são **colineares** se, e somente se, $x_i + x_j + x_k = 0$.



Exemplo: $X = (0.75, -0.4, 1, -1, 0.25, -0.7)$

3SUM $\propto_n$ COL

- Logo, τ_S custa $O(1)$,



Exemplo: $X = (0.75, -0.4, 1, -1, 0.25, -0.7)$

3SUM $\propto_n$ COL

- Como a redução descrita é **linear**, **se** $\Omega(h(n))$ é uma cota inferior de 3SUM, **então** $\Omega(h(n))$ também é uma cota inferior de COL.

  Entretanto, a única cota inferior conhecida é o trivial $\Omega(n)$.

- Surpreendentemente Grønlund e Pettie (2014) mostraram que 3SUM pode ser resolvido em tempo $O(n^2/(\log n/\log\log n)^{2/3})$.

  Originalmente, conjecturava-se que $\Omega(n^2)$ era uma cota inferior para 3SUM. Sabe-se agora que isto não é verdade.

- **Problema em aberto:** existe uma constante $\epsilon > 0$ tal que 3SUM tem cota inferior $\Omega(n^{2-\epsilon})$?

Exercício

O Problema 3SUM' consiste em dados uma sequência $X = (x_1, x_2, \ldots, x_n)$ de reais e um real b, determinar se existem três índices distintos i, j e k tais que $x_i + x_j + x_k = b$.

1. Mostre que 3SUM $\propto_n$ 3SUM'.
2. Mostre que 3SUM' $\propto_n$ 3SUM.
3. Suponha que o Professor Sabit Udo provou uma cota inferior $\Omega(n^{1.9})$ para 3SUM'.

   Quais das afirmações abaixo podemos concluir que são verdadeiras a partir deste fato?

   a. Não existe algoritmo $O(n^{1.5})$ para 3SUM'.
   b. Não existe algoritmo $O(n^{1.5})$ para 3SUM.
   c. Existe um algoritmo $O(n^{1.9})$ para 3SUM'.
   d. Existe um algoritmo $O(n^{1.9})$ para 3SUM.
   e. 3SUM e 3SUM' têm a mesma cota superior e inferior.

Exercício

Prof. Sabin Ada afirma que desenvolveu uma estrutura de dados (chamada 2Good2BTru) que armazena um conjunto S de números reais e é capaz de realizar as seguintes operações:

- inicializar S como vazio em tempo $O(1)$,
- inserir um número real x em S em tempo $O(1)$ e
- remover (e devolver) o menor elemento de S em tempo $O(1)$.

O que você acha?