Algoritmos aleatorizados

e desaleatorização de algoritmos

Armando Matos

`EDA: só páginas 1-14, 20-39`

Plano

- Algoritmos com aleatórios
- Classes de complexidade
- Exemplos: “matching perfeito”, primalidade, corte máximo
- Desaleatorização
- Trabalhos

Algoritmos aleatorizados

Porquê algoritmos com aleatórios?

- Algoritmos mais simples que os algoritmos determinísticos equivalentes (muitos exemplos).
- Algoritmos/eficiências que não se conseguem determinísticamente.
- (Sobretudo juntamente com interacção) Conseguir coisas que parecem impossíveis.

Porque teria interesse determinizar algoritmos aleatórios?

Testemunhas

Classe NPTIME:

$$X \in NPTIME \longrightarrow \langle f(\cdot,\cdot),\, q(\cdot)\rangle$$

tal que $f(x, y)$ é computável em tempo polinomial em $|x|$ e

$$X = \{x \;:\; \exists y,\, |y| \leq q(|x|),\, f(x, y)\}$$

x: instância
y: testemunho

Exemplo: Números compostos

Problema: $x \in \mathcal{N}$ é composto? (=não é primo?)

Testemunha: d que divide x com $2 \leq d < x$

Algoritmo (mau algoritmo):

```
Dados: x
Pergunta: x é composto?
boolean primo(x){
  d <- aleatório(2,x-1);
  if(x%d==0)
    return(SIM);
  return(NÃO);
}
```

Exemplo: Números compostos

1. O erro, probabilidade de falhar
2. Repetir o processo (um número polinomial de vezes)
3. Divisores: Maus testemunhos de composicionalidade (pouco frequentes)
4. Veremos melhores testemunhos de composicionalidade

Se há muitas Testemunhas...

Suponhamos que, para cada x existe um espaço finito $A \subset \{0,1\}^\star$ tal que

$$\begin{cases} x \notin X & \Rightarrow \quad \forall y \in A,\ \neg f(x,y) \text{ (Claro!)} \\ x \in X & \Rightarrow \quad |\{y \ : \ f(x,y)\}| \geq |A|/2 \end{cases}$$

(Em vez do factor $1/2$ qualquer ε com $0 < \varepsilon \leq 1$, serve)

RP

...um algoritmo probabilístico é simples

```
decide(x: instância){
   y ← random(A) /* uniformemente */
   if f(x,y)
      return(YES);
   else
      return(NÃO);
}
```

$$\begin{cases} x \notin X & \text{Resposta correcta} \\ x \in X & \text{Probabilidade de errar} \leq 1/2 \end{cases}$$

... com pequeníssima probabilidade de errar

```
decide(x: instância){
   for i=1 to 500 {
      y ← random(A) /* uniformemente */
      if f(x,y)
         return(YES);
   }
   else
      return(NÃO);
}
```

. . . da pureza da Matemática. . .

Probabilidade de errar ($x \in S$ e dizer `NÃO`) é $\leq 1/2^{500}$

Probabilidade de num determinado microsegundo cair um meteorito em cima do computador e destruí-lo é incrivelmente superior a $1/2^{500}$!

Se só existem testemunhos de que $x \notin S$

(Possivelmente alguns problemas co-NPC)

$$\begin{cases} x \in X & \text{Resposta correcta} \\ x \notin X & \text{Probabilidade de errar} \leq 1/2 \end{cases}$$

Algoritmo semelhante

co-RP

... Se existem testemunhos de $x \in S$ e de $x \notin S$

Exemplo: primalidade!

```
decide(x: instância){
   while TRUE{
      y ← random(A)
   if f(x,y)
      return(YES);
   if g(x,y)
      return(NÃO);
   }
}
```

RP∩co-RP

. . . O algoritmo não erra. . .

$$\begin{cases} x \in X & \text{Resposta correcta} \\ x \notin X & \text{Resposta correcta} \end{cases}$$

. . . e demora tempo polinomial *só em média!*

Agora temos certeza (ignorando os meteoritos) mas, no máximo, temos de esperar. . . glup!

$\boxed{\textbf{Classes de complexidade aleatorizadas}}$

Classes de complexidade

$X \in$NPTIME: dado por $f(\cdot,\cdot)$ e $p(\cdot)$ (polinómio):

$$x \in X \Leftrightarrow \exists y, |y| \leq p(|x|),\ f(x,y)$$

$X \in$RP: dado por $f(\cdot,\cdot)$, $p(\cdot)$ e $D(x)$

$$\begin{aligned} x \in X \quad &\Rightarrow \quad |\{y\ :\ y \in D(x),\ f(x,y)\}| \geq |D(x)|/2 \\ x \notin X \quad &\Rightarrow \quad |\{y\ :\ y \in D(x),\ f(x,y)\}| = 0 \end{aligned}$$

$X \in$co-RP: ao contrário

Classe de complexidade BPP

$X \in$BPP

$$\begin{array}{lcl} x \in X & \Rightarrow & \text{pelo menos } 3/4 \text{ de testemunhos} \\ x \notin X & \Rightarrow & \text{pelo menos } 3/4 \text{ de testemunhos} \end{array}$$

BPP: a classe mais geral usualmente considerada “eficiente” (“feasible”).

Exemplos

Problema	Classe	de facto...
Primalidade	co-RP	RP$\cap$co-RP
Composicionalidade	RP	RP$\cap$co-RP

BPP: a classe mais geral usualmente considerada "eficiente" ("feasible").

Algumas inclusões

$$\text{PTIME} \;\subseteq\; \text{RP} \cap \text{co-RP} \;\subseteq\; \text{RP} \cup \text{co-RP} \;\subseteq\; \text{BPP}$$

São estritas?

Figurinha:

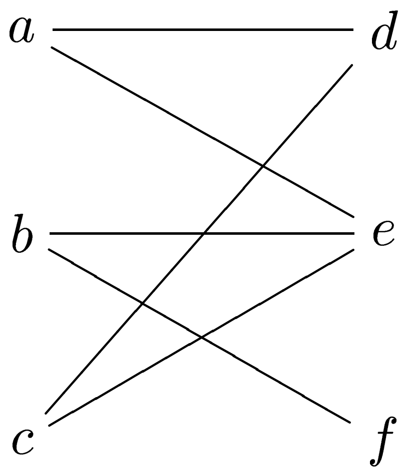
$\boxed{\textbf{Exemplo de problema}}$

"Matching" perfeito

INSTÂNCIA: Um grafo bipartido $G = (A \cup B, E)$
com $|A| = |B|$ e $E \subseteq A \times B$.

PERGUNTA: Existe $S \subseteq E$, $|S| = |A|$,
$[\forall a \in A,\ \exists b,\ (a, b) \in E] \wedge [\forall b \in B,\ \exists a,\ (a, b) \in E]$

“Matching perfeito” – exemplo



Resposta Sim. Exemplo: $\{(a, e), (b, f), (c, d)\}$

“Matching perfeito” – equação equivalente

$$M_G = \begin{bmatrix} x_{ad} & x_{ae} & 0 \\ 0 & x_{be} & x_{bf} \\ x_{cd} & x_{ce} & 0 \end{bmatrix}$$

$$\boxed{\text{A resposta é } \textsc{Sim} \text{ sse } |M_G| \neq 0}$$

“Matching perfeito” – calcular o determinante!

$$|M_G| = -x_{ad}x_{ce}x_{bf} - x_{cd}x_{ae}x_{bf} \neq 0$$

logo tem solução, existe “matching”.

Um pequeno problema

O determinante pode ter um tamanho exponencial (no número de linhas). O algoritmo não é polinomial.

Não desistamos

Ideias para uma solução

1. Não necessitamos de calcular o determinante, só saber se é 0.
2. Um determinante sem variáveis pode ser calculado em tempo polinomial (ex: método de Gauss).
3. Seja m o número de variáveis. Seja A um inteiro apropriado. Escolhemos aleatóriamente e uniformemente valores para as variáveis

$$\overline{x} \in_U = \{0, 1, \cdots, A-1\} \quad \langle n\text{-tuplo de variáveis}\rangle$$

4. Se $|M_G(\overline{x})| \neq 0$, a resposta é EXISTE
Se $|M_G(\overline{x})| = 0$, a resposta é NÃO SEI

O A apropriado

Um polinómio de grau d tem no máximo d raízes inteiras em $A = \{0, 1, \cdots, 2d-1\}$ (aliás raizes reais em qualquer intervalo). A probabilidade de $x \in_u A$ ser raíz é pois $\leq 1/2$.

Generalizando...
Seja $p(\overline{x})$ um polinómio não nulo, cada uma das m variáveis tendo, no máximo, grau d. O número de m-tuplos $\overline{x} \in \{0, \cdots, A-1\}^m$ tais que $p(\overline{x}) = 0$ é $\leq mdA^{m-1}$.

$A^m \geq 2mdA^{m-1}$, $A \geq 2md$. Como $d = 1$, tomamos $A = 2m$. O resto da história é igual...

$$\boxed{\textbf{Exemplo de problema}}$$

Corte máximo

Em vez de MAX-CUT seja <u>BIG-CUT</u>

INSTÂNCIA: : Um grafo não dirigido, (V, E).

PERGUNTA: : Existe uma partição (ou coloração) de V em 2 partes, V e W tal que, pelo menos metade dos ramos tenham extremidades em partes distintas? Se sim, encontre uma.

Corte máximo

Coloração: $c: V \to \{0,1\}$.

Consideremos: coloração uniforme c (com igual probabilidade, 1/2) dos vértices.

Qual o número médio de ramos cortados?

$$\Sigma_{(a,b)\in E} \Pr[c(a) \neq c(b)] = \frac{|E|}{2}$$

Portanto, qualquer que seja o grafo, existe uma solução!

Mais: uma coloração aleatória, uniforme, fornece com probabilidade $\geq 1/2$ uma solução!

Mais: não se conhece nenhum algoritmo determinístico polinomial simples para este problema.

$\boxed{\textbf{Exemplo de problema}}$

Primalidade - testemunhas

Seja n um inteiro não primo.

Uma testemunha de que n é composto é um factor x de n.

COMPOSTO$\in$NPTIME

Mas as testemunhas podem ser poucas; por exemplo, $n = p^2$ com p primo só tem a testemunha p; ex: 121 só tem a testemunha 11.

Primalidade - muitas testemunhas

Seja n um inteiro e $1 \leq a \leq n$. Se n é primo é

$$a^{n-1} = 1 \mod n \qquad \text{(pequeno Teorema de Fermat)}$$

Portanto um valor de a tal que $a^{n-1} \neq 1 \mod n$ é um *testemunho* de que n é composto:

$$[a^{n-1} \neq 1 \mod n] \;\Rightarrow\; n \text{ é composto}$$

Exercício: calcular para $n = 5$ e para $n = 6$ todos os valores $a^{n-1} \mod n$. ◇

Exercício: Mostre que o valor de $a^{n-1} \mod n$ pode ser calculado em tempo polinomial em n.

ALGORITMO APROPRIADO / TRABALHAR SEMPRE "MÓDULO n"

Algoritmo polinomial para $a^{n-1} \bmod n$

```
power(a,m)
   if m = 0
      return 1;
   t ← power(a,m/2);
   if m%2= 0
      return t * t mod n;
   else
      return t * t * a mod n
```

Dos testemunhos de composicionalidade

Temos, módulo 5, se não me enganei. . .

$$1^4 = 1,\ 2^4 = 16 = 1,\ 3^4 = 81 = 1,\ 4^4 = 256 = 1$$

Logo 5 é primo

e, módulo 6, se não me enganei. . .

$$1^5 = 1,\ 2^5 = 32 = 2,\ 3^5 = 243 = 3,\ 4^5 = 1024 = 4,\ 5^5 = \dots$$

Logo 6 não é primo (bastava $2^5 \neq 1$, módulo 6).

Da frequência dos testemunhos $|T|/(n-2)$

3	0.0%
4	100.0%
5	0.0%
6	100.0%
7	0.0%
8	100.0%
9	85.7%
10	100.0%
11	0.0%
12	100.0%
13	0.0%
14	100.0%
15	76.9%

Os chatos

n composto $\Rightarrow$ $|T| \geq 1/2$?

Ninguém conseguiu demonstrar

(porque não é verdade!)

558	100.0%	
559	93.7%	
560	100.0%	
561	42.9%	Primeiro inteiro de Carmichael
562	100.0%	
563	0.0%	
564	100.0%	
565	97.3%	
566	100.0%	
567	99.5%	
568	98.9%	

Para todos os divisores primos p de n, $p-1|n-1$.

As Novas Testemunhas

Teste mais complexo, mas polinomial.

Teste de Rabin-Miller (cada composto tem pelo menos 1/4 de testemunhas entre 2 e $n-1$) :
Seja n ímpar com $n = 2^r s + 1$, s ímpar e seja $a \in_u \{1, \cdots, n-1\}$.

```
se a^s = 1 (mod n) ou
    a^(2^j) s = -1 (mod n) para algum 0 ≤ j ≤ r-1
        então n é primo
senão n é provavelmente composto
```

$$\texttt{se } a^s = 1 \ (\texttt{mod } n) \texttt{ ou } \quad a^{2^j} s = -1 \ (\texttt{mod } n) \texttt{ para algum } 0 \le j \le r-1$$

$\boxed{\textbf{Desaleatorização}}$

Corte máximo outra vez

$$
\begin{aligned}
&= \Pr[c(a) \neq c(b)] \\
&= \Pr[c(a) = 0 \wedge c(b) = 1] + \Pr[c(a) = 1 \wedge c(b) = 0] \\
&= \tfrac{1}{2} \times \tfrac{1}{2} + \tfrac{1}{2} \times \tfrac{1}{2} \\
&= \tfrac{1}{2}
\end{aligned}
$$

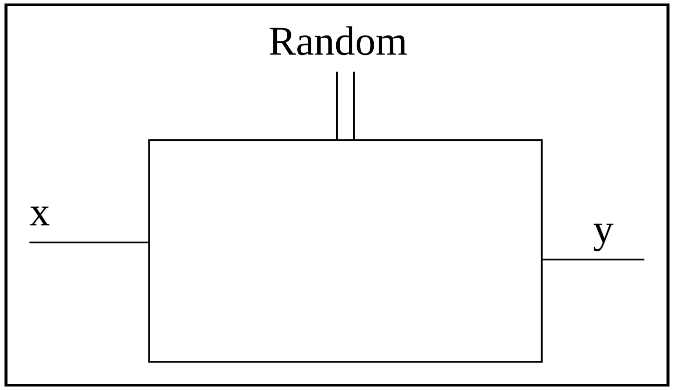
Só é utilizada a independência 2 a 2, não a independência total:

$$\Pr[A_1 \wedge A_2 \wedge \cdots \wedge A_n] = \Pr[A_1] \times \Pr[A_2] \times \cdots \times \Pr[A_n]$$

Isto vai ser essencial para permitir a desaleatorização.

Algoritmos aleatorizados

Simples, mas podem errar; depende da classe.



Exemplos:

Primalidade

Corte máximo

etc., etc..

Problemas

- Podem errar, embora o erro se possa reduzir arbitrariamente.
- Os aleatórios verdadeiros não são fáceis de obter.
- Os programadores não costumam gostar de probabilidades…

Um modo de determinizar

Tornar os algoritmos determinísticos:

> `Se os aleatórios são de` r `bits, fazer um ciclo pelos` 2^r `valores possíveis e procurar uma solução.`

Mas. . . fica geralmente um algoritmo que demora um tempo exponencial, glup, `^C`

Como, às vezes, se consegue polinomial

Basta "ciclar" um pequeno subconjunto de $\{0,1\}^r$ com propriedades apropriadas.

Ir ao algoritmo e procurar as propriedades necessárias dos aleatórios.

Independência das variáveis!

Sejam A_i variáveis aleatórias e a_i valores que tomam.

Convenção: $\Pr[a_i]$ significa $\Pr[A_i = a_i]$

Independência de pares de variáveis aleatórias: para todos os pares distintos de variáveis A_i e A_j é

$$\Pr[a_i \wedge a_j] = \Pr[a_i] \times \Pr[a_j]$$

Independência (total): para todos os n-tuplos de variáveis aleatórias distintas é

$$\Pr[a_1 \wedge a_2 \wedge \cdots \wedge a_n] = \Pr[a_1] \times \Pr[a_2] \times \cdots \times \Pr[a_n]$$

Exemplo...

Prob.	Ind. hash	Var. aleatórias		
	s	Z_1	Z_2	Z_3
1/4	00	0	0	0
1/4	01	0	1	1
1/4	10	1	0	1
1/4	11	1	1	1

Independência aos pares $\not\Rightarrow$ independência total

$$\mathcal{S} = \{00, 01, 10, 11\},\ U = \{Z_1, Z_2, Z_3\},\ T = \{0, 1\}$$

Índice do hashing com distribuição uniforme

$$\Pr(00) = \Pr(01) = \Pr(10) = \Pr(11) = 1/4$$

$$\Pr[Z_1 = 0 \wedge Z_2 = 0] = \ldots = \Pr[Z_2 = 1 \wedge Z_3 = 1] = 1/4$$

$$\Pr[Z_1 = 0] = \ldots = \Pr[Z_3 = 1] = 1/2$$

Em particular (colisão)

$$\Pr[Z_i = Z_j] = 1/2 = 1/|T|$$

(continuação)

$$\Pr[Z_1 = 0 \wedge Z_2 = 0 \wedge Z_3 = 1] = 0$$
$$\neq$$
$$\Pr[Z_1 = 0] \times \Pr[Z_2 = 0] \times \Pr[Z_3 = 1]$$
$$= (1/2)(1/2)(1/2)$$
$$= 1/8$$

Hasing universal do tipo 2

Para cada $s \in \mathcal{S}$, uma função $h_s : U \to T$.
No exemplo, $U = \{1, 2, 3\}$, $T = \{0, 1\}$.

Hasing universal do tipo 2 (2-universal hashing):

$$x \neq y \Rightarrow \Pr[h_s(x) = \alpha \wedge h_s(x) = \beta] = 1/|T|^2 \text{ (=1/4 no exemplo)}$$

Resulta para $x \neq y$

$$\Pr[h_s(x) = h_s(y)] = 1/|T|$$

Uma construção do "2-universal hashing"

$$\mathcal{S} = \{0,1\}^n \times \{0,1\}^n, \text{ seja } (a,b) \in \mathcal{S},\ x \in \{0,1\}^n$$

Define-se

$$h_s(x) = ax + b \quad (\mathrm{GF}[2^n])$$

Se quisermos contradomínio 2^k com $k < n$, tomamos os primeiros k bits de $h_s(x)$.

Nota: Só 2^{2n} funções de hashing de um total de 2^{2^n}, logaritmicamente menos!

(continuação)

Recordando

	$Z_1 \quad \cdots \quad Z_{2^n}$
s_1	valores
$\cdots$	de
$s_{2^{2n}}$	0 a $2^n - 1$

“2-universal hashing”

$$\begin{bmatrix} h_s(x) \\ h_s(y) \end{bmatrix} = \begin{bmatrix} x & 1 \\ y & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \mod 2^n$$

Se $x \neq y$, matriz não singular: para cada $x, y \in \{0,1\}^n$, $(h_s(x), h_s(y))$ toma todos os 2^{2n} valores.

Se $s \in_u \mathcal{S}$, $(h_s(x), h_s(y)) \in_u \{0,1\}^n \times \{0,1\}^n$.
É universal do tipo 2!=2

Total de
$2^{2^4} = 65\,536$
funções

"Hashing universal":
$2^{2\times 4} = 256$
funções

Aplicação ao corte e costura

Lembram-se? Seja $|V| = 2^n$. Em vez de ciclar através de todas as funções

$$V \to \{0,1\}, \ \ |V| = 2^n$$

(em número de 2^{2^n}), usamos o "2-universal hashing"que só tem $|V|^2$ funções. O algoritmo fica polinomial!!!

```
grande-corte((V,E))
   for s ← 1 to 2^{2n}{
      Considere a coloração h_s|_2
      if tamanho do corte ≥ |E|/2
         return h_s;
   }
   return ‘‘não existe’’
    /* Isto nunca acontece /*
```

$\boxed{\textbf{Trabalhos}}$

Trabalho – Primalidade

Estudar detalhadamente o que se sabe sobre os algoritmos de primalidade.

Trabalho teórico (relacionado, por exemplo, com a análise complexa) embora, dependendo da orientação que se queira dar, possa ter implementações práticas (por exemplo, algoritmo do Rabin com múltipla precisão).

Trabalho – Geradores de pseudo-aleatórios

Estudar detalhadamente os geradores de pseudo-aleatórios (PRG, no sentido criptográfico); por exemplo, o método linear congruente não pode ser considerado um PRG!

- Distâncias estatísticas e indistinguibilidade computacional.
- Técnicas utilizadas na construção de PRGs.
- Medidas de aleatoridade aparente e entropia computacional.

Trabalho – Classes de complexidade

O uso de aleatórios parece simplificar os algoritmos ou tornar possível algoritmos eficientes.

- É verdade?
- Qual o estado actual da "arte"?
- Que problemas interessantes se conhecem em cada classe?
- Que algoritmos se conseguiram desaleatorizar?

Trabalho – Aplicações do "hashing" universal

Aplicações do "hashing" universal (independência de pares) à Ciência de Computadores:

1. Desaleatorização.
2. "Hashing" perfeito
3. Amplificacacação determinística (pseudo-aleatórios)

... ...

486 Trabalhos de Seminários.

O que se pretende?

- Trabalho de síntese, não necessariamente com ideias novas – se as houver, óptimo!
- Esforço de pesquisa bibliográfica, compreensão dos artigos, estudos de "background" necessários...
- A compreensão pode eventualmente ser facilitada com, por exemplo, trabalhos de programação.
- Obter uma visão pessoal da matéria: estruturar o assunto, possivelmente relacioná-lo com outras áreas de CC, mencionar possíveis aplicações.

Alguma bibliografia para os trabalhos

Motwani and Raghavan, *Randomized Algorithms*

Papadimitriou, *Computational Complexity*

Luby and Wigderson, *Pairwise Independence and Derandomization*

Luby, *Pseudorandomness and Cryptographic Applications*

Saks, *Randomization and Derandomization in Space-Bounded Computation*

Cursos diversos espalhados pela Internet.